aws

用户指南

Amazon Neptune

Amazon Neptune: 用户指南

Table of Contents

什么是 Amazon Neptune？

Amazon Neptune 是一项快速、可靠且完全托管式的图数据库服务，可帮助您轻松构建和运行适用于高度互连数据集的应用程序。Neptune 的核心是一个专门打造的高性能图形数据库引擎。该引擎经过优化，可存储数十亿个关系并能以毫秒级延迟进行图形查询。Neptune 支持常见的属性图查询语言 Apache TinkerPop Gremlin 和 Neo4J 的 openCypher，也支持 W3C 的 RDF 查询语言 SPARQL。这可让您构建能够高效地浏览高度互联的数据集的查询。Neptune 支持图形用例，如建议引擎、欺诈检测、知识图谱、药物开发和网络安全。

Neptune 数据库具有高可用性，并提供只读副本、时间点恢复、到 Amazon S3 的持续备份以及跨可用区的复制。Neptune 提供了数据安全特征，并支持加密静态数据和传输中的数据。Neptune 是完全托管式的，因此，您再也无需担心数据库管理任务，例如硬件配置、软件修补、设置、配置或备份。

Neptune Analytics 是一种分析数据库引擎，它补充了 Neptune 数据库，可以快速分析内存中的大量图表数据，以获得见解和发现趋势。Neptune Analaytics 是一种用于快速分析存储在数据湖中的现有图形数据库或图形数据集的解决方案。它使用流行的图形分析算法和低延迟分析查询。

要了解有关使用 Amazon Neptune 的更多信息，我们建议您从以下几个部分入手：

- Amazon Neptune 入门
- Amazon Neptune 特征概述

如果您不熟悉图形，或者还没有准备好投资完整的 Neptune 生成环境，请访问我们的 开始使用 主题，了解如何在不产生成本的情况下使用 Neptune Jupyter 笔记本进行学习和开发。

此外，在您开始设计数据库之前，我们还建议您参考 GitHub 存储库用于图形数据库的 AWS 参考架构，在其中您可以了解有关图形数据模型和查询语言选择的信息，并浏览参考部署架构的示例。

关键服务组件

- 主数据库实例 – 支持读取和写入操作，并执行针对集群卷的所有数据修改。每个 Neptune 数据库集群均有一个主数据库实例，负责写入（即，加载或修改）图形数据库内容。
- Neptune 副本 – 连接到同一存储卷作为主数据库实例并仅支持读取操作。除主数据库实例之外，每个 Neptune 数据库集群最多可拥有 15 个 Neptune 副本。这样，通过将 Neptune 副本放在单独的可用区中并分配来自读取客户端的负载，可以实现高可用性。
- 集群卷 – Neptune 数据存储在集群卷中，旨在实现可靠性和高可用性。集群卷由跨单个 AWS 区域中的多个可用区的数据副本组成。由于您的数据会自动跨可用区复制，因此具有高持久性，数据丢失的可能性较小。

支持开放图谱 API

Amazon Neptune 对于属性图（Gremlin 和 openCypher）和 RDF 图 (SPARQL) 支持开放图形 API。它为图形模型及其查询语言提供了高性能。您可以选择属性图 (PG) 模型，然后使用 openCypher 查询语言和/或 Gremlin 查询语言访问同一个图形。如果您使用 W3C 标准资源描述框架 (RDF) 模型，则可以使用标准 SPARQL 查询语言访问图形。

高度安全

Neptune 可以为您的数据库提供多级安全保护。安全特征包括使用 Amazon VPC 进行网络隔离，使用您通过 AWS Key Management Service (AWS KMS) 创建和控制的密钥进行静态加密。在加密的 Neptune 实例上，底层存储中的数据会被加密，位于同一集群中的自动备份、快照和副本也会被加密。

完全托管

使用 Amazon Neptune，您不必担心数据库管理任务，例如硬件预配置、软件修补、设置、配置或备份。

您可以使用 Neptune 创建可在数毫秒内查询数十亿个关系的先进的交互式图形应用程序。为了提高性能而调整针对高度互连数据的 SQL 查询既复杂又困难。利用 Neptune，您可以使用常用图形查询语言 Gremlin、openCypher 和 SPARQL 执行功能强大的查询，此类查询易于编写并且非常适用于互连数据。此功能大幅降低代码复杂性，这样您可以快速地创建用于处理关系的应用程序。

Neptune 的设计旨在提供高于 99.99% 的可用性。它通过将数据库引擎与为数据库工作负载构建的 SSD 支持型虚拟化存储层紧密集成，来提高数据库的性能和可用性。Neptune 存储具有容错和自我修复能力。磁盘故障可在后台修复，而不会丢失数据库可用性。Neptune 将自动检测数据库崩溃并重新启动，无需进行崩溃恢复或重新构建数据库缓存。如果整个实例发生故障，Neptune 会自动向最多 15 个只读副本中的一个进行失效转移。

Amazon Neptune 的更改和更新

下表介绍了对 Amazon Neptune 的一些重要更改。

变更	说明	日期
引擎版本 1.3.2.1	截至2024年6月20日，引擎版本1.3.2.1已全面部署。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.3.2.1。	2024年6月20日
引擎版本 1.3.2.0	自 2024 年 6 月 10 日起，引擎版本 1.3.2.0 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.3.2.0。	2024年6月10日
引擎版本 1.2.1.1	自 2024 年 3 月 11 日起，引擎版本 1.2.1.1 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.1。	2024 年 3 月 11 日
引擎版本 1.3.1.0	自 2024 年 3 月 6 日起，引擎版本 1.3.1.0 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.3.1.0。	2024 年 3 月 6 日
更新到 AWS 托管策略权限	`NeptuneReadOnlyAccess` 和`NeptuneFullAccess` 托管策略现在	2024 年 1 月 22 日

	将Sid（声明 ID）作为标识符包含在策略声明中。	
Neptune 现在提供 I/O 优化型存储	使用 I/O 优化型存储，您只需为正在使用的存储和实例付费。存储成本高于标准存储，但您无需为所使用的 I/O 支付任何费用。	2023 年 12 月 13 日
Neptune 的 IAM 托管策略更改	NeptuneConsoleFullAccessIAM 托管策略已更新，授予与 Neptune Analytics 图表交互所需的权限，添加了新的NeptuneGraphReadOnly访问策略以提供对 Neptune Analytics 图表资源的只读访问权限，并添加了一项新AWSServiceRoleForNeptuneGraphPolicy 政策，允许海王星分析图表发布 CloudWatch 操作和使用指标及日志。	2023 年 11 月 29 日
引擎版本 1.3.0.0	截至 2023 年 11 月 15 日，引擎版本 1.3.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune Engine Release 1.3.0.0。	2023 年 11 月 15 日
Neptune 在以色列（特拉维夫）区域推出	Amazon Neptune 现已在以色列（特拉维夫）(il-central-1 ）区域推出。	2023 年 11 月 13 日

关于在 Neptune 属性图 time-to-live 中实现的博客文章	请参阅 Melissa Kwok、Mike Havey 和 Kevin Phillips 撰写的在 Amazon Neptune 中实现生存时间，第 1 部分：属性图。	2023 年 10 月 27 日
引擎版本 1.2.1.0.R7	截至 2023 年 10 月 6 日，引擎版本 1.2.1.0.R7 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R7。	2023 年 10 月 6 日
引擎版本 1.2.0.0.R4	截至 2023 年 9 月 29 日，引擎版本 1.2.0.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.0.R4。	2023 年 9 月 29 日
引擎版本 1.2.0.1.R3	截至 2023 年 9 月 27 日，引擎版本 1.2.0.1.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.1.R3。	2023 年 9 月 27 日
引擎版本 1.2.1.0.R6	截至 2023 年 9 月 12 日，引擎版本 1.2.1.0.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R6。	2023 年 9 月 12 日

引擎版本 1.2.0.2.R6	截至 2023 年 9 月 12 日，引擎版本 1.2.0.2.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2.R6。	2023 年 9 月 12 日
关于使用蓝绿部署策略来升级 Neptune 引擎的博客文章	请参阅 Ankit Gupta 和 Abhishek Mishra 撰写的使用蓝绿部署在引擎升级期间提高 Amazon Neptune 的可用性。	2023 年 9 月 11 日
引擎版本 1.2.1.0.R5	截至 2023 年 9 月 2 日，引擎版本 1.2.1.0.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R5。	2023 年 9 月 2 日
引擎版本 1.2.0.2.R5	截至 2023 年 8 月 16 日，引擎版本 1.2.0.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2.R5。	2023 年 8 月 16 日
引擎版本 1.2.1.0.R4	截至 2023 年 8 月 10 日，引擎版本 1.2.1.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R4。	2023 年 8 月 10 日

关于 Neptune 引擎版本 1.2.1.0 的博客文章	请参阅 Joy Wang、Kevin Phillips、Andrea Nassisi 和 Navtanay Sinha 撰写的探索 Amazon Neptune 的特征打包 1.2.1.0 版本。	2023 年 8 月 4 日
关于使用 Neptune 构建多模式数据库解决方案的博客文章	请参阅 Mike Havey 撰写的使用 Amazon Neptune 设计用例驱动、可扩展性高的多模式数据库解决方案。	2023 年 7 月 18 日
有关 Neptune 无服务器用例和最佳实践的博客文章	请参阅 Kevin Phillips 和 Ankit Gupta 撰写的使用 Amazon Neptune 无服务器优化成本和性能的用例和最佳实践。	2023 年 6 月 28 日
引擎版本 1.2.1.0.R3	截至 2023 年 6 月 13 日，引擎版本 1.2.1.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R3。	2023 年 6 月 13 日
关于实时生成休闲建议的博客文章	请参阅 Michael Meidlinger 和 Nils Müller 撰写的使用 Amazon Neptune 实时生成休闲活动的建议。	2023 年 6 月 6 日
关于使用 Neptune 和 RDKit 进行分子建模的博客文章	请参阅 Graham Kutchek 撰写的利用 Amazon Neptune 和 RDKit 为分子 SMILES 数据建模。	2023 年 6 月 1 日

关于使用缓存加速 Neptune 性能（第 3 部分）的博客文章	参见 Taylor Riggan、Abhishek Mishra、Melissa Kwok 和 Kelvin Lawrence 的《在亚马逊海王星中使用缓存提高图形查询性能》，第 3 部分：使用 ElastiCache亚马逊的 Neptune 集群范围的缓存架构。	2023 年 5 月 26 日
关于使用缓存加速 Neptune 性能（第 2 部分）的博客文章	请参阅 Taylor Riggan、Abhishek Mishra、Melissa Kwok 和 Kelvin Lawrence 撰写的在 Amazon Neptune 中使用缓存加速图形查询性能，第 2 部分：其它 Neptune 缓存特征。	2023 年 5 月 26 日
关于使用缓存加速 Neptune 性能（第 1 部分）的博客文章	请参阅 Taylor Riggan、Abhishek Mishra、Melissa Kwok 和 Kelvin Lawrence 撰写的在 Amazon Neptune 中使用缓存加速图形查询性能，第 1 部分：查询和缓冲池缓存。	2023 年 5 月 26 日
关于使用 Neptune 进行供应链分析的博客文章	请参阅 Dhiraj Thakur 和 Rajdip Chaudhur 撰写的使用 Amazon Neptune 和 Neptune Workbench 进行供应链数据分析和可视化。	2023 年 5 月 10 日
引擎版本 1.2.0.2.R4	截至 2023 年 5 月 8 日，引擎版本 1.2.0.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2.R4。	2023 年 5 月 8 日

Neptune 在中东（阿联酋）区域推出	Amazon Neptune 现已在中东（阿联酋）(me-central-1) 区域推出。	2023 年 5 月 2 日
引擎版本 1.2.1.0.R2	截至 2023 年 5 月 2 日，引擎版本 1.2.1.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0.R2。	2023 年 5 月 2 日
关于使用 Media2Cloud 通过人工智能驱动的视频分析在 Neptune 上构建知识图谱的博客文章	请参阅 Mike Havey 撰写的使用 Media2Cloud 通过人工智能驱动的视频分析在 Amazon Neptune 上构建知识图谱。	2023 年 5 月 2 日
关于如何使用 Neptune DevOcean 构建漏洞修复平台的博客文章	参见 G il Makmel 和 Charles Ivie 的《如何使用 Amazon Neptune 为云原生应用程序 DevOcean 构建漏洞修复管理平台》。	2023 年 4 月 25 日
关于 Getir 如何使用 Neptune 构建欺诈检测系统的博客文章	请参阅 Berkay Berkman、M ahmut Turan、Mutlu Polatcan、Umut Cemal Kıraç、Yağız Yanıkoğlu 和 Esra Kayabali 撰写的 Getir 如何使用 Amazon Neptune 和 Amazon DynamoDB 构建全面的欺诈检测系统。	2023 年 4 月 6 日
关于 Wiz 如何使用 Neptune 重新构想云安全的博客文章	请参阅 Ami Luttwak 和 Brad Bebee 撰写的世界是一张图：Wiz 如何使用 Amazon Neptune 中的图形重新构想云安全。	2023 年 3 月 31 日

引擎版本 1.2.0.2.R3	截至 2023 年 3 月 27 日，引擎版本 1.2.0.2.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2.R3。	2023 年 3 月 27 日
关于 CSC Generation 如何使用 Neptune 推动产品发现的博客文章	请参阅 Bobber Cheng、Ronit Rudra 和 Melissa Kwok 撰写的 CSC Generation 如何使用 Amazon Neptune 通过知识图谱推动产品发现。	2023 年 3 月 21 日
引擎版本 1.2.1.0	截至 2023 年 3 月 8 日，引擎版本 1.2.1.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.1.0。	2023 年 3 月 8 日
关于在 Neptune 中探索 TinkerPop 3.6.x 新功能的博客文章	参见 Stephen Mallet te 在 Amazon Neptune 中探索 Apache TinkerPop 3.6.x 的新功能。	2023 年 3 月 8 日
关于使用语义推理从 RDF 图形中推断新事实的博客文章	请参阅 Charles Ivie 和 Diana Marks 撰写的通过将 RDFox 与 Amazon Neptune 集成，使用语义推理从 RDF 图形中推断新的事实。	2023 年 2 月 20 日
关于使用 Neptune 分析医疗保健 FHIR 数据的博客文章	请参阅 Alena Schmickl 撰写的使用 Amazon Neptune 分析医疗保健 FHIR 数据。	2023 年 2 月 13 日

关于使用 Neptune ML 构建实时欺诈检测解决方案的博客文章	请参阅 Hua Shu 和 Soji Adeshina 撰写的使用 Amazon Neptune ML 构建实时欺诈检测解决方案。	2023 年 2 月 8 日
引擎版本 1.1.1.0.R7	截至 2023 年 1 月 23 日，引擎版本 1.1.1.0.R7 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R7。	2023 年 1 月 23 日
图形浏览器已发布	图形浏览器是一款用于可视化图形数据的开源前端 Web 应用程序工具。请参阅 https://github.com/aws/graph-explorer。	2023 年 1 月 3 日
维护发行版 1.1.0.0.R3	截至 2022 年 12 月 23 日，引擎版本 1.1.0.0.R3 维护发行版正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.0.0.R3。	2022 年 12 月 23 日
Neptune 工作台现在可以在亚马逊 Linux 2 和 3 上运行。JupyterLab	Neptune 图形笔记本现在可以在带 JupyterLab 有 3 的亚马逊 Linux 2 环境中运行。有关如何迁移到这个新环境的信息，请参阅将 Neptune 笔记本从 Jupyter 迁移到 JupyterLab 3。	2022 年 12 月 21 日

关于使用 IMDb 知识图谱进行功率推荐和搜索（第 3 部分）的博客文章	请参阅 Divya Bhargavi、Soji Adeshina、Gaurav Rele、Karan Sindwani、Vidya Sagar Ravipati 和 Matthew Rhode 撰写的使用 IMDb 知识图谱进行功率推荐和搜索 - 第 3 部分。	2022 年 12 月 20 日
关于使用 IMDb 知识图谱进行功率推荐和搜索（第 2 部分）的博客文章	请参阅 Matthew Rhodes、Soji Adeshina、Divya Bhargavi、Gaurav Rele、Karan Sindwani 和 Vidya Sagar Ravipati 撰写的使用 IMDb 知识图谱进行功率推荐和搜索 - 第 2 部分。	2022 年 12 月 20 日
关于使用 IMDb 知识图谱进行功率推荐和搜索（第 1 部分）的博客文章	请参阅 Gaurav Rele、Soji Adeshina、Divya Bhargavi、Karan Sindwani、Vidya Sagar Ravipati 和 Matthew Rhodes 撰写的使用 IMDb 知识图谱进行功率推荐和搜索 - 第 1 部分。	2022 年 12 月 20 日
Neptune Serverless 现已在新地区推出 AWS	截至 2022 年 12 月 16 日，Neptune 无服务器已在以下新的 AWS 区域推出：加拿大（中部）、欧洲地区（斯德哥尔摩）、欧洲地区（法兰克福）、亚太地区（新加坡）、亚太地区（悉尼）。有关所有推出 Neptune 无服务器的区域，请参阅 Amazon Neptune 无服务器限制。	2022 年 12 月 16 日

引擎版本 1.2.0.2.R2	截至 2022 年 12 月 15 日，引擎版本 1.2.0.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2.R2。	2022 年 12 月 15 日
引擎版本 1.2.0.0.R3	截至 2022 年 12 月 15 日，引擎版本 1.2.0.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.0.R3。	2022 年 12 月 15 日
引擎版本 1.2.0.1.R2	截至 2022 年 12 月 13 日，引擎版本 1.2.0.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.1.R2。	2022 年 12 月 13 日
关于设计教育大数据分析架构的博客文章 AWS	请参阅 Lavanya Sood 撰写的使用 AWS设计教育大数据分析架构。	2022 年 12 月 13 日
关于图形数据库如何增强学习能力的博客文章	请参阅 Lavanya Sood 撰写的图形数据库如何增强学习能力。	2022 年 12 月 8 日
关于使用 Glue 将 RDF 数据加载到 Nep AWS tune 的博客文章	参见 Mike Havey 和 Fabrizio Napolitano 的《使用 AWS Glue 将 RDF 数据加载到亚马逊 Neptun e》中。	2022 年 11 月 23 日

引擎版本 1.2.0.2	截至 2022 年 11 月 20 日，引擎版本 1.2.0.2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.2。	2022 年 11 月 20 日
引擎版本 1.2.0.1	截至 2022 年 10 月 26 日，引擎版本 1.2.0.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.1。	2022 年 10 月 26 日
关于使用 Neptune 进行欺诈检测的博客文章	请参阅 Wilson Tang、Amr Elnaggar、Matias Pons、Mohammad Azzam、Saurabh Deshpande 和 Luis Rodrigues Soares 撰写的借助 Amazon Neptune 在 Delivery Hero 中实现欺诈检测。	2022 年 10 月 26 日
关于 Neptune 无服务器的博客文章	请参阅 Danilo Poccia 撰写的介绍 Amazon Neptune 无服务器 – 一个完全托管式图形数据库，可根据您的工作负载调整容量。	2022 年 10 月 26 日
关于使用 Lambda 和 SPARQL UPDATE LOAD 将事件驱动的 RDF 导入到 Neptune 的博客文章	了解恩智浦如何使用约翰·沃克、Onno Buijs、Charles Ivie 和Javy de Koning的L AWS ambda和SPARQL更新加载向亚马逊Neptune执行事件驱动的RDF 导入。	2022 年 10 月 20 日

引擎版本 1.2.0.0.R2	截至 2022 年 10 月 14 日，引擎版本 1.2.0.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.0.R2。	2022 年 10 月 14 日
关于在 Neptune 中对多语言文本属性进行编码的博客文章	请参阅 Jiani Zhang 撰写的在 Amazon Neptune 中对多语言文本属性进行编码以训练预测模型。	2022 年 10 月 14 日
关于自动测试 Neptune 数据访问的博客文章	参见 Greg Biegel 的 Greg Biegel 使用 Apache TinkerPop Gremlin 自动测试亚马逊 Neptune 数据访问权限。	2022 年 9 月 28 日
引擎版本 1.1.1.0.R6	截至 2022 年 9 月 23 日，引擎版本 1.1.1.0.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R6。	2022 年 9 月 23 日
关于 Informatica® 如何使用 Neptune 的博客文章	请参阅 Tiju Titus John、Deepak Ram 和 Farooq Ashraf 撰写的 Informatica® 云数据治理和目录如何使用 Amazon Neptune 绘制知识图谱。	2022 年 9 月 20 日
关于使用 Neptune 和 Tom Sawyer Perspectives 揭露财务欺诈的博客文章	请参阅 Tom Sawyer 软件公司高级产品经理 Janet M.Six 撰写的使用 Amazon Neptune 和 Tom Sawyer Perspectives 揭露财务欺诈。	2022 年 8 月 30 日

关于使用 Ne SageMaker ptune 和 DGL 构建基于 GNN 的实时欺诈检测解决方案的博客文章	参见使用亚马逊 SageMaker 、Amazon Neptune 构建基于 GNN 的实时欺诈检测解决方案，以及张健、王浩珠和朱梦欣的 Deep Graph Libr ary。	2022 年 8 月 11 日
关于使用资源标签停止和启动 Neptune 环境资源的博客文章	请参阅 Kevin Phillips 撰写的使用资源标签自动停止和启动 Amazon Neptune 环境资源。	2022 年 8 月 1 日
关于一位为 Apache 做出贡献的艺术家的博客文章 TinkerPop	参见 Stephen Mallette 和 Ketrina T hompson TinkerPop 的《超越代码：为 Apache 做出贡献的艺术家	2022 年 8 月 1 日
关于 Neptune 数据面板操作的精细访问控制的博客文章	请参阅 Abhishek Mishra 和 Ankit Gupta 撰写的 Amazon Neptune 数据面板操作的精细访问控制。	2022 年 7 月 29 日
关于 Neptune 全球数据库的博客文章	请参阅 Navtanay Sinha 撰写的介绍 Amazon Neptune 全球数据库。	2022 年 7 月 27 日
引擎版本 1.2.0.0	截至 2022 年 7 月 21 日，引擎版本 1.2.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.2.0.0。	2022 年 7 月 21 日
引擎版本 1.1.1.0.R5	截至 2022 年 7 月 21 日，引擎版本 1.1.1.0.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R5。	2022 年 7 月 21 日

引擎版本 1.1.1.0.R4	截至 2022 年 6 月 23 日，引擎版本 1.1.1.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R4。	2022 年 6 月 23 日
通过 Python 集成简化图形分析和机器学习工作流程	现在，您可以使用可简化数据科学和机器学习工作流程的开源 Python 集成，对存储在 Amazon Neptune 中的图形数据运行图形分析和机器学习任务。请参阅 Neptune 的AWS Data Wrangler 文档。	2022 年 6 月 7 日
引擎版本 1.1.1.0.R3	截至 2022 年 6 月 7 日，引擎版本 1.1.1.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R3。	2022 年 6 月 7 日
关于检测虚假新闻的博客文章	请参阅 Hasan Shojaei 和 Sarita Joshi 撰写的使用图形机器学习和 Amazon Neptune ML 检测社交媒体虚假新闻。	2022 年 5 月 19 日
关于在 Neptune 中使用 SQL Server Integration Services (SSIS) 的博客文章	请参阅 Mesgana Gormley 和 Melissa Kwok 撰写的使用 SQL Server Integration Services (SSIS) 和 Amazon Neptune 从您的数据中发现新的见解。	2022 年 5 月 18 日

维护发行版 1.1.1.0.R2	截至 2022 年 5 月 16 日，引擎版本 1.1.1.0.R2 维护发行版正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0.R2。	2022 年 5 月 16 日
维护发行版 1.1.0.0.R2	截至 2022 年 5 月 16 日，引擎版本 1.1.0.0.R2 维护发行版正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.0.0.R2。	2022 年 5 月 16 日
维护发行版 1.0.5.1.R4	截至 2022 年 5 月 16 日，引擎版本 1.0.5.1.R4 维护发行版正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.1.R4。	2022 年 5 月 16 日
维护发行版 1.0.5.0.R5	截至 2022 年 5 月 16 日，引擎版本 1.0.5.0.R5 维护发行版正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.0.R5。	2022 年 5 月 16 日
关于 openCypher 在 Neptune 中正式发布的博客文章	请参阅 Navtanay Sinha 和 Dave Bechberger 撰写的宣布 openCypher 对 Amazon Neptune 的支持正式发布。	2022 年 4 月 22 日

引擎版本 1.1.1.0	截至 2022 年 4 月 19 日，引擎版本 1.1.1.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.1.0。	2022 年 4 月 19 日
关于数据谱系的博客文章	参见 Kho a Nguyen、Kr ithivasan Balasubr AWS amaniyan 和 Rahul Shaurya 的《使用 Glue、Amazon Neptune 和 Spline 为数据湖构建数据谱系》。	2022 年 4 月 1 日
重新启用到引擎版本 1.1.0.0 的升级	截至 2022 年 2 月 21 日，暂时禁用升级到引擎版本 1.1.0.0。它们现在已重新启用。	2022 年 3 月 22 日
关于工程公用事业可靠性的博客文章	请参阅 Abhineet Parchure 撰写的使用基于云的、以数据为依据的电力系统模型来设计公用事业可靠性。	2022 年 3 月 22 日
Neptune 在非洲（开普敦）推出	Amazon Neptune 现已在非洲（开普敦）(`af-south-1`) 推出。但是，该区域的 Neptune 控制台暂时禁用了 Neptune Workbench 笔记本支持。	2022 年 2 月 24 日
关于使用 OWL 的模型驱动图形的博客文章	请参阅 Mike Havey 撰写的 Amazon Neptune 中使用 OWL 的模型驱动图形。	2022 年 2 月 23 日
关于使用 Rhizomer 探索语义知识图谱的博客文章	请参阅 Roberto García 撰写的使用 Amazon Neptune 和 Rhizomer 在不使用 SPARQL 的情况下探索语义知识图谱。	2022 年 2 月 22 日

关于绘制公用电网图形的博客文章	参见 Bobby Wilson 和 Jos e AWS ph Beer 的《绘制公用电网图》。	2022 年 2 月 18 日
新的 Neptune ML 文本特征编码选项	Neptune 现在支持用于 FastText 训练的句子 BERT 文本编码。查看 Neptune ML 中的FastText功能和 Neptune ML 中的句子 BERT 功能。	2022 年 2 月 15 日
关于在 Neptune 中使用地理空间查询 OpenSearch 的博客文章	有关地理空间查询，请参阅 Ross Gabay 和 Abhilash Vinod 撰写的《合并亚马逊 Neptune 和亚马逊 OpenSearch 服务》。	2022 年 2 月 1 日
关于使用 Amazon EKS 和 Neptune 发现金融犯罪的博客文章	请参阅 Severin Gassauer-Fleissner 和 Zahi Ben Shabat 撰写的使用 Amazon EKS 和图形数据库发现金融犯罪。	2022 年 2 月 1 日
Neptune 集群卷现在可以增长到 128TiB 大小	在除中国以外的所有受支持区域 GovCloud，Neptune 集群体积的大小限制现已从 64 TiB 增加到 128 Tib。这适用于从版本 1.0.2.2 开始的所有引擎版本。请参阅 Amazon Neptune 存储页面。	2022 年 2 月 1 日
Neptune全文搜索现在与所有版本的集成. OpenSearch	查看使用亚马逊服务在 Amazon Neptune 中进行全文搜索。 OpenSearch	2022 年 1 月 28 日
关于使用 Docker 容器部署图形笔记本的博客文章	请参阅 Ganesh Sawhney 和 Qiang Zhang 的《 AWS使用 Docker 容器部署 Grap h Notebook》	2022 年 1 月 22 日

引擎版本 1.0.5.1.R3	截至 2022 年 1 月 13 日，引擎版本 1.0.5.1.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.1.R3。	2022 年 1 月 13 日
关于使用 Neptune ML 的基于图形的推荐系统的博客文章	请参阅 Yanwei Cui 和 Will Badr 撰写的 Neptune ML 的基于图形的推荐系统：社交网络链接预测挑战的例证。	2022 年 1 月 12 日
关于自动扩缩 Neptune 的博客文章	请参阅 Navtanay Sinha 和 Sudhanshu Gupta 撰写的自动扩缩您的 Amazon Neptune 数据库以满足工作负载需求。	2021 年 11 月 29 日
关于交互式图形数据分析和可视化的博客文章	参见 Sandeep Veldi 和 Abhishek Mishra 的《使用亚马逊 Neptune、A QuickSigh t mazon Athena Federation Query 和 Abhishek Mishra 构建交互式图表数据分析和可视化》。	2021 年 11 月 24 日
关于使用基于图形的深度学习检测身份欺诈的博客文章	请参阅 Kevin O'Brien、K amran Habib 和 Will Badr 撰写的 Careem 如何使用基于图形的深度学习和 Amazon Neptune 检测身份欺诈。	2021 年 11 月 23 日

引擎版本 1.1.0.0	截至 2021 年 11 月 19 日，引擎版本 1.1.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.1.0.0。	2021 年 11 月 19 日
关于集中数据保护和合规性的博客文章	参见 Brian O'Keefe 的《使用 AWS Backup》在 Amazon Neptune 中集中数据保护和合规性。	2021 年 11 月 8 日
关于打击欺诈和不当支付的博客文章	请参阅 Vladi Royzman 和 Spencer Smith 撰写的以联邦支出的规模实时打击欺诈和不当支付。	2021 年 11 月 2 日
关于防止虚假账户注册的博客文章	请参阅 Anjan Biswas 撰写的使用 Amazon Fraud Detector 借助人工智能实时防止虚假账户注册。	2021 年 10 月 29 日
引擎版本 1.0.5.1.R2	截至 2021 年 10 月 26 日，引擎版本 1.0.5.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.1.R2。	2021 年 10 月 26 日
关于 HawkEye 360 使用深度图库预测船舶风险的博客文章	查看 HawkEye 360 使用蒂姆·帕夫利克、伊恩·阿维莱斯、丹·福特和高拉夫·雷尔的 Deep Graph Library 和 Amazon Neptune 预测船舶风险。	2021 年 10 月 15 日

引擎版本 1.0.5.1	截至 2021 年 10 月 1 日，引擎版本 1.0.5.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.1。	2021 年 10 月 1 日
关于开发者为何喜欢的博客文章 TinkerPop	参见 Brad Beb ee TinkerPop 、Kelvin Lawrence 和 Stephen Mallette 的《为什么开发者喜欢图形计算开源框架 Apach e》。	2021 年 9 月 27 日
引擎版本 1.0.5.0.R3	截至 2021 年 9 月 15 日，引擎版本 1.0.5.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.0.R3。	2021 年 9 月 15 日
关于使用 Amundsen 和 Neptune 构建数据发现解决方案的博客文章	请参阅 Peter Hanssens 和 Don Simpson 撰写的使用 Amundsen 和 Amazon Neptune 构建数据发现解决方案。	2021 年 9 月 8 日
Neptune 更新了流轮询器以支持非字符串全文搜索查询	此版本中包括对全文搜索的许多改进，包括支持对非字符串的属性值编制索引。参见 Amazon Neptune 中的非字符串 OpenSearch 索引。	2021 年 8 月 23 日

引擎版本 1.0.5.0.R2	截至 2021 年 8 月 16 日，引擎版本 1.0.5.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.0.R2。	2021 年 8 月 16 日
引擎版本 1.0.4.2.R5	截至 2021 年 8 月 16 日，引擎版本 1.0.4.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.2.R5。	2021 年 8 月 16 日
关于 Neptune 中支持图形存储协议的博客文章	请参阅 Chris Smith 撰写的介绍 Amazon Neptune 的图形存储协议支持。	2021 年 8 月 2 日
关于 Neptune ML 中新特征的博客文章	请参阅 Soji Adeshina 撰写的使用 Amazon Neptune ML 的新特征在您的图表中发现更多见解。	2021 年 7 月 30 日
关于使用 Neptune ML 更快地获得预测的博客文章	请参阅 Soji Adeshina 撰写的使用 Amazon Neptune ML 更快地预测不断演变的图形数据。	2021 年 7 月 30 日
关于使用 Neptune ML 更轻松、更快速地进行图形机器学习的博客文章	请参阅 Soji Adeshina 撰写的使用 Amazon Neptune ML 更轻松、更快速地进行图形机器学习。	2021 年 7 月 30 日

关于 Neptune openCypher 支持的博客文章	请参阅 Brad Bebee 撰写的推出适用于 Amazon Neptune 的 openCypher：与 openCypher 和 Gremlin 一起构建更好的图形应用程序。	2021 年 7 月 29 日
引擎版本 1.0.5.0	截至 2021 年 7 月 27 日，引擎版本 1.0.5.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.5.0。	2021 年 7 月 27 日
引擎版本 1.0.4.2.R4	截至 2021 年 7 月 23 日，引擎版本 1.0.4.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.2.R4。	2021 年 7 月 23 日
Neptune 在中国（北京）推出	Amazon Neptune 现已在中国（北京）(cn-north-1 ）推出。	2021 年 7 月 21 日
引擎版本 1.0.4.2.R3	截至 2021 年 6 月 28 日，引擎版本 1.0.4.2.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.2.R3。	2021 年 6 月 28 日
关于 Dream11 如何使用 Neptune 扩展社交网络的博客文章	了解全球最大的幻想体育平台 Dream11 如何通过亚马逊 Neptune 和亚马逊扩展其社交网络。 ElastiCache	2021 年 6 月 25 日

关于使用 Neptune 和 PoolParty 语义套件将数据转化为知识的博客文章	参见 Ioanna Lytra 和 Albin Ahmeti 的《使用 PoolParty 语义套件和 Amazon Neptune 将数据转化为知识》。	2021 年 6 月 16 日
关于使用 Neptune 浏览知识库的博客文章 UniProt	参见 Eric Greene、Rafa Xu 和 Yuan Shi 的《使用 AWS 开放数据和 Amazon Neptune 探索 UniProt 蛋白质知识库》。	2021 年 6 月 10 日
关于使用 Neptune 进行数据驱动风险分析的博客文章	参见 Adriaan de J onge 和 Rohit Satyanarayana 的《实地笔记：使用 OpenSearch 亚马逊 Neptune 和亚马逊服务进行数据驱动的风险分析》。	2021 年 6 月 10 日
引擎版本 1.0.4.2.R2	截至 2021 年 6 月 1 日，引擎版本 1.0.4.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.2.R2。	2021 年 6 月 1 日
关于使用 Neptune 实现基础设施可视化的博客文章 AWS	参见 Rohan Raizada 和 Amey Dhavle 的《使用 Amazon Neptune and AWS Config 可视化你的 AWS 基础设施》。	2021 年 5 月 25 日
关于在 Neptune 上使用 Data Lens 的配置的博客文章	参见罗素·沃特森的《使用数据镜头在 Amazon Neptune 中配置 AWS 服务以构建知识图表》。	2021 年 5 月 5 日
关于使用 Data Lens 在 Neptune 中构建知识图谱的博客文章	请参阅 Russell Waterson 撰写的使用 Data Lens 在 Amazon Neptune 中构建知识图谱。	2021 年 5 月 5 日

引擎版本 1.0.1.0、1.0.1.1 和 1.0.1.2 现已弃用	从现在开始，将不会使用这些引擎版本中的任何一个或与之相关的任何补丁来创建新的数据库实例。	2021 年 4 月 26 日
关于日本经济产业省使用 Neptune 的案例研究的英文翻译	请参阅日本经济产业省使用 AWS为 gBizINFO Corporate 信息搜索数据库提供支持。	2021 年 3 月 31 日
关于在 Amazon Comprehend 和 Lex 上使用 Neptune 的博客文章	请参阅 Dave Bechberger 撰写的使用 Amazon Neptune、Amazon Comprehend 和 Amazon Lex 增强您的知识图谱。	2021 年 3 月 31 日
关于在 Neptune 中使用 Lambda 函数的博客文章	参见伊恩·罗宾逊的《在亚马逊 Neptune 上使用 AWS Lambda 函数》。	2021 年 3 月 26 日
引擎版本 1.0.4.1.R1.1	截至 2021 年 3 月 22 日，引擎版本 1.0.4.1.R1.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.1.R1.1。	2021 年 3 月 22 日
引擎版本 1.0.4.1.R2.1	截至 2021 年 3 月 11 日，引擎版本 1.0.4.1.R2.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.1.R2.1。	2021 年 3 月 11 日

关于使用 Neptune 的开源图形笔记本进行图形可视化的博客文章	请参阅 Joy Wang、Ora Lassila 和 Stephen Mallette 撰写的开始使用开源图形笔记本进行图形可视化。	2021 年 3 月 10 日
关于将 Neptune 与 Amundsen 数据发现和元数据引擎集成的教程	请参阅 Andrew Ciambrone 撰写的如何在 Amazon Neptune 上使用 Amundsen。	2021 年 3 月 2 日
引擎版本 1.0.4.1.R2	截至 2021 年 2 月 24 日，引擎版本 1.0.4.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.1.R2。	2021 年 2 月 24 日
引擎版本 1.0.4.0.R2	截至 2021 年 2 月 24 日，引擎版本 1.0.4.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.0.R2。	2021 年 2 月 24 日
引擎版本 1.0.3.0.R3	截至 2021 年 2 月 19 日，引擎版本 1.0.3.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.3.0.R3。	2021 年 2 月 19 日

引擎版本 1.0.2.2.R6	截至 2021 年 2 月 19 日，引擎版本 1.0.2.2.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.2.R6。	2021 年 2 月 19 日
关于使用 Amazon Comprehend 事件构建知识图谱的博客文章	请参阅 Brian O'Keefe、Graham Horwood 和 Navtanay Sinha 撰写的使用 Amazon Comprehend Events 在 Amazon Neptune 中构建知识图谱。	2021 年 1 月 19 日
关于启用低代码图形数据应用程序的博客文章	请参阅 Leo Meyerovich、Dave Bechberger 和 Taylor Riggan 撰写的使用 Amazon Neptune 和 Graphistry 启用低代码图形数据应用程序。	2021 年 1 月 18 日
添加了用于开始使用图形数据的笔记本文档。	添加了与 Neptune Workbench 集成的部分，可帮助您开始创建图形数据和开发图形应用程序，而无需在准备就绪之前启动 Neptune 集群。	2021 年 1 月 15 日
关于在几秒钟内重置 Neptune 图形数据的博客文章	请参阅 Niraj Jetly 和 Navtanay Sinha 撰写的在几秒钟内重置 Amazon Neptune 中的图形数据。	2020 年 12 月 17 日

关于诺华股份公司如何使用 Nept SageMaker une 和 BERT 的博客文章	参见 Othmane Hamzaoui SageMaker 、Fatema Alkhanaizi 和 Viktor Malesevic 的 Neptune 使用亚马逊和亚马逊 Neptune 来构建和丰富知识图谱。	2020 年 12 月 14 日
关于使用主题网络构建知识图谱的博客文章	请参阅 Edward Brown（人工智能项目负责人）、Eduardo Piairo（架构师）、Marcia Oliveira（首席数据科学家）和 Jack Hampson（Deeper Insights 首席执行官）撰写的在 Amazon Neptune 中使用主题网络构建知识图谱。	2020 年 12 月 14 日
引擎版本 1.0.4.1	截至 2020 年 12 月 8 日，引擎版本 1.0.4.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.1。	2020 年 12 月 8 日
关于开始使用 Neptune ML 的博客文章	请参阅 George Karypis、Dave Bechberger 和 Karthik Bharathy 撰写的如何开始使用 Neptune ML。	2020 年 12 月 8 日
Neptune 现在有了快速重置 API	使用快速重置 API，您可以快速、轻松地删除数据库集群中的所有数据。请参阅快速重置 API。	2020 年 12 月 4 日
关于在 Pendulum 构建生物学知识图谱的博客文章	请参阅 Connor Skennerton 撰写的使用 Amazon Neptune 在 Pendulum 构建生物学知识图谱。	2020 年 11 月 26 日

关于 Neptune 中 TinkerPop 3.4.8 新功能的博客文章	参见 Stephe n Mallette 在 Amazon Neptune 中探索 Apache TinkerPop 3.4.8 的新功能。	2020 年 11 月 18 日
关于在 Neptune 中使用 Amazon Kendra 搜索服务的博客文章	请参阅 Yazdan Shirvany、Mohit Mehta 和 Dipto Chakravarty 撰写的将您的企业知识图谱纳入 Amazon Kendra。	2020 年 11 月 17 日
事件通知现已推出	Neptune 现在支持事件通知，您可以使用这些通知更轻松地监控数据库集群。请参阅使用 Neptune 事件通知。	2020 年 10 月 29 日
自定义端点现已推出	Neptune 现在支持自定义端点，以便更好地控制与数据库实例的连接。请参阅连接到 Amazon Neptune 端点。	2020 年 10 月 29 日
关于使用 AWS 数据库迁移服务 (DMS) 填充海王星图的博客文章	参见 Chris Smith 的《 AWS 使用数据库迁移服务 (DMS) 从关系数据库在 Amazon Neptune 中填充图表 — 第 4 部分：汇总所有内容》。	2020 年 10 月 22 日
关于使用 AWS 数据库迁移服务 (DMS) 填充海王星图的博客文章	参见 Chris Smith 的《 AWS 使用数据库迁移服务 (DMS) 从关系数据库在 Amazon Neptune 中填充图表 — 第 3 部分：设计 RDF 模型》。	2020 年 10 月 22 日
关于使用 AWS 数据库迁移服务 (DMS) 填充海王星图的博客文章	参见 Chris Smith 的《 AWS 使用数据库迁移服务 (DMS) 从关系数据库在 Amazon Neptune 中填充图表 — 第 2 部分：设计属性图模型》。	2020 年 10 月 22 日

关于使用 AWS 数据库迁移服务 (DMS) 填充海王星图的博客文章	参见 Chris Smith 的《AWS 使用数据库迁移服务 (DMS) 从关系数据库在 Amazon Neptune 中填充图表 — 第 1 部分：搭建舞台》。	2020 年 10 月 22 日
引擎版本 1.0.4.0	截至 2020 年 10 月 12 日，引擎版本 1.0.4.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.4.0。	2020 年 10 月 12 日
引擎版本 1.0.3.0.R2	截至 2020 年 10 月 12 日，引擎版本 1.0.3.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.3.0.R2。	2020 年 10 月 12 日
引擎版本 1.0.2.2.R5	截至 2020 年 10 月 12 日，引擎版本 1.0.2.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.2.R5。	2020 年 10 月 12 日
关于为 SPARQL 联合查询配置 VPC 的博客文章	请参阅 Charles Ivie 撰写的使用 Amazon Neptune 为 SPARQL 1.1 联合查询配置 Amazon VPC。	2020 年 10 月 12 日
关于编写 SPARQL 级联删除的博客文章	请参阅 Ora Lassila 撰写的在 SPARQL 中编写级联删除。	2020 年 10 月 5 日

关于使用 Neptune 绘制 AWS 资源图表的博客文章	参见 Dave Bechberger 的《使用亚马逊 Neptune 绘制你的 AWS 资源图表》。	2020 年 9 月 28 日
关于使用 Neptune 为药物警戒和不良事件报告构建 MedDRA 术语映射的博客文章	请参阅 Vaijayanti Joshi、Deven Atnoor 博士和 Sudhanshu Malhotra 撰写的为药物警戒和不良事件报告构建基于 Amazon Neptune 的 MedDRA 术语映射。	2020 年 9 月 24 日
关于使用 Neptune 从数据仓库中构建知识图谱以补充商业智能的博客文章	请参阅 Shahria Hossain 和 Mikael Graindorge 撰写的使用 Amazon Neptune 从数据仓库中构建知识图谱以补充商业智能。	2020 年 9 月 23 日
关于使用 Neptune Gremlin 客户端进行负载均衡的博客文章	请参阅 Ian Robinson 撰写的使用 Amazon Neptune Gremlin 客户端进行负载均衡图形查询。	2020 年 9 月 16 日
关于在 Cox Automotive 使用身份图谱进行数字个性化的博客文章	请参阅 Carlos Rendon 和 Niraj Jetly 撰写的 Cox Automotive 使用由 Amazon Neptune 提供支持的身份图谱来扩展数字个性化。	2020 年 9 月 16 日
关于协同筛选 Yelp 数据的博客文章	请参阅 Chad Tindel 撰写的在 Amazon Neptune 中使用 Yelp 数据协同筛选构建推荐系统。	2020 年 9 月 8 日
关于 Amazon Neptune 中查询结果可视化的博客文章	请参阅 Kelvin Lawrence 撰写的使用 Amazon Neptune Workbench 可视化查询结果。	2020 年 9 月 2 日

Neptune 发布了图形可视化	Amazon Neptune 现在在 Neptune Workbench 的 Jupyter 笔记本中提供了广泛的图形可视化功能，以及许多使笔记本更易于使用的新特征。请参阅图形可视化。	2020 年 8 月 12 日
Neptune 在南美洲（圣保罗）推出	Amazon Neptune 现已在南美洲（圣保罗）(`sa-east-1`) 推出。	2020 年 8 月 6 日
Neptune 在亚太地区（香港）推出	Amazon Neptune 现已在亚太地区（香港）(`ap-east-1`) 推出。	2020 年 8 月 6 日
引擎版本 1.0.3.0	截至 2020 年 8 月 3 日，引擎版本 1.0.3.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.3.0。	2020 年 8 月 3 日
引擎版本 1.0.2.2.R4	截至 2020 年 7 月 23 日，引擎版本 1.0.2.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.2.R4。	2020 年 7 月 23 日
关于 Zerobase 使用 Amazon Neptune 自动追踪联系人的博客文章	请参阅 David Harris 和 Aron Szanto 撰写的 Zerobase 使用 Amazon Neptune 创建私密、安全和自动的联系人追踪。	2020 年 7 月 13 日
Neptune 在美国西部（北加利福尼亚）推出	Amazon Neptune 现已在美国西部（北加利福尼亚）(`us-west-1`) 推出。	2020 年 7 月 9 日

Amazon Neptune 支持基于标签的访问控制	现在，您可以在 IAM 策略中使用 AWS 标签来控制对您的 Neptune 数据库的访问。请参阅 Amazon Neptune 中基于标签的访问控制。	2020 年 7 月 7 日
Java 流轮询器现已推出	Amazon Neptune 现在对于 Neptune 流以及 Python 流支持 Java 版本的 lambda 流轮询器。请参阅添加有关您正在创建的 Neptune 流使用者堆栈的详细信息。	2020 年 7 月 6 日
关于 AWS COVID-19 知识图谱的博客文章	参见 Ninad Kulkarni、Colby Wise、George Price 和 Miguel Romero 的《构建和查询 AWS COVID-19 知识图谱》。	2020 年 7 月 1 日
引擎版本 1.0.1.1	截至 2020 年 6 月 26 日，引擎版本 1.0.1.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.1.1。	2020 年 6 月 26 日
关于从 Blazegraph 迁移到 Amazon Neptune 的博客文章	请参阅 Dave Bechberger 撰写的博客文章迁移到云：将 Blazegraph 迁移到 Amazon Neptune。	2020 年 6 月 25 日
关于将数据捕获从 Neo4j 更改为 Amazon Neptune 的博客文章	请参阅 Sanjeet Sahay 撰写的博客文章使用 Amazon Managed Streaming for Apache Kafka 将数据捕获从 Neo4j 更改为 Amazon Neptune。	2020 年 6 月 22 日

关于 Waves 如何使用 Amazon Neptune 的博客文章	请参阅 Pavel Vasilyev 撰写的博客文章 Waves 如何使用 Amazon Neptune 大规模运行用户查询和建议。	2020 年 6 月 16 日
引擎版本 1.0.1.2	截至 2020 年 6 月 10 日，引擎版本 1.0.1.2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.1.2。	2020 年 6 月 10 日
关于构建客户知识存储库的博客文章	请参阅 Ram Bhandarkar 撰写的博客文章使用 Amazon Neptune 和 Amazon Redshift 构建全面的客户知识存储库。	2020 年 6 月 9 日
关于 Gunosy 如何使用 Amazon Neptune 的博客文章	请参阅 Yosuke Uchiyama 撰写的博客文章 Gunosy 如何使用 Amazon Neptune 在 News Pass 中构建评论功能。	2020 年 6 月 8 日
关于 AWS COVID-19 知识图谱的博客文章	参见 Ninad Kulkarni、Colby Wise、George Price 和 Miguel Romero 的 AWS COVID-19 知识图谱构建和查询。	2020 年 6 月 2 日
关于使用 Amazon Neptune 探索 COVID-19 研究的博客文章	请参阅 George Price、Colby Wise、Miguel Romero 和 Ninad Kulkarni 撰写的使用 Amazon Neptune、Amazon Comprehend Medical 和 Tom Sawyer 图数据库浏览器探索 COVID-19 的科学研究。	2020 年 6 月 2 日
现在，你可以使用以下命令将数据加载到 Neptune AWS DMS	请参阅使用 AWS 数据库迁移服务将数据从其他数据存储加载到 Amazon Neptune。	2020 年 6 月 1 日

引擎版本 1.0.2.0 被弃用	Amazon Neptune 引擎版本 1.0.2.0 现已弃用。在此引擎版本上运行的集群将在 2020 年 6 月 1 日之后的第一个维护时段内自动升级到版本 1.0.2.1。	2020 年 5 月 19 日
关于使用 Neptune 构建客户身份图谱的博客文章	请参阅 Rajesh Wunnava 和 Taylor Riggan 编纂的使用 Amazon Neptune 构建客户身份图谱。	2020 年 5 月 12 日
引擎版本 1.0.2.0.R3	截至 2020 年 5 月 5 日，引擎版本 1.0.2.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.0.R3。	2020 年 5 月 5 日
引擎版本 1.0.2.1.R6	截至 2020 年 4 月 22 日，引擎版本 1.0.2.1.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.1.R6。	2020 年 4 月 22 日
关于将数据从 Neo4j 迁移到 Neptune 的博客文章	请参阅通过全自动实用工具将 Neo4j 图形数据库迁移到 Amazon Neptune，作者 Sanjeet Sahay。	2020 年 4 月 13 日
关于使用 Neptune 降低构建图形应用程序成本的博客文章	请参阅使用 Amazon Neptune T3 实例将构建图形应用程序的成本减少达 76%，作者 Karthik Bharathy 和 Brad Bebee。	2020 年 4 月 9 日

Neptune 提供了 T3 可突增实例类	您现在可以创建 Amazon Neptune T3 可突增实例以实现经济高效的开发和测试用途。请参阅 Neptune T3 可突增实例类。	2020 年 4 月 8 日
引擎版本 1.0.2.2.R2	截至 2020 年 4 月 2 日，引擎版本 1.0.2.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.2.R2。	2020 年 4 月 2 日
关于在 EDGAR 描绘投资依赖关系的博客文章	请参阅描绘与 Amazon Neptune 的投资依赖关系，作者 Lawrence Verdi。	2020 年 3 月 17 日
Neptune 在欧洲地区（巴黎）推出	Amazon Neptune 现已在欧洲地区（巴黎）(eu-west-3) 推出。	2020 年 3 月 11 日
引擎版本 1.0.2.2	截至 2020 年 3 月 9 日，引擎版本 1.0.2.2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。有关此引擎版本的更多信息，请参阅 Neptune 引擎版本 1.0.2.2。	2020 年 3 月 9 日

停止和重新启动数据库集群	现在，您可以使用 Neptune 控制台将数据库集群停止 7 天，然后在您再次需要时重新启动它。在停止数据库集群后，您只需支付集群存储、手动快照和自动备份存储的费用，而无需支付任何数据库实例小时的费用。请参阅停止和启动 Amazon Neptune 数据库集群。	2020 年 2 月 19 日
有关耐克社交图景的视频	收听托德·埃斯卡洛纳与耐克高级工程经理马克·旺根海姆的 AWS 会谈，内容涉及该公司如何通过基于Amazon Neptune构建的社交图表为许多应用程序提供支持。请参阅耐克：借助 Amazon Neptune 实现的大规模社交图景。	2020 年 2 月 11 日
现在可将 Neptune 集群配置为需要 SSL 连接	在仍支持 HTTP 连接的区域中，默认情况下，现已在所有新参数组中启用 SSL。虽然未对现有参数组进行更改，但可通过将 `neptune_enforce_ssl` 参数更改为 1 来强制客户端使用 SSL。有关如何为仍支持 HTTP 连接的区域中的集群启用这些连接的信息，请参阅传输中加密：使用 SSL/HTTPS 连接到 Neptune。有关集群和实例参数的介绍，请参阅可用于配置 Amazon Neptune 的参数。	2020 年 2 月 10 日

现在，你可以在 Neptune 的模板中指定引擎版本和删除保护 CloudFormation	Amazon Neptune 已更新其 CloudFormation 模板，添加了一个`AWS::Neptune::DBCluster.EngineVersion` 允许您为新数据库集群指定特定引擎版本的`AWS::Neptune::DBCluster.DeletionProtection` 参数和一个允许您为其开启删除保护的参数。	2020 年 2 月 9 日
删除保护	Amazon Neptune 为数据库集群和实例提供了删除保护。一旦在数据库集群或实例上启用删除保护，就无法将其删除。请参阅如果已启用删除保护，则无法删除数据库实例。	2020 年 1 月 20 日
Neptune 在中国（宁夏）推出	Amazon Neptune 现已在中国（宁夏）(cn-northwest-1) 推出。	2020 年 1 月 15 日
引擎版本 1.0.2.1.R4	引擎版本 1.0.2.1 的补丁 R4 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.1.R4。	2019 年 12 月 20 日
引擎版本 1.0.2.1.R3	引擎版本 1.0.2.1 的补丁 R3 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.1.R3。	2019 年 12 月 12 日
关于使用 Neptune 分析社交媒体源的博客文章	请参阅使用 Amazon Neptune 分析社交媒体源。	2019 年 11 月 27 日

引擎版本 1.0.2.1.R2	引擎版本 1.0.2.1 的补丁 R2 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.1.R2。	2019 年 11 月 25 日
引擎版本 1.0.2.1.R1	Amazon Neptune 引擎版本 1.0.2.1.R1 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.1。	2019 年 11 月 22 日
引擎版本 1.0.2.0.R2	引擎版本 1.0.2.0 的补丁 R2 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.0.R2。	2019 年 11 月 21 日
关于 re:Invent 2019 上的 Neptune 会议和研讨会的博客文章	在 re: Invent 2019 上查看你的 Amazon Neptune 会议、研讨会和粉笔讲座 AWS 指南。	2019 年 11 月 20 日
引擎版本 1.0.2.0.R1	Amazon Neptune 引擎版本 1.0.2.0.R1 已正式发布。有关更多信息，请参阅 Neptune 引擎版本 1.0.2.0。	2019 年 11 月 8 日
关于使用 Neptune Streams 捕获图形更改的博客文章	请参阅使用 Neptune Streams 捕获图形更改。	2019 年 11 月 6 日
引擎版本 1.0.1.0.200502.0	Amazon Neptune 引擎版本 1.0.1.0.200502.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200502.0。	2019 年 10 月 31 日
Neptune 在中东（巴林）推出	Amazon Neptune 现已在中东（巴林）(me-south-1) 推出。	2019 年 10 月 30 日
Neptune 在加拿大（中部）推出	Amazon Neptune 现已在加拿大（中部）(ca-central-1) 推出。	2019 年 10 月 30 日

关于 Neptune 的新 SPARQL 流特征和 SPARQL 联合查询支持的博客文章	请参阅 Amazon Neptune 发布了流、SPARQL 联合查询图等。	2019 年 10 月 17 日
引擎版本 1.0.1.0.200463.0	Amazon Neptune 引擎版本 1.0.1.0.200463.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200463.0。	2019 年 10 月 15 日
引擎版本 1.0.1.0.200457.0	Amazon Neptune 引擎版本 1.0.1.0.200457.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200457.0。	2019 年 9 月 19 日
关于 Neptune 的新 SPARQL Explain 特征的博客文章	请参阅使用 SPARQL Explain 了解 Amazon Neptune 中的查询执行。	2019 年 9 月 17 日
关于 Neptune 支持 3.4 的博客文章 TinkerPop	参见 Amazon Neptune 现在支持 TinkerPop 3.4 功能。	2019 年 9 月 6 日
关于在亚马逊上使用 Neptune 的 PyTorch 博客文章 SageMaker	查看在亚马逊 SageMaker 和 PyTorch 亚马逊 Neptune 上使用的个性化 “按风格购物” 体验。	2019 年 8 月 22 日
关于使用 Neptune 和 AWS AppSync Amazon Elasticache 的博客文章	参见将替代数据源与 AWS AppSync：亚马逊 Neptune 和亚马逊集成。 ElastiCache	2019 年 8 月 22 日
Neptune 在 AWS GovCloud（美国东部）发射	亚马逊 Neptune 现已在 AWS GovCloud（美国东部）（us-gov-east-1）上市。	2019 年 8 月 21 日
Neptune 在 AWS GovCloud（美国西部）发射	亚马逊 Neptune 现已在 AWS GovCloud（美国西部）（us-gov-west-1）上市。	2019 年 8 月 14 日

引擎版本 1.0.1.0.200369.0	Amazon Neptune 引擎版本 1.0.1.0.200369.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200369.0。	2019 年 8 月 13 日
引擎版本 1.0.1.0.200366.0	Amazon Neptune 引擎版本 1.0.1.0.200366.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200366.0。	2019 年 7 月 26 日
关于在亚马逊上使用 Neptune 的 PyTorch 博客文章 SageMaker	查看在亚马逊 SageMaker 和 PyTorch 亚马逊 Neptune 上使用的个性化“按风格购物”体验。	2019 年 7 月 3 日
引擎版本 1.0.1.0.200348.0	Amazon Neptune 引擎版本 1.0.1.0.200348.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200348.0。	2019 年 7 月 2 日
Neptune 在欧洲地区（斯德哥尔摩）推出	Amazon Neptune 现已在欧洲地区（斯德哥尔摩）(eu-north-1) 推出。	2019 年 6 月 27 日
Neptune 现在可以将审核日志发布到日志 CloudWatch	有关更多信息，请参阅将 Nep tune 日志发布到亚马逊 CloudWatch 日志。	2019 年 6 月 18 日
引擎版本 1.0.1.0.200310.0	Amazon Neptune 引擎版本 1.0.1.0.200310.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200310.0。	2019 年 6 月 12 日
关于 LifeOmic's 的博客文章 JupiterOne	查看如何使用 Amazon N LifeOmic eptune JupiterOne 简化安全与合规操作。	2019 年 5 月 2 日

Neptune 在亚太地区（首尔）推出	Amazon Neptune 现已在亚太地区（首尔）(ap-northeast-2) 推出。	2019 年 5 月 1 日
引擎版本 1.0.1.0.200296.0	Amazon Neptune 引擎版本 1.0.1.0.200296.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200296.0。	2019 年 5 月 1 日
Neptune 在亚太地区（孟买）推出	Amazon Neptune 现已在亚太地区（孟买）(ap-south-1) 推出。	2019 年 3 月 6 日
有关 Gremlin 查询提示的博客文章	请参阅 Amazon Neptune 的 Gremlin 查询提示简介。	2019 年 2 月 26 日
Neptune 在亚太地区（东京）推出	Amazon Neptune 现已在亚太地区（东京）(ap-northeast-1) 推出。	2019 年 1 月 23 日
AWS CloudFormation 用于创建访问 Nept AWS Lambda une 的函数的模板	更新了入门部分，并添加了一个 AWS CloudFormation 模板来创建用于 Neptune 的 Lambda 函数。有关更多信息，请参阅 Neptune 入门。	2019 年 1 月 23 日
引擎版本 1.0.1.0.200267.0	Amazon Neptune 引擎版本 1.0.1.0.200267.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200267.0。	2019 年 1 月 21 日
Neptune 在亚太地区（悉尼）推出	Amazon Neptune 现已在亚太地区（悉尼）(ap-southeast-2) 推出。	2019 年 1 月 9 日
有关使用 Metaphactory 的博客文章	请参阅利用 Metaphactory 探索 Amazon Neptune 上的知识图谱。	2019 年 1 月 9 日

Neptune 在亚太地区（新加坡）推出	Amazon Neptune 现已在亚太地区（新加坡）(ap-southeast-1) 推出。	2018 年 12 月 13 日
引擎版本 1.0.1.0.200264.0	Amazon Neptune 引擎版本 1.0.1.0.200264.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200264.0。	2018 年 11 月 19 日
Amazon Neptune SSL 支持	Neptune 现在支持 SSL 连接。	2018 年 11 月 19 日
整合了错误主题	所有错误消息和代码信息现在都位于一个主题中。	2018 年 11 月 15 日
更新了入门主题	使用更多链接和经过认可的文档更新了入门主题。	2018 年 11 月 14 日
引擎版本 1.0.1.0.200258.0	Amazon Neptune 引擎版本 1.0.1.0.200258.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200258.0。	2018 年 11 月 8 日
Neptune 在欧洲地区（法兰克福）推出	Amazon Neptune 现已在欧洲地区（法兰克福）(eu-central-1) 推出。	2018 年 11 月 7 日
系列博客文章 1	请参阅我来为您创建该图形 - 第 1 部分 - Air Routes。	2018 年 11 月 7 日
关于使用 Amazon SageMaker Jupyter 笔记本的博客文章	请参阅使用亚马逊 Jupyter SageMaker 笔记本分析亚马逊 Neptune 图表。	2018 年 11 月 1 日
引擎版本 1.0.1.0.200255.0	Amazon Neptune 引擎版本 1.0.1.0.200255.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200255.0。	2018 年 10 月 29 日

Neptune 在欧洲地区（伦敦）推出	Amazon Neptune 现已在欧洲地区（伦敦）(eu-west-2) 推出。	2018 年 10 月 3 日
引擎版本 1.0.1.0.200237.0	Amazon Neptune 引擎版本 1.0.1.0.200237.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200237.0。	2018 年 9 月 6 日
引擎版本 1.0.1.0.200236.0	Amazon Neptune 引擎版本 1.0.1.0.200236.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200236.0。	2018 年 7 月 24 日
引擎版本 1.0.1.0.200233.0	Amazon Neptune 引擎版本 1.0.1.0.200233.0 已正式发布。有关更多信息，请参阅更新 1.0.1.0.200233.0。	2018 年 6 月 22 日
新 Neptune 快速入门	更新了快速入门 AWS CloudFormation 和 Gremlin 控制台教程。有关更多信息，请参阅 Amazon Neptune 快速入门使用。 AWS CloudFormation	2018 年 6 月 19 日
Amazon Neptune 初始版本	这是 Neptune 用户指南的初始版本。另请参阅发布博客文章 Amazon Neptune 已全面推出。	2018 年 5 月 30 日
介绍性 Neptune 博客文章	请参阅 Amazon Neptune – 一项完全托管的图形数据库服务。	2017 年 11 月 29 日

Amazon Neptune 入门

Amazon Neptune 是一项完全托管式图形数据库服务，可扩展到处理数十亿个关系，并允许您以毫秒延迟查询它们，而这种容量的成本却很低。

如果您正在寻找有关 Neptune 的更多详细信息，请参阅Amazon Neptune 特征概述。

如果您已经了解图形，可以直接跳到使用图形笔记本。或者，如果您想立即创建 Neptune 数据库，请参阅使用 AWS CloudFormation 堆栈创建 Neptune 数据库集群。

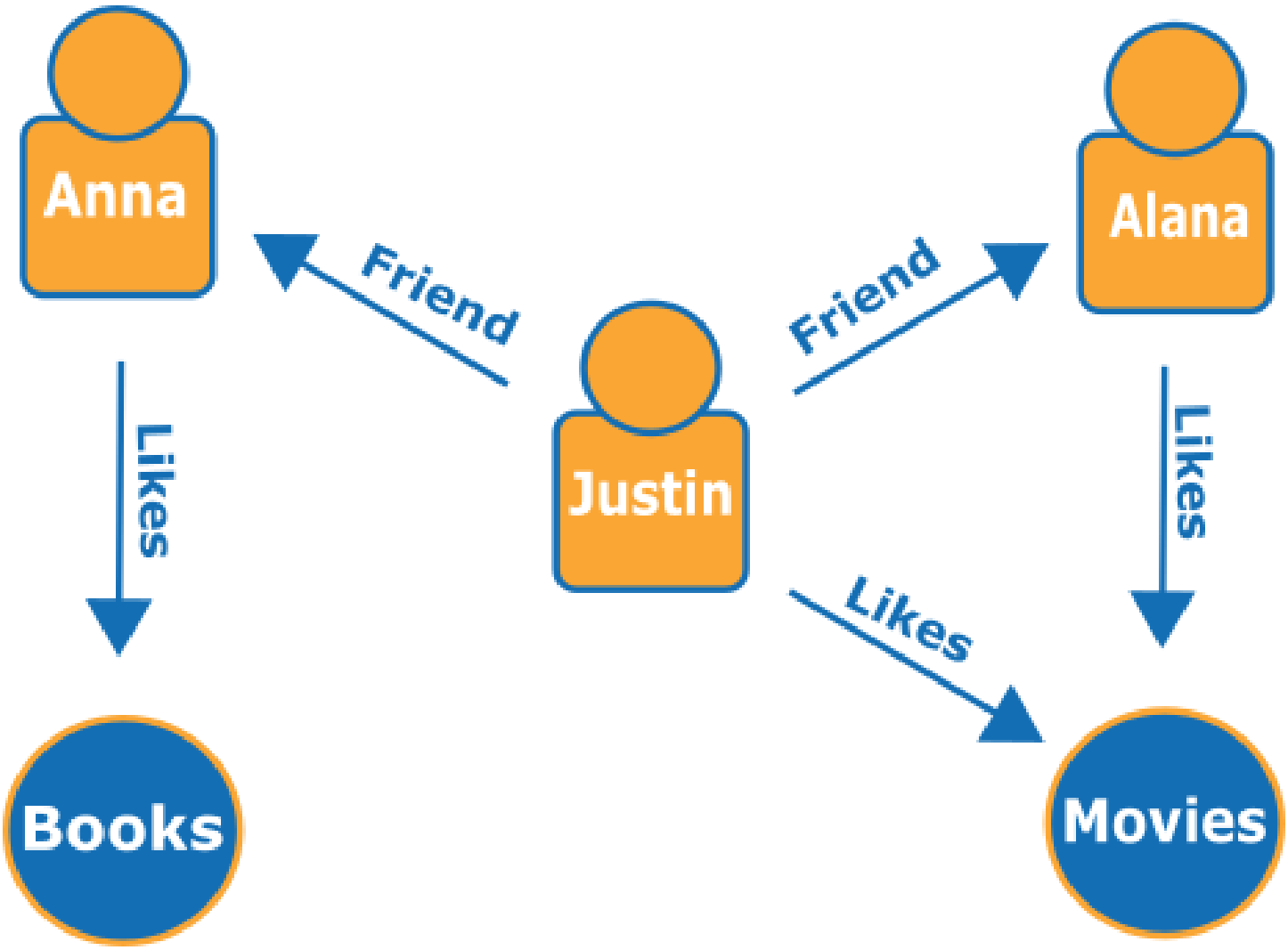
否则，在开始之前，您可能需要了解更多关于图形数据库的信息。

图形数据库到底是 什么？

图形数据库经过优化，可存储和查询数据项之间的关系。

它们将数据项本身存储为图形的顶点，将它们之间的关系存储为边缘。每个边缘都有一种类型，并且从一个顶点（起点）指向另一个顶点（终点）。关系可以称为谓词，也可以称为边缘，顶点有时也被称为节点。在所谓的属性图中，顶点和边缘也可以具有与之关联的其它属性。

这是一张代表社交网络中朋友和爱好的小图形



边缘显示为命名箭头，顶点代表它们连接的特定人物和爱好。

此图形的简单遍历可以告知您 Justin 朋友们的爱好。

为什么要使用图形数据库？

如果实体间的连接或关系是您正在尝试建模的数据的核心，那就适合使用图形数据库。

首先，可以很容易地将数据互连建模为图形，然后编写复杂的查询，以从图形中提取现实世界的信息。

使用关系数据库构建等效应用程序要求您创建许多包含多个外键的表，然后编写嵌套的 SQL 查询和复杂的联接。从编码的角度来看，这种方法不仅很快变得笨拙，而且随着数据量增加，其性能也会迅速下降。

相比之下，像 Neptune 这样的图形数据库可以查询数十亿个顶点之间的关系，而不会陷入困境。

您能用图形数据库做什么？

图形可以通过多种方式表示现实世界中实体的相互关系，包括操作、所有权、父母身份、购买选择、人际关系、家庭关系等。

以下是使用图形数据库的一些最常见领域：

- 知识图谱 – 知识图谱可让您组织和查询各种关联信息，以回答一般问题。使用知识图谱，您可以将主题信息添加到产品目录中，并对维基数据中包含的不同信息进行建模。

  要详细了解知识图谱的工作原理及其用途，请参阅 AWS 上的知识图谱。

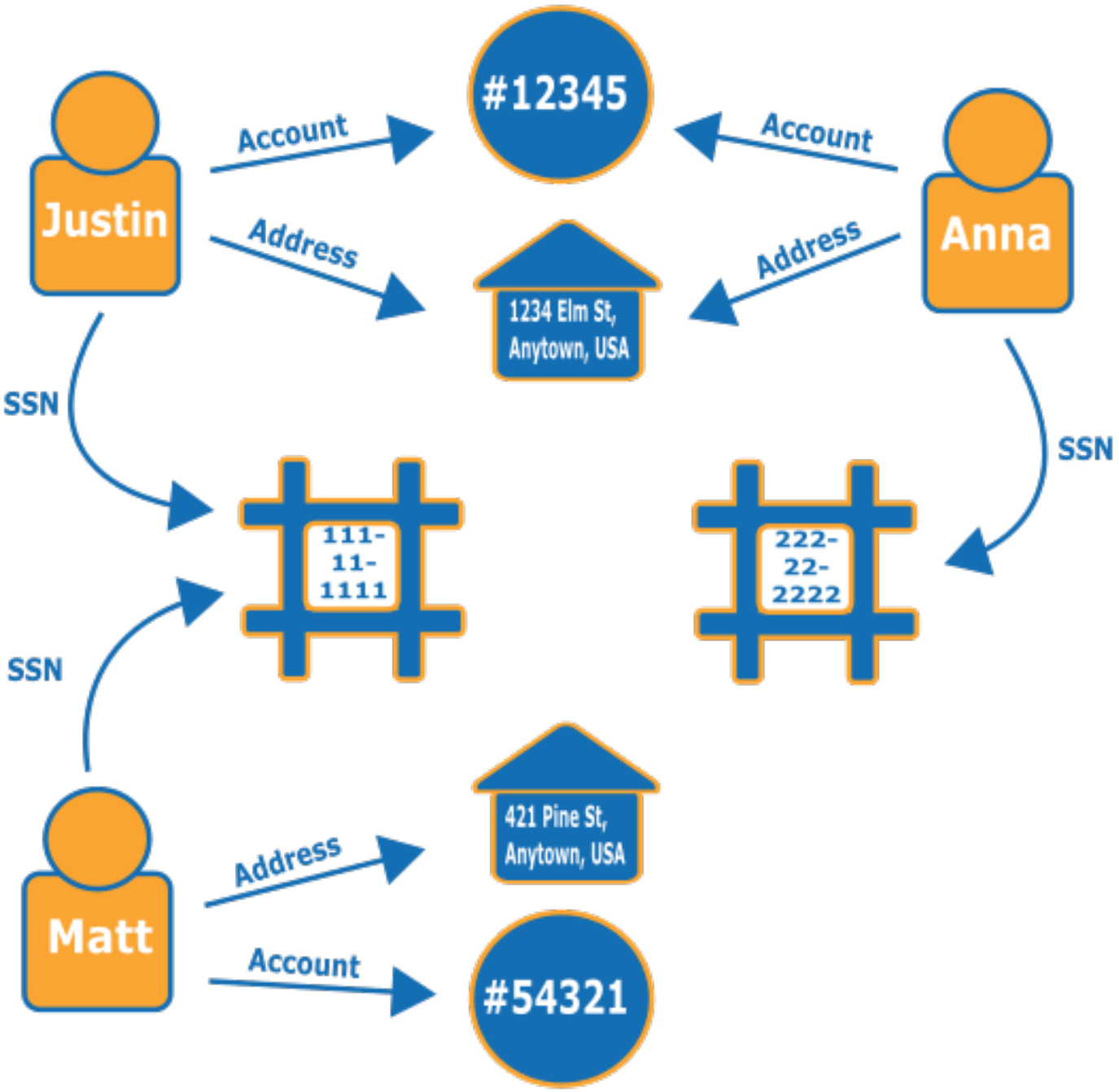
- 身份图形 - 在图形数据库中，您可以存储诸如客户兴趣、朋友和购买历史记录等信息类别之间的关系，然后查询这些数据以提出个性化和相关的推荐。

  例如，您可以使用图形数据库，根据关注相同运动内容且具有类似购买历史记录的其他人购买的产品，向用户提供产品推荐。或者，您可以识别有共同好友但还不认识对方的人员，然后提供好友推荐。

  这种图形称为身份图形，广泛用于个性化与用户之间的互动。要了解更多信息，请参阅 AWS 上的身份图形。要开始构建自己的身份图形，您可以从使用 Amazon Neptune 的身份图形示例开始。

- 欺诈图形 - 这是图形数据库的常用用途。它们可以帮助您跟踪信用卡购买和购买地点，以发现不寻常的使用情况，或者检测购买者正在尝试使用与已知欺诈案件中相同的电子邮件地址和信用卡。这些图形可以让您检查与个人电子邮件地址关联的多个人，或者多个位于不同物理位置但共享同一 IP 地址的人。

  请考虑以下图形。它显示了三个人员之间的关系以及他们的身份相关信息。每个人员都有一个地址、一个银行账户和一个身份证号。但是，我们可以看到，Matt 和 Justin 共享同一身份证号，这不正常，并表示可能其中一个人进行了诈骗。对欺诈图形的查询可以揭示此类联系，以便对其进行审查。



要了解有关欺诈图形及其使用位置的更多信息，请参阅 AWS 上的欺诈图形。

- 社交网络 – 使用图形数据库的首要也是最常见的领域之一是社交网络应用程序。

例如，假设您要构建一个馈入网站的社交媒体源。您可以轻松地使用后端的图形数据库向用户提供结果，这些结果反映了来自用户的家人、朋友、用户“喜欢”其动态的人以及住在其身边的人的最新动态。

- 行车路线 – 根据当前的路况和典型的路况模式，图形可以帮助找到从起点到目的地的最佳路线。
- 物流 – 图形可以帮助确定使用提供的运输和配送资源以满足客户要求的最有效方式。
- 诊断 - 图形可以表示复杂的诊断树，可以对这些树进行查询以确定观察到的问题和故障的来源。
- 科学研究 – 借助图形数据库，您可以构建应用程序，以使用静态加密来存储和浏览科学数据，甚至敏感的医疗信息。例如，您可以存储疾病与基因相互作用的模型。您可以在蛋白质通路中寻找图形模式，以找到可能与疾病相关的其他基因。您可以将化合物建模为图形，并查询分子结构中的模式。您可以关联不同系统内医疗记录中的患者数据。您可以按主题组织研究出版物，以便快速找到相关信息。
- 监管规则 - 您可以将复杂的监管要求存储为图形，并对其进行查询，以发现它们可能适用于您的日常业务运营的情况。
- 网络拓扑和事件 - 图形数据库可以帮助您管理和保护 IT 网络。将网络拓扑存储为图形时，还可以在网络上存储和处理许多不同类型的事件。您可以回答诸如有多少台主机在运行给定应用程序之类的问题。您可以查询可能显示给定主机已被恶意程序入侵的模式，并查询连接数据，以帮助将该程序追踪到下载该程序的原始主机。

如何查询图形？

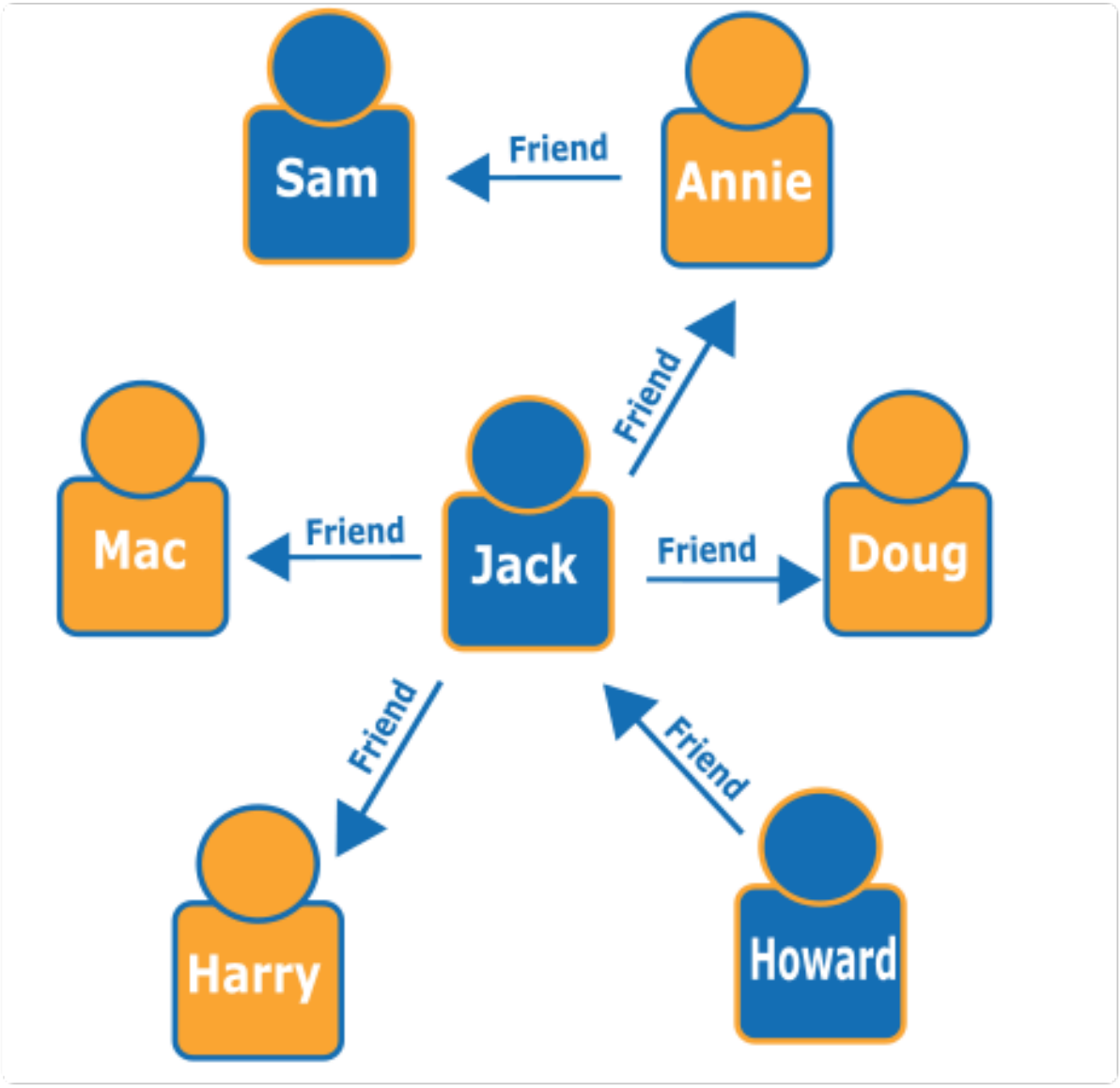
Neptune 支持三种特殊用途的查询语言，它们专为查询不同类型的图形数据而设计。您可以使用以下语言来添加、修改、删除和查询 Neptune 图形数据库中的数据：

- Gremlin 是用于属性图的图形遍历语言。Gremlin 中的查询是由离散步骤组成的遍历，每个步骤都沿着一个边缘到达一个节点。有关更多信息，请参阅 Apache TinkerPop3 中的 Gremlin 文档。

  Gremlin 的 Neptune 实施与其它实施有一些差别，尤其是在您使用 Gremlin-Groovy 时（作为序列化文本发送的 Gremlin 查询）。有关更多信息，请参阅Amazon Neptune 中的 Gremlin 标准合规性。
- openCypher – openCypher 是一种用于属性图的声明式查询语言，最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。有关语言规范，请参阅密码查询语言参考（版本 9），有关其它信息，请参阅密码风格指南。
- SPARQL 是一种用于 RDF 数据的声明性查询语言，基于由万维网联盟 (W3C) 标准化的图形模式匹配，并在 SPARQL 1.1 概述和 SPARQL 1.1 查询语言规范中描述）。有关 SPARQL 的 Neptune 实现的具体细节，请参阅Amazon Neptune 中的 SPARQL 标准合规性。

匹配 Gremlin 和 SPARQL 查询的示例

假设有如下的人员（节点）及其关系（边缘）的图形，您可以找出特定人员的“好友的好友”是谁，例如，Howard 的好友的好友。



通过观察该图形，您可以看到 Howard 有一个好友 Jack，而 Jack 有四个好友：Annie、Harry、Doug 和 Mac。这是一个包含简单图形的简单示例，但这些类型的查询可以降低复杂性、数据集大小和结果大小。

此处是一个 Gremlin 遍历查询，该查询将返回 Howard 的好友的好友的姓名：

```
g.V().has('name', 'Howard').out('friend').out('friend').values('name')
```

此处是一个 SPARQL 遍历查询，该查询将返回 Howard 的好友的好友的姓名：

```
prefix : <#>

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

> **Note**
>
> 任何资源描述框架 (RDF) 三角的各个部分都有与之关联的 URI。在上述示例中，URI 前缀经过故意缩短。

参加有关使用 Amazon Neptune 的在线课程

如果您喜欢通过视频学习，AWS 在 AWS 在线技术讲座中提供了在线课程，可帮助您开始学习。介绍图形数据库的课程是：

Amazon Neptune 中的图形数据库简介、深入研究和演示。

更深入地研究图形参考架构

当您思考图形数据库可以为您解决哪些问题以及如何解决这些问题时，最好的起点之一是 Neptune 图形参考架构 GitHub 项目。

在这里，您可以找到图形工作负载类型的详细描述，以及三个章节来帮助您设计有效的图形数据库：

- 数据模型和查询语言 – 本节将向您演练 Gremlin 和 SPARQL 之间的区别以及如何在它们之间进行选择。
- 图形数据建模 – 这是对如何做出图形数据建模决策的详尽讨论，包括使用 Gremlin 进行属性图建模和使用 SPARQL 进行 RDF 建模的详细演练。
- 将其它数据模型转换为图形模型 – 在这里，您可以了解如何将关系数据模型转换为图形模型。

还有三个章节将引导您完成使用 Neptune 的具体步骤：

- 从 Neptune VPC 外部的客户端连接到 Amazon Neptune – 本节向您展示从数据库集群所在的 VPC 外部连接到 Neptune 的几个选项。
- 通过 AWS Lambda 函数访问 Amazon Neptune – 在这里，您将了解如何通过 Lambda 函数可靠地连接到 Neptune。
- 从 Amazon Kinesis Data Streams 写入 Amazon Neptune – 本节可以帮助您使用 Neptune 处理高写入吞吐量的场景。

使用 Neptune 图形笔记本快速入门

您不必使用 Neptune 图形笔记本来处理 Neptune 图形，因此，如果您愿意，您可以继续使用 AWS CloudFormation 模板立即创建一个新的 Neptune 数据库。

同时，无论您是图形新手并想学习和体验，还是经验丰富并想要完善查询，Neptune Workbench 都提供了一个交互式开发环境 (IDE)，可以在您构建图形应用程序时提高工作效率。

Neptune 在开源 Neptune 图形JupyterLab笔记本项目和海王星工作台中提供 Jupyter GitHub 和笔记本。这些笔记本在交互式编码环境中提供示例应用程序教程和代码片段，您可以在此环境中学习图形技术和 Neptune。您可以使用它们在后端通过不同的查询语言、不同的数据集甚至不同的数据库来逐步设置、配置、填充和查询图形。

可以通过多种不同方式托管这些笔记本：

- Nep tune 工作台允许您在托管在亚马逊的完全托管的环境中运行 Jupyter 笔记本 SageMaker，并自动为您加载最新版本的 Neptune 图形笔记本项目。创建新的 Neptune 数据库时，可以轻松地在 Neptune 控制台中设置工作台。

  > **Note**
  >
  > 创建 Neptune 笔记本实例时，您可以选择两个网络访问选项：通过 Amazon 直接访问 SageMaker（默认）和通过 VPC 进行访问。无论哪种选择，笔记本电脑都需要访问互联网才能获取安装Neptune工作台的软件包依赖关系。无法访问互联网将导致 Neptune 笔记本实例的创建失败。

- 您也可以在本地安装 Jupyter。这使您可以从便携式电脑上运行笔记本，而便携式电脑连接到 Neptune 或其中一个开源图形数据库的本地实例。在后一种情况下，您可以随心所欲地尝试图形技术，而不必花一分钱。然后，准备就绪后，您可以顺利迁移到 Neptune 提供的托管式生产环境。

使用 Neptune Workbench 托管 Neptune 笔记本

Neptune 提供的 T3 和 T4g 实例类型可供您入门，每小时不到 0.10 美元。您需要通过亚马逊 SageMaker为工作台资源付费，这与 Neptune 账单是分开的。请参阅 Neptune 定价页面。Jupyter 和在 Neptune 工作台上创建的 JupyterLab 笔记本都使用 Amazon Linux 2 和 3 环境。 JupyterLab 有关 JupyterLab 笔记本支持的更多信息，请参阅 Amazon SageMaker 文档。

您可以通过以下两种方式使用 Neptune 工作台创建 Jupyter 或 JupyterLab 笔记本： AWS Management Console

- 创建新的 Neptune 数据库集群时，使用笔记本配置菜单。为此，请按照使用 AWS Management Console启动 Neptune 数据库集群中概述的步骤操作。
- 创建数据库集群后，使用左侧导航窗格中的笔记本菜单。为此，请执行以下步骤。

使用 “笔记本” 菜单创建 Jupyter 或 JupyterLab 笔记本

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在左侧的导航窗格中，选择 Notebooks (笔记本)。
3. 选择创建笔记本。
4. 在集群列表中，选择您的 Neptune 数据库集群。如果您还没有数据库集群，请选择 Create cluster (创建集群) 以创建一个。
5. 选择笔记本实例类型。
6. 为您的笔记本提供一个名称以及可选的描述。
7. 除非您已经为笔记本创建了 AWS Identity and Access Management (IAM) 角色，否则请选择创建 IAM 角色并输入 IAM 角色名称。

   > **Note**
   >
   > 如果您选择重用为之前的笔记本创建的 IAM 角色，则该角色策略必须包含访问您正在使用的 Neptune 数据库集群的正确权限。您可以通过检查资源 ARN 中 `neptune-db:*` 操作下的组件是否与该集群匹配来验证这一点。当您尝试运行笔记本魔术命令时，权限配置不正确会导致连接错误。

8. 选择创建笔记本。在一切准备就绪之前，创建过程可能需要 5 到 10 分钟。
9. 创建笔记本后，将其选中，然后选择 “打开 Jupyter” 或 “打开”。 JupyterLab

控制台可以为您的笔记本创建 AWS Identity and Access Management (IAM) 角色，您也可以自己创建一个角色。此角色的策略应包括以下内容：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::aws-neptune-notebook-(AWS region)",
        "arn:aws:s3:::aws-neptune-notebook-(AWS region)/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": [
        "arn:aws:neptune-db:(AWS region):(AWS account ID):(Neptune resource ID)/*"
      ]
    }
  ]
}
```

请注意，上述策略中的第二条语句列出了一个或多个 Neptune 集群资源 ID。

此外，角色应建立以下信任关系：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

再说一遍，一切准备就绪可能需要 5 至 10 分钟。

您可以将新笔记本配置为与 Neptune ML 结合使用，如手动为 Neptune ML 配置 Neptune 笔记本中所述。

使用 Python 将通用 SageMaker 笔记本连接到 Neptune

如果你已经安装了海王星魔法，那么将笔记本电脑连接到海王星很容易，但是即使你没有使用 SageMaker 海王星笔记本电脑，也可以使用 Python 将笔记本连接到海王星。

在笔记本电脑手机中连接到 Neptune 需要采取的步骤 SageMaker

1. 安装 Gremlin Python 客户端：

   ```
   !pip install gremlinpython
   ```

   Neptune 笔记本会为你安装 Gremlin Python 客户端，因此只有在你使用普通笔记本时才需要执行此步骤。 SageMaker
2. 编写如下代码来连接和发出 Gremlin 查询：

   ```
   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
   from gremlin_python.driver.aiohttp.transport import AiohttpTransport
   from gremlin_python.process.traversal import *
   import os

   port = 8182
   server = '(your server endpoint)'

   endpoint = f'wss://{server}:{port}/gremlin'

   graph=Graph()

   connection = DriverRemoteConnection(endpoint,'g',

    transport_factory=lambda:AiohttpTransport(call_from_event_loop=True))

   g = graph.traversal().withRemote(connection)
   ```

```
results = (g.V().hasLabel('airport')
                .sample(10)
                .order()
                .by('code')
                .local(__.values('code','city').fold())
                .toList())

# Print the results in a tabular form with a row index
for i,c in enumerate(results,1):
    print("%3d %4s %s" % (i,c[0],c[1]))

connection.close()
```

> **Note**
>
> 如果您碰巧使用的是早于 3.5.0 的 Gremlin Python 客户端，那么这行：
>
> ```
> connection = DriverRemoteConnection(endpoint,'g',
>
>  transport_factory=lambda:AiohttpTransport(call_from_event_loop=True))
> ```
>
> 只会是：
>
> ```
> connection = DriverRemoteConnection(endpoint,'g')
> ```

在 Neptune 笔记本电脑上启用 CloudWatch 日志

CloudWatch Neptune 笔记本现在默认启用日志。如果您的旧笔记本无法生成 CloudWatch 日志，请按照以下步骤手动启用日志：

1. 登录 AWS Management Console 并打开SageMaker 控制台。
2. 在左侧的导航窗格上，选择笔记本，然后选择笔记本实例。查找要为其启用日志的 Neptune 笔记本的名称。
3. 选择该笔记本实例的名称，进入详细信息页面。
4. 如果笔记本实例正在运行，请选择笔记本详细信息页面右上角的停止按钮。
5. 在权限和加密下，有一个对应于 IAM 角色 ARN 的字段。选择此字段中的链接可转到运行此笔记本实例的 IAM 角色。

6. 创建以下策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DeleteLogDelivery",
        "logs:Describe*",
        "logs:GetLogDelivery",
        "logs:GetLogEvents",
        "logs:ListLogDeliveries",
        "logs:PutLogEvents",
        "logs:PutResourcePolicy",
        "logs:UpdateLogDelivery"
      ],
      "Resource": "*"
    }
  ]
}
```

7. 保存此新策略并将其附加到在步骤 4 中找到的 IAM 角色。
8. 点击 SageMaker 笔记本实例详细信息页面右上角的 “启动”。
9. 日志开始流动后，您应该会在详细信息页面笔记本实例设置部分的左下角附近标有生命周期配置的字段下方看到查看日志链接。

如果笔记本无法启动，则 SageMaker 主机上的笔记本详细信息页面上会显示一条消息，说明笔记本实例花了 5 分钟才启动。 CloudWatch 可以在以下名称下找到与此问题相关的日志：

```
(your-notebook-name)/LifecycleConfigOnStart
```

在本地计算机上设置图形笔记本

图形笔记本项目包含有关在本地计算机上设置 Neptune 笔记本的说明：

- 先决条件

- Jupyter 和安装 JupyterLab
- 连接到图形数据库

您可以将本地笔记本连接到 Neptune 数据库集群，也可以连接到开源图形数据库的本地或远程实例。

将 Neptune 笔记本与 Neptune 集群结合使用

如果您在后端连接到 Neptune 集群，则可能需要在 Amazon 中运行笔记本电脑。 SageMaker从连接到 Neptune 比从本地安装笔记本电脑更方便，而且它 SageMaker 可以让你更轻松地使用 Neptune ML。

有关如何在中 SageMaker设置笔记本的说明，请参阅使用 Amazon 启动 graph-not ebook。 SageMaker

有关如何设置和配置 Neptune 本身的说明，请参阅设置 Neptune。

您也可以将本地安装的 Neptune 笔记本连接到 Neptune 数据库集群。这可能稍微复杂一些，因为 Amazon Neptune 数据库集群只能在设计上与外界隔离的 Amazon Virtual Private Cloud (VPC) 中创建。有多种方法可以从 VPC 外部连接到 VPC。一种是使用负载均衡器。另一种方法是使用 VPC 对等连接（请参阅 Amazon Virtual Private Cloud 对等连接指南）。

但是，对于大多数人来说，最便捷的方法是进行连接以在 VPC 中设置 Amazon EC2 代理服务器，然后使用 SSH 隧道（也称为端口转发）来连接到它。你可以在图形 GitHub 笔记本项目的`additional-databases/neptune`文件夹中将绘图笔记本本地连接到 Amazon Neptune 中找到有关如何设置的说明。

使用带有开源图形数据库的 Neptune 笔记本

要免费开始使用图形技术，还可以在后端使用带有各种开源数据库的 Neptune 笔记本。例如 TinkerPop Gremlin 服务器和 Blaz egraph 数据库。

要使用 Gremlin 服务器作为后端数据库，请按照以下说明进行操作：

- 将图形笔记本连接到 Gremlin 服务器文件夹。 GitHub
- 图形笔记本 Grem lin 配置文件夹。 GitHub

要使用 Blazegraph 的本地实例作为后端数据库，请按照以下说明进行操作：

- Blazegraph 快速入门说明
- 图形笔记本 Blaze graph 配置文件夹。 GitHub

将你的 Neptune 笔记本从 Jupyter 迁移到 3 JupyterLab

2022 年 12 月 21 日之前创建的 Neptune 笔记本使用 Amazon Linux 1 环境。您可以按照以下 AWS 博客文章中描述的步骤，将在此日期之前创建的较旧的 Jupyter 笔记本迁移到新的 Amazon Linux 2 环境 JupyterLab 3：使用亚马逊 Linux 2 将您的工作迁移到带有 Amazon Linux 2 的亚马逊 SageMaker 笔记本实例 2。

此外，还有一些步骤专门适用于将 Neptune 笔记本迁移到新环境：

Neptune 特定的先决条件

在源 Neptune 笔记本的 IAM 角色中，添加以下所有权限：

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetObject",
    "s3:ListBucket",
    "s3:CreateBucket",
    "s3:PutObject"
  ],
  "Resource": [
    "arn:aws:s3:::(your ebs backup bucket name)",
    "arn:aws:s3:::(your ebs backup bucket name)/*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "sagemaker:ListTags"
  ],
  "Resource": [
    "*"
  ]
}
```

请务必为要用于备份的 S3 桶指定正确的 ARN。

特定于 Neptune 的生命周期配置

按博客文章中所述创建第二个生命周期配置脚本以恢复备份（从 `on-create.sh`）时，生命周期名称必须遵循 `aws-neptune-*` 格式，例如 `aws-neptune-sync-from-s3`。这样可以确保在 Neptune 控制台中创建笔记本时可以选择 LCC。

从快照到新实例的 Neptune 特定同步

在博客文章中描述的从快照同步到新实例的步骤中，以下是特定于 Neptune 的更改：

- 在步骤 4 中，选择 notebook-al2-v2。
- 在步骤 5 中，重用源 Neptune 笔记本中的 IAM 角色。
- 在步骤 7 和 8 之间：
  - 在笔记本实例设置中，设置使用 `aws-neptune-*` 格式的名称。
  - 打开网络设置手风琴式折叠组件，选择与源笔记本中相同的 VPC、子网和安全组。

创建新笔记本后 Neptune 特定的步骤

1. 选择笔记本的打开 Jupyter 按钮。一旦 `SYNC_COMPLETE` 文件出现在主目录中，请继续执行下一步。
2. 前往 SageMaker 控制台中的笔记本实例页面。
3. 停止笔记本。
4. 选择编辑。
5. 在笔记本实例设置中，通过选择源 Neptune 笔记本的原始生命周期来编辑生命周期配置字段。请注意，这不是 EBS 备份生命周期。
6. 选择更新笔记本设置。
7. 再次启动笔记本。

通过对博客文章中概述的步骤进行此处描述的修改，您的图形笔记本现在应该迁移到使用 Amazon Linux 2 和 JupyterLab 3 环境的新 Neptune 笔记本实例上。它们将显示在的 Neptune 页面上以供访问和管理，现在 AWS Management Console，你可以选择 “打开 Jupyter” 或 “打开”，从上次停下来的地方继续工作。 JupyterLab

在笔记本中使用 Neptune Workbench 魔术命令

Neptune Workbench 在笔记本中提供了许多所谓的魔术命令，可以节省大量的时间和精力。它们分为两类：行魔术命令和单元格魔术命令。

行魔术命令是指前面有单个百分号 (%) 的命令。它们只接受行输入，而不接受来自单元格正文其余部分的输入。Neptune Workbench 提供了以下行魔术命令：

- %seed
- %load
- %load_ids
- %load_status
- %cancel_load
- %status
- %gremlin_status
- %opencypher_status 或 %oc_status
- %stream_viewer
- %sparql_status
- %graph_notebook_config
- %graph_notebook_host
- %graph_notebook_version
- %graph_notebook_vis_options
- %statistics
- %summary

单元格魔术命令前面有两个百分号 (%%)，而不是一个百分号，它们使用单元格内容作为输入，尽管它们也可以将行内容作为输入。Neptune Workbench 提供以下单元格魔术命令：

- %%sparql
- %%gremlin
- %%opencypher 或 %%oc
- %%graph_notebook_config

- %%graph_notebook_vis_options

还有两种魔术命令（行魔术命令和单元格魔术命令）可用来处理Neptune 机器学习：

- %neptune_ml
- %%neptune_ml

> **Note**
>
> 使用 Neptune 魔术命令时，您通常可以使用 `--help` 或 `-h` 参数获取帮助文本。有了单元格魔术命令，正文就不可能为空，所以在寻求帮助时，在正文内放上填充文字，甚至是单个字符。例如：
>
> ```
> %%gremlin --help
>   x
> ```

单元格或行魔术命令中的变量注入

可以在笔记本中的任何单元格或行魔术命令中使用以下格式引用笔记本中定义的变量：`${VAR_NAME}`。

例如，假设您定义了以下变量：

```
c = 'code'
my_edge_labels = '{"route":"dist"}'
```

然后，单元格魔术命令中的这一 Gremlin 查询：

```
%%gremlin -de $my_edge_labels
g.V().has('${c}','SAF').out('route').values('${c}')
```

等同于：

```
%%gremlin -de {"route":"dist"}
g.V().has('code','SAF').out('route').values('code')
```

适用于所有查询语言的查询参数

以下查询参数可与 Neptune Workbench 中的 %%gremlin、%%opencypher 和 %%sparql 魔术命令一起使用：

常用查询参数

- **--store-to**（或 **-s**）- 指定用于存储查询结果的变量的名称。
- **--silent** – 如果存在，则查询完成后不显示任何输出。

- **--group-by**（或 **-g**）- 指定用于对节点进行分组的属性（例如 code 或 T.region）。顶点根据其分配的组进行着色。
- **--ignore-groups** – 如果存在，则忽略所有分组选项。
- **--display-property**（或 **-d**）- 指定应为每个顶点显示其值的属性。

  每种查询语言的默认值如下：

  - 对于 Gremlin：T.label。
  - 对于 openCypher：~labels。
  - 对于 SPARQL：type。
- **--edge-display-property**（或 **-t**）- 指定应为每个边缘显示其值的属性。

  每种查询语言的默认值如下：

  - 对于 Gremlin：T.label。
  - 对于 openCypher：~labels。
  - 对于 SPARQL：type。
- **--tooltip-property**（或 **-de**）- 指定一个属性，其值应显示为每个节点的工具提示。

  每种查询语言的默认值如下：

  - 对于 Gremlin：T.label。
  - 对于 openCypher：~labels。
  - 对于 SPARQL：type。
- **--edge-tooltip-property**（或 **-te**）- 指定一个属性，其值应显示为每个边缘的工具提示。

  每种查询语言的默认值如下：

  - 对于 Gremlin：T.label。

  - 对于 openCypher：`~labels`。
  - 对于 SPARQL：`type`。
- **`--label-max-length`**（或 **`-l`**）- 指定任何顶点标签的最大字符长度。默认值为 10。
- **`--edge-label-max-length`**（或 **`-le`**）- 指定任何边缘标签的最大字符长度。默认值为 10。

  仅就 openCypher 而言，这是 `--rel-label-max-length`，或改而为 `-rel`。
- **`--simulation-duration`**（或 **`-sd`**）- 指定可视化物理模拟的最大持续时间。默认值为 1500 毫秒。
- **`--stop-physics`**（或 **`-sp`**）- 在初始模拟稳定后禁用可视化物理特性。

这些参数的属性值可以由单个属性键组成，也可以由可以为每种标签类型指定不同属性的 JSON 字符串组成。JSON 字符串只能使用变量注入来指定。

`%seed` 行魔术命令

`%seed` 行魔术命令是一种向 Neptune 端点添加数据的便捷方式，可以用它来探索和体验 Gremlin、openCypher 或 SPARQL 查询。它提供了一个表单，您可以在其中选择要浏览的数据模型（属性图或 RDF），然后从 Neptune 提供的许多不同的样本数据集中进行选择。

`%load` 行魔术命令

`%load` 行魔术命令生成一个表单，您可以使用该表单向 Neptune 提交批量加载请求（请参阅Neptune 加载程序命令）。来源必须是与 Neptune 集群位于同一区域的 Amazon S3 路径。

`%load_ids` 行魔术命令

`%load_ids` 行魔术命令检索已提交到笔记本主机端点的负载 ID（请参阅Neptune 加载程序获取状态请求参数）。请求采用以下形式：

```
GET https://your-neptune-endpoint:port/loader
```

`%load_status` 行魔术命令

`%load_status` 行魔术命令检索已提交到笔记本主机端点的特定加载任务的加载状态，由行输入指定（请参阅Neptune 加载程序获取状态请求参数）。请求采用以下形式：

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

该行魔术命令如下所示：

```
%load_status load id
```

%cancel_load 行魔术命令

%cancel_load 行魔术命令取消特定的加载任务（请参阅Neptune 加载程序取消任务）。请求采用以下形式：

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

该行魔术命令如下所示：

```
%cancel_load load id
```

%status 行魔术命令

从笔记本的主机端点（%graph_notebook_config显示主机端点）检索状态信息。

%gremlin_status 行魔术命令

检索 Gremlin 查询状态信息。

%opencypher_status 行魔术命令（也为 %oc_status）

检索 opencypher 查询的查询状态。此行魔术命令接受以下可选参数：

- **--queryId** 或 **-q** – 指定要显示其状态的正在运行的特定查询的 ID。
- **--cancel_query** 或 **-c** – 取消正在运行的查询。不取值。
- **--silent** 或 **-s** – 如果在取消查询时 --silent 设置为 true，则取消正在运行的查询，HTTP 响应代码为 200。否则，HTTP 响应代码将是 500。
- **--store-to** – 指定用于存储查询结果的变量的名称。

%sparql_status 行魔术命令

检索 SPARQL 查询状态信息。

%stream_viewer 行魔术命令

如果在 Neptune 集群上启用了流，则 %stream_viewer 行魔术命令显示一个界面，允许交互式浏览 Neptune 流中记录的条目。它接受以下可选参数：

- **language** – 流数据的查询语言：gremlin 或 sparql。如果您不提供此参数，则默认值为 gremlin。
- **--limit** – 指定每页显示的最大流条目数。如果您不提供此参数，则默认值为 10。

> **Note**
>
> 只有引擎版本 1.0.5.1 及更早版本才完全支持 %stream_viewer 行魔术命令。

%graph_notebook_config 行魔术命令

此行魔术命令显示一个 JSON 对象，其中包含笔记本用于与 Neptune 通信的配置。配置包括：

- host：要连接和向其发出命令的端点。
- port：向 Neptune 发出命令时使用的端口。默认值为 8182。
- auth_mode：向 Neptune 发出命令时要使用的身份验证模式。如果连接到启用了 IAM 身份验证的集群，则必须为 IAM，否则为 DEFAULT。
- load_from_s3_arn：指定一个供 %load 魔术命令使用的 Amazon S3 ARN。如果此值为空，则必须在 %load 命令中指定 ARN。
- ssl：表示是否使用 TLS 连接到 Neptune 的布尔值。默认值为 true。
- aws_region：部署此笔记本的区域。此信息用于 IAM 身份验证和 %load 请求。

您可以通过将 %graph_notebook_config 输出复制到新单元格中并在那里对其进行更改来更改配置。然后，如果您在新的单元格上运行 %%graph_notebook_config 单元格魔术命令，则配置将相应更改。

%graph_notebook_host 行魔术命令

将行输入设置为笔记本的主机。

%graph_notebook_version 行魔术命令

%graph_notebook_version 行魔术命令返回 Neptune Workbench 笔记本的版本号。例如，版本 1.27 中引入了图形可视化。

%graph_notebook_vis_options 行魔术命令

%graph_notebook_vis_options 行魔术命令显示笔记本使用的当前可视化设置。vis.js 文档中对这些选项进行了说明。

您可以修改这些设置，方法是将输出复制到新单元格中，进行所需的更改，然后在单元格上运行 %%graph_notebook_vis_options 单元格魔术命令。

要将可视化设置恢复为其默认值，您可以使用 reset 参数运行 %graph_notebook_vis_options 行魔术命令。这将重置所有可视化设置：

```
%graph_notebook_vis_options reset
```

%statistics 行魔术命令

%statistics 行魔术命令用于检索或管理 DFE 引擎统计数据（请参阅管理 Neptune DFE 要使用的统计数据）。这个魔术命令也可用来检索图形摘要。

它接受以下参数：

- **--language**– 统计端点的查询语言：propertygraph（或 pg）或 rdf。

  如果未提供，则默认为 propertygraph。
- **--mode**（或 **-m**）- 指定要提交的请求或操作的类型：status、disableAutoCompute、enableAutoCompute、refresh、delete、detailed 或 basic 之一。

  如果未提供，则默认值为 status，除非指定了 --summary，在这种情况下，默认值为 basic。
- **--summary** – 从所选语言的统计摘要端点检索图形摘要。
- **--silent** – 如果存在，则查询完成后不显示任何输出。
- **--store-to** – 用于指定用来存储查询结果的变量。

%summary 行魔术命令

%summary 行魔术命令用于检索图形摘要信息。它从 Neptune 引擎版本 1.2.1.0 开始提供。

它接受以下参数：

- **--language**– 统计端点的查询语言：propertygraph（或 pg）或 rdf。

  如果未提供，则默认为 propertygraph。
- **--detailed** – 在输出中打开或关闭结构字段的显示。

  如果未提供，则默认为 basic 摘要显示模式。
- **--silent** – 如果存在，则查询完成后不显示任何输出。
- **--store-to** – 用于指定用来存储查询结果的变量。

%%graph_notebook_config 单元格魔术命令

如果可能，%%graph_notebook_config 单元格魔术命令使用包含配置信息的 JSON 对象来修改笔记本用于与 Neptune 通信的设置。配置采用%graph_notebook_config行魔术命令返回的相同形式。

例如：

```
%%graph_notebook_config
{
  "host": "my-new-cluster-endpoint.amazon.com",
  "port": 8182,
  "auth_mode": "DEFAULT",
  "load_from_s3_arn": "",
  "ssl": true,
  "aws_region": "us-east-1"
}
```

%%sparql 单元格魔术命令

%%sparql 单元格魔术命令向 Neptune 端点发出 SPARQL 查询。它接受以下可选行输入：

- **-h** 或 **--help** – 返回有关这些参数的帮助文本。
- **--path**– 为指向 SPARQL 端点的路径添加前缀。例如，如果您指定 --path "abc/def"，则调用的端点将是 *host*:*port*/abc/def。

- **`--expand-all`** – 这是一个查询可视化提示，它告诉可视化工具在图形示意图中包含所有 `?s ?p ?o` 结果，而无论绑定类型如何。

  默认情况下，SPARQL 可视化仅包含三重模式，其中 o? 为 `uri` 或 bnode（空白节点）。所有其它 ?o 绑定类型（例如文本字符串或整数）都视为 ?s 节点的属性，可以使用图形选项卡中的详细信息窗格查看这些属性。

  如果您想改为在可视化中包含顶点等文本值，请使用 `--expand-all` 查询提示。

  请勿将此可视化提示与 explain 参数结合使用，因为 explain 查询不会可视化。
- **`--explain-type`** – 用于指定要使用的 explain 模式（`dynamic`、`static` 或 `details` 之一）。
- **`--explain-format`** – 用于指定 explain 查询的响应格式（`text/csv` 或 `text/html` 之一）。
- --store-to – 用于指定用来存储查询结果的变量。

`explain` 查询示例：

```
%%sparql explain

SELECT * WHERE {?s ?p ?o} LIMIT 10
```

带有可视化提示参数的 `--expand-all` 可视化查询示例（请参阅SPARQL 可视化）：

```
%%sparql --expand-all

SELECT * WHERE {?s ?p ?o} LIMIT 10
```

`%%gremlin` 单元格魔术命令

`%%gremlin`细胞魔法使用向海王星端点发出 Gremlin 查询。 WebSocket它接受可选的行输入以切换到 Gremlin `explain` 模式或 Gremlin `profile` API，以及用于修改可视化输出行为的单独可选可视化提示输入（请参阅Gremlin 可视化）。

`explain` 查询示例：

```
%%gremlin explain

g.V().limit(10)
```

profile 查询示例：

```
%%gremlin profile

g.V().limit(10)
```

带有可视化查询提示的可视化查询示例：

```
%%gremlin -p v,outv

g.V().out().limit(10)
```

%%gremlin profile 查询的可选参数

- **--chop** – 指定 profile 结果字符串的最大长度。如果您不提供此参数，则默认值为 250。
- **--serializer** – 指定要用于结果的序列化程序。允许的值是任何有效的 MIME 类型或 TinkerPop 驱动程序 “序列化器” 枚举值。如果您不提供此参数，则默认值为 application.json。
- **--no-results** – 仅显示结果计数。如果不使用，则默认情况下，所有查询结果都会显示在配置文件报告中。
- **--indexOps** – 显示所有索引操作的详细报告。

%%opencypher 单元格魔术命令（也为 %%oc）

%%opencypher 单元格魔术命令（也具有缩写 %%oc 形式）向 Neptune 端点发出 openCypher 查询。它接受以下可选的行输入参数：

- mode – 查询模式：query 或 bolt。如果您不提供此参数，则默认值为 query。
- **--group-by** 或 **-g** – 指定用于对节点进行分组的属性。例如，code, ~id。如果您不提供此参数，则默认值为 ~labels。
- **--ignore-groups** – 如果存在，则忽略所有分组选项。
- **--display-propery** 或 **-d** - 指定应为每个顶点显示其值的属性。如果您不提供此参数，则默认值为 ~labels。
- **--edge-display-propery** 或 **-de** - 指定应为每个边缘显示其值的属性。如果您不提供此参数，则默认值为 ~labels。
- **--label-max-length** 或 **-l** – 指定要显示的顶点标签的最大字符数。如果您不提供此参数，则默认值为 10。

- **--store-to** 或 **-s** - 指定用于存储查询结果的变量的名称。
- **--plan-cache** 或 **-pc** - 指定要使用的计划缓存模式。默认值为auto。 （*计划缓存仅适用于 Neptune Analytics）
- **--query-timeout** 或 **-qt** - 指定最大查询超时时间（以毫秒为单位）。默认值为 1800000。
- **--query-parameters** 或 **qp** - 要应用于查询的参数定义。此选项可以接受单个变量名称，也可以接受映射的字符串表示。

  --query-parameters 的用法示例

  1. 在一个笔记本单元格中定义 openCypher 参数的映射。

  ```
  params = '''{
   "name":"john",
   "age": 20,
  }'''
  ```

  2. 使用 %%oc 将参数传递到另一个单元格的 --query-parameters 中。

  ```
  %%oc --query-parameters params

  MATCH (n {name: $name, age: $age})
  RETURN n
  ```

- --explain-type — 用于指定要使用的解释模式（其中之一：动态、静态或细节）。

%%graph_notebook_vis_options 单元格魔术命令

%%graph_notebook_vis_options 单元格魔术命令可让您为笔记本设置可视化选项。您可以将 %graph-notebook-vis-options 行魔术命令返回的设置复制到新的单元格中，对其进行更改，然后使用 %%graph_notebook_vis_options 单元格魔术命令设置新值。

vis.js 文档中对这些选项进行了说明。

要将可视化设置恢复为其默认值，您可以使用 reset 参数运行 %graph_notebook_vis_options 行魔术命令。这将重置所有可视化设置：

```
%graph_notebook_vis_options reset
```

%neptune_ml 行魔术命令

可以使用 %neptune_ml 行魔术命令来启动和管理各种 Neptune ML 操作。

> **Note**
>
> 还可以使用 %%neptune_ml 单元格魔术命令启动和管理一些 Neptune ML 操作。

- **%neptune_ml export start** – 开始新的导出任务。

  参数

  - **--export-url** *exporter-endpoint* - (可选) 可以在其中调用导出程序的 Amazon API Gateway 端点。
  - **--export-iam** – (可选) 表示必须使用 SigV4 对导出 url 的请求进行签名的标志。
  - **--export-no-ssl** – (可选) 表示在连接到导出程序时不应使用 SSL 的标志。
  - **--wait** – (可选) 指示操作应等到导出完成的标志。
  - **--wait-interval***interval-to-wait*— (可选) 以秒为单位设置两次导出状态检查之间的时间 (默认值：60) 。
  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待导出任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。
  - **--store-to***location-to-store-result*— (可选) 用于存储导出结果的变量。如果指定 --wait，则最终状态将存储在那里。
- **%neptune_ml export status** – 检索导出任务的状态。

  参数

  - **--job-id** *export job ID* – 要检索其状态的导出任务的 ID。
  - **--export-url** *exporter-endpoint* - (可选) 可以在其中调用导出程序的 Amazon API Gateway 端点。
  - **--export-iam** – (可选) 表示必须使用 SigV4 对导出 url 的请求进行签名的标志。
  - **--export-no-ssl** – (可选) 表示在连接到导出程序时不应使用 SSL 的标志。
  - **--wait** – (可选) 指示操作应等到导出完成的标志。
  - **--wait-interval***interval-to-wait*— (可选) 以秒为单位设置两次导出状态检查之间的时间 (默认值：60) 。

  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待导出任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。
  - **--store-to*location-to-store-result*** — (可选) 用于存储导出结果的变量。如果指定 `--wait`，则最终状态将存储在那里。
- **%neptune_ml dataprocessing start** – 启动 Neptune ML 数据处理步骤。

  参数

  - **--job-id** *ID for this job* – (可选) 分配给此任务的 ID。
  - **--s3-input-uri** *S3 URI* – (可选) 用于查找此数据处理任务的输入的 S3 URI。
  - **--config-file-name** *file name* – (可选) 此数据处理任务的配置文件的名称。
  - **--store-to*location-to-store-result*** — (可选) 用于存储数据处理结果的变量。
  - **--instance-type** *(instance type)* - (可选) 用于此数据处理任务的实例大小。
  - **--wait** – (可选) 指示操作应等到数据处理完成的标志。
  - **--wait-interval*interval-to-wait*** — (可选) 以秒为单位设置两次数据处理状态检查之间的时间 (默认值：60) 。
  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待数据处理任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。
- **%neptune_ml dataprocessing status** – 检索数据处理任务的状态。

  参数

  - **--job-id** *ID of the job* – 要检索其状态的任务的 ID。
  - **--store-to** *instance type* - (可选) 用于存储模型训练结果的变量。
  - **--wait** – (可选) 指示操作应等到模型训练完成的标志。
  - **--wait-interval*interval-to-wait*** — (可选) 设置模型训练状态检查之间的时间 (以秒为单位) (默认值：60) 。
  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待数据处理任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。
- **%neptune_ml training start**– 启动 Neptune ML 模型训练过程。

  参数

  - **--job-id** *ID for this job* – (可选) 分配给此任务的 ID。
  - **--data-processing-id** *dataprocessing job ID* – (可选) 创建用于训练的构件的数据处理任务的 ID。

- **--s3-output-uri** *S3 URI* – (可选) 用于存储此模型训练任务的输出的 S3 URI。
- **--instance-type** *(instance type)* – (可选) 用于此模型训练任务的实例大小。
- **--store-to***location-to-store-result*— (可选) 用于存储模型训练结果的变量。
- **--wait** – (可选) 指示操作应等到模型训练完成的标志。
- **--wait-interval***interval-to-wait*— (可选) 设置模型训练状态检查之间的时间 (以秒为单位) (默认值：60) 。
- **--wait-timeout** *timeout-seconds* – (可选) 设置等待模型训练任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。

- **%neptune_ml training status** – 检索 Neptune ML 模型训练任务的状态。

  参数

  - **--job-id** *ID of the job* – 要检索其状态的任务的 ID。
  - **--store-to** *instance type* - (可选) 用于存储状态结果的变量。
  - **--wait** – (可选) 指示操作应等到模型训练完成的标志。
  - **--wait-interval***interval-to-wait*— (可选) 设置模型训练状态检查之间的时间 (以秒为单位) (默认值：60) 。
  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待数据处理任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。

- **%neptune_ml endpoint create**– 为 Neptune ML 模型创建查询端点。

  参数

  - **--job-id** *ID for this job* – (可选) 分配给此任务的 ID。
  - **--model-job-id** *model-training job ID* – (可选) 要为其创建查询端点的模型训练任务的 ID。
  - **--instance-type** *(instance type)* – (可选) 用于查询端点的实例大小。
  - **--store-to***location-to-store-result*— (可选) 用于存储端点创建结果的变量。
  - **--wait** – (可选) 指示操作应等到端点创建完成的标志。
  - **--wait-interval***interval-to-wait*— (可选) 设置两次状态检查之间的时间，以秒为单位 (默认值：60) 。
  - **--wait-timeout** *timeout-seconds* – (可选) 设置等待端点创建任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。

- **%neptune_ml endpoint status** – 检索 Neptune ML 查询端点的状态。

参数

- **--job-id** *endpoint creation ID* – (可选) 要报告其状态的端点创建任务的 ID。
- **--store-to***location-to-store-result*— (可选) 用于存储状态结果的变量。
- **--wait** – (可选) 指示操作应等到端点创建完成的标志。
- **--wait-interval***interval-to-wait*— (可选) 设置两次状态检查之间的时间，以秒为单位 (默认值：60) 。
- **--wait-timeout** *timeout-seconds* – (可选) 设置等待端点创建任务完成后再返回最新状态的时间 (以秒为单位，默认值：3600) 。

%%neptune_ml 单元格魔术命令

%%neptune_ml 单元格魔术命令会忽略行输入，例如 --job-id 或 --export-url。相反，它允许您在单元格正文内提供这些输入和其它输入。

也可以将此类输入保存在另一个单元格中，分配给 Jupyter 变量，然后使用该变量将它们注入单元格正文中。这样，可以一遍又一遍地使用这样的输入，而不必每次都重新输入。

只有当注入变量是单元格的唯一内容时，这才起作用。不能在一个单元格中使用多个变量，也不能使用文本和变量的组合。

例如，%%neptune_ml export start 单元格魔术命令可以在单元格正文中使用包含用于控制 Neptune 导出过程的参数中描述的所有参数的 JSON 文档。

在 Neptune-ML-01-Introduction-to-Node-Classification-Gremlin 笔记本中，在导出数据和模型配置部分的配置特征下，您可以看到以下单元格如何在分配给名为 export-params 的 Jupyter 变量的文档中保存导出参数：

```
export_params = {
  "command": "export-pg",
  "params": {
    "endpoint": neptune_ml.get_host(),
    "profile": "neptune_ml",
    "useIamAuth": neptune_ml.get_iam(),
    "cloneCluster": False
   },
  "outputS3Path": f'{s3_bucket_uri}/neptune-export',
  "additionalParams": {
    "neptune_ml": {
```

```
      "targets": [
        {
          "node": "movie",
          "property": "genre"
        }
      ],
      "features": [
        {
          "node": "movie",
          "property": "title",
          "type": "word2vec"
        },
        {
          "node": "user",
          "property": "age",
          "type": "bucket_numerical",
          "range" : [1, 100],
          "num_buckets": 10
        }
      ]
    }
  },
  "jobSize": "medium"}
```

运行此单元格时，Jupyter 会以该名称保存该参数文档。然后，可以使用 `${export_params}` 将 JSON 文档注入到 `%%neptune_ml export start cell` 的正文中，如下所示：

```
%%neptune_ml export start --export-url {neptune_ml.get_export_service_host()} --export-
iam --wait --store-to export_results

${export_params}
```

%%neptune_ml 单元格魔术命令的可用形式

%%neptune_ml 单元格魔术命令可以按下列形式使用：

- **%%neptune_ml export start** – 启动 Neptune ML 导出过程。
- **%%neptune_ml dataprocessing start**– 启动 Neptune ML 数据处理任务。
- **%%neptune_ml training start**– 启动 Neptune ML 模型训练任务。
- **%%neptune_ml endpoint create**– 为模型创建 Neptune ML 查询端点。

Neptune Workbench 中的图形可视化

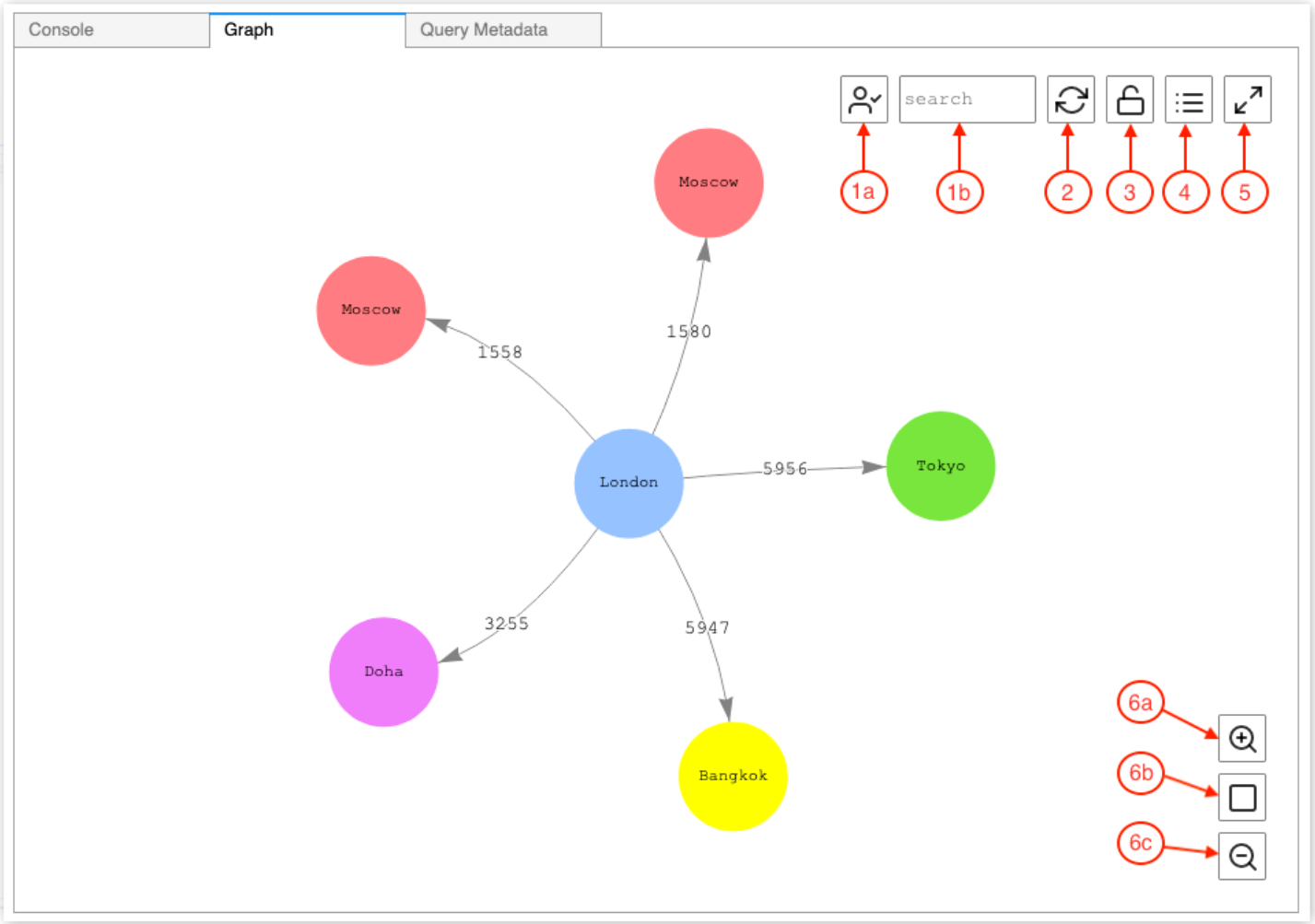
在许多情况下，Neptune Workbench 可以创建查询结果的可视化图形，并以表格形式返回这些结果。只要可以进行可视化，就可以在查询结果的图形选项卡中使用图形可视化。

除了此处描述的内置可视化功能外，还可以在 Neptune 图形笔记本上使用更高级的可视化工具。

> **Note**
>
> 要访问在您已在使用的笔记本中最近添加的功能和修复程序，请先停止，然后重新启动您的笔记本实例。

图形选项卡界面概述

此图标识了“图形”选项卡中存在的用户界面元素：



1. 图形搜索
   a. UUID 切换：切换图形搜索中是否包含 ID 属性值。默认情况下启用 ID 包含。如果禁用，则 ID 属性匹配（包括引用节点 ID 的边缘属性）不会导致元素突出显示。
   b. 搜索文本字段：突出显示包含您在此处指定的文本字符串的所有顶点和边缘属性值。
2. 图形重置 – 重新运行图形物理特性模拟，并设置缩放以适合窗口中的图形大小。
3. 切换图形物理特性 - 切换图形物理特性模拟的运行。默认情况下，物理特性处于启用状态，允许图形动态变化。如果禁用，则移动其它顶点时，顶点将保持锁定位置。
4. 详细信息视图 - 选择节点或边缘后，这将显示该元素的属性键和值的列表（如果查询结果中有）。
5. 全屏视图 - 展开图形选项卡窗口以适合屏幕。再次单击可最小化图形选项卡。
6. 缩放选项
   a. 放大
   b. 缩放重置：将缩放设置为适合图形选项卡窗口中的所有顶点。
   c. 缩小

可视化 Gremlin 查询结果

Neptune Workbench 为任何返回 `path` 的 Gremlin 查询创建查询结果的可视化。要查看可视化，请在运行查询后，选择查询下方控制台选项卡右侧的图形选项卡。

您可以使用查询可视化提示来控制可视化工具图形查询输出的方式。这些提示遵循 `%%gremlin` 单元格魔术命令，前面有 `--path-pattern`（或其简写形式 `-p`）参数名称：

```
%%gremlin -p comma-separated hints
```

您也可以使用 `--group-by`（或 `-g`）标志来指定顶点的属性以对其进行分组。这允许为不同的顶点组指定颜色或图标。

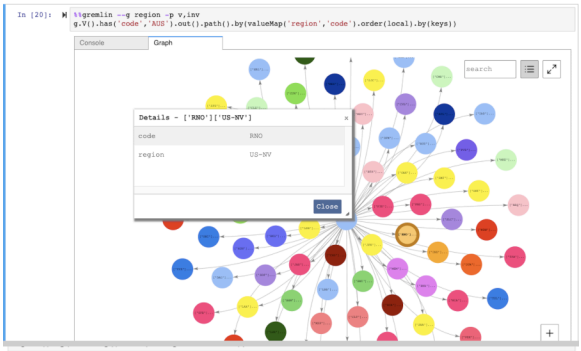
提示的名称反映了在顶点之间遍历时常用的 Gremlin 步骤，它们的行为也是相对应的。多个提示可以组合使用，用逗号分隔，中间没有任何空格。使用的提示应与正在可视化的查询中相应的 Gremlin 步骤相匹配。示例如下：

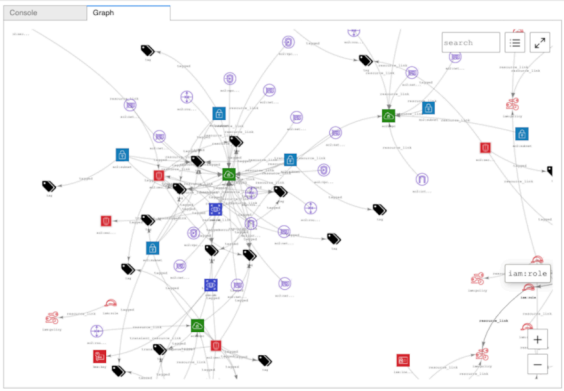
```
%%gremlin -p v,oute,inv
g.V().hasLabel('airport').outE().inV().path().by('code').by('dist').limit(5)
```

可用的可视化提示如下：

```
v
inv
outv
e
ine
oute
```

以下是一些使用组进行图形可视化的示例：





可视化 SPARQL 查询结果

Neptune Workbench 可为采用以下任一形式的 SPARQL 查询创建查询结果的可视化：

- `SELECT ?subject ?predicate ?object`
- `SELECT ?s ?p ?o`

要查看可视化，请在运行查询后，选择查询下方表选项卡右侧的图形选项卡。

默认情况下，SPARQL 可视化仅包含三重模式，其中 o? 为 `uri` 或 bnode（空白节点）。所有其它 ?o 绑定类型（例如文本字符串或整数）都视为 `?s` 节点的属性，可以使用图形选项卡中的详细信息窗格查看这些属性。

但是，在许多情况下，您可能希望在可视化中包含诸如顶点之类的文本值。为此，请在 `%%sparql` 单元格魔术命令之后使用 `--expand-all` 查询提示：

```
%%sparql --expand-all
```

这会告诉可视化工具在图形示意图中包含所有 `?s ?p ?o` 结果，而无论绑定类型如何。

您可以在整个 `Air-Routes-SPARQL.ipynb` 笔记本中看到这个提示，并可以通过运行带有和不带提示的查询来进行实验，看看它在可视化中有什么不同。

在 Neptune Workbench 中访问可视化教程笔记本

Neptune Workbench 附带的两个可视化教程笔记本在 Gremlin 和 SPARQL 中提供了大量示例，说明如何有效地查询图形数据并可视化结果。

导航到可视化笔记本

1. 在左侧的导航窗格中，选择右侧的打开笔记本按钮。
2. 一旦 Neptune Workbench 打开，会运行 Jupyter，您将在顶层看到一个 Neptune 文件夹。选择该文件夹以打开它。
3. 下一级是一个名为 02-Visualization 的文件夹。打开此文件夹。里面有几个笔记本，它们引导您了解在 Gremlin 和 SPARQL 中查询图形数据的不同方法，以及如何可视化查询结果：

   - Air-Routes-Gremlin
   - Air-Routes-SPARQL
   - Workbench Visualization blog
   - EPL-Gremlin
   - EPL-SPARQL

   选择一个笔记本来尝试它所包含的查询。

设置 Neptune

欢迎使用 Amazon Neptune。本节可帮助您创建新的 Neptune 数据库集群和在 Neptune 文档中查找所需的内容。

> **Note**
>
> 有关 AWS 图形数据库参考架构和参考部署架构，请参阅 Amazon Neptune 资源。这些资源可帮助您了解有关选择图形数据模型和查询语言的信息，并加快开发过程。

主题

选择正确的 Neptune 数据库实例类型

Amazon Neptune 提供了许多不同的实例大小和系列，它们提供了适合不同图形工作负载的不同功能。本节旨在帮助您选择最适合您需求的实例类型。

有关这些系列中每种实例类型的定价，请参阅 Neptune 定价页面。

实例资源分配概述

Neptune 中使用的每种 Amazon EC2 实例类型和大小都提供一定数量的计算内存 (vCPU) 和系统内存。Neptune 的主存储位于集群中数据库实例的外部，这使得计算和存储容量可以相互独立扩展。

本节重点介绍如何扩展计算资源，以及每种不同实例系列之间的差异。

在所有实例系列中，都将分配 vCPU 资源以便每个 vCPU 支持两 (2) 个查询执行线程。这种支持由实例大小决定。在确定给定 Neptune 数据库实例的适当大小时，您需要考虑应用程序可能的并发性以及查询的平均延迟。您可以按如下方式估算所需的 vCPU 数量，其中延迟按平均查询延迟（以秒为单位）来衡量，并发度按每秒的目标查询数来衡量：

$$vCPUs = \frac{latency \times concurrency}{2}$$

> **Note**
>
> 在某些情况下，使用 DFE 查询引擎的 SPARQL 查询、openCypher 查询和 Gremlin 读取查询可以为每个查询使用多个执行线程。在最初调整数据库集群大小时，首先假设每个查询每次执行将消耗单个执行线程，如果您观察到查询队列中有背压，则可以纵向扩展。这可以通过使用`/gremlin/status/oc/status`、或 `/sparql/status` API 来观察，也可以使用`MainRequestsPendingRequestsQueue` CloudWatch 指标进行观察。

每个实例上的系统内存分为两个主要分配：缓冲池缓存和查询执行线程内存。

实例中大约有三分之二的可用内存分配给缓冲池缓存。缓冲池缓存用于缓存图形中最近使用的组件，以便更快地访问重复访问这些组件的查询。系统内存量较大的实例具有较大的缓冲池缓存，可以在本地存储更多的图形。用户可以通过监控中提供的缓冲区缓存命中和未命中指标来调整缓冲池缓存的适当量。CloudWatch

如果缓存命中率持续降至 99.9% 以下，则可能需要增加实例的大小。这表明缓冲池不够大，引擎不得不更高效、更频繁地从底层存储卷提取数据。

其余三分之一的系统内存在各个查询执行线程之间均匀分布，一些内存留给操作系统，还有一个小型动态池供线程根据需要使用。每个线程的可用内存从一个实例大小略微增加到下一个实例大小，直至 `8xl` 实例类型，达到该大小后，每个线程分配的内存达到最大值。

当您遇到 `OutOfMemoryException` (OOM) 时，则是时候添加更多线程内存了。当一个线程需要的内存超过分配给它的最大内存时，就会出现 OOM 异常（这与整个实例耗尽内存不同）。

t3 和 t4g 实例类型

t3 和 t4g 实例系列为开始使用图形数据库以及初始开发和测试提供了一种低成本的选项。这些实例有资格享受 Neptune 免费套餐优惠，该优惠允许新客户在独立 AWS 账户中使用的前 750 个实例小时内免费使用 Neptune，或者累计到具有整合账单的 AWS 组织（付款人账户）下。

t3 和 t4g 实例仅在中型配置（t3.medium 和 t4g.medium）中提供。

它们不适用于生产环境。

由于这些实例的资源非常有限，因此不建议将其用于测试查询执行时间或数据库整体性能。要评测查询性能，请升级到其它实例系列之一。

r4 实例类型系列

已弃用 – r4 系列是在 2018 年 Neptune 推出时提供的，但现在更新的实例类型提供了高得多的性价比。从引擎版本 1.1.0.0 开始，Neptune 不再支持 r4 实例类型。

r5 实例类型系列

r5 系列包含内存优化型实例类型，适用于大多数图形用例。r5 系列包含的实例类型从 r5.large 直至 r5.24xlarge。随着大小增加，它们的计算性能会线性扩展。例如，一个 r5.xlarge（4 个 vCPU 和 32GiB 内存）的 vCPU 和内存是 r5.large（2 个 vCPU 和 16GiB 内存）的两倍，而一个 r5.2xlarge（8 个 vCPU 和 64GiB 内存）的 vCPU 和内存是 r5.xlarge 的两倍。您可以预期查询性能会随着计算容量而直接扩展，直至 r5.12xlarge 实例类型。

r5 实例系列采用双插槽 Intel CPU 架构。r5.12xlarge 和更小的类型使用单插槽和该单插槽处理器拥有的系统内存。r5.16xlarge 和 r5.24xlarge 类型使用这两个插槽和可用内存。由于在双插槽架构中，两个物理处理器之间需要一些内存管理开销，因此从 r5.12xlarge 扩展到 r5.16xlarge 或 r5.24xlarge 实例类型的性能增益并不像在较小大小上纵向扩展时那样线性。

r5d 实例类型系列

Neptune 具有查找缓存特征，可用于提高需要提取和返回大量属性值和文本的查询的性能。此特征主要由需要返回许多属性的查询的客户使用。查找缓存通过在本地提取这些属性值，而不是在 Neptune 索引存储中一遍又一遍地查找每个属性值，来提高这些查询的性能。

查找缓存是在 r5d 实例类型上使用 NVMe 连接的 EBS 卷来实现的。它是使用集群的参数组启用的。从 Neptune 索引存储中提取数据时，属性值和 RDF 文本值会缓存在此 NVMe 卷中。

如果您不需要查找缓存特征，请使用标准 r5 实例类型而不是 r5d，以避免更高的 r5d 成本。

r5d 系列的实例类型与 r5 系列的大小相同（从 r5d.large 到 r5d.24xlarge）。

r6g 实例类型系列

AWS 开发了自己的基于ARM的处理器，名为 Graviton，其性价比比英特尔和AMD同类处理器更好。r6g 系列使用 Graviton2 处理器。在我们的测试中，Graviton2 处理器在 OLTP 风格（受限）图形查询方面的性能提高了 10-20%。但是，由于内存分页性能略低，使用 Graviton2 处理器时更大的 OLAP 类查询的性能可能略低于 Intel 处理器。

还需要注意的是，r6g 系列采用单插槽架构，这意味着随着计算容量从 r6g.large 到 r6g.16xlarge（该系列中的最大类型），性能将线性扩展。

r6i 实例类型系列

Amazon R6i 实例由第三代 Intel Xeon 可扩展处理器（代号为 Ice Lake）提供支持，非常适合内存密集型工作负载。一般来说，与同类的 R5 实例类型相比，它们的计算性价比高出多达 15%，每个 vCPU 的内存带宽高出多达 20%。

x2g 实例类型系列

当实例的缓冲池缓存较大时，某些图形用例的性能会更好。推出 x2g 系列是为了更好地支持这些用例。该x2g系列 memory-to-v的 CPU 比r5或r6g系列高。x2g 实例还使用 Graviton2 处理器，具有许多与 r6g 实例类型相同的性能特征，而且缓冲池缓存更大。

如果您使用的是 CPU 利用率低且缓冲池缓存未命中率高的 r5 或 r6g 实例类型，请尝试改用 x2g 系列。这样，您就可以获得所需的额外内存，而无需为更多 CPU 容量付费。

serverless 实例类型

Neptune 无服务器特征可以根据工作负载的资源需求动态扩展实例大小。Neptune 无服务器可让您为数据库集群中的实例设置计算容量下限和上限（以 Neptune 容量单位衡量），而不是计算应用程序需要多少个 vCPU。使用无服务器实例而不是预调配实例可以对具有不同利用率的工作负载进行成本优化。

您可以在同一个数据库集群中同时设置预调配实例和无服务器实例，以实现最佳性价比配置。

为 Neptune 数据库集群选择正确的存储类型

Neptune 提供两种具有不同定价模式的存储类型：

- 标准存储 - 标准存储为 I/O 使用率为中到低的应用程序提供经济实惠的数据库存储。
- I/O — 优化存储 — 借助引擎版本 1.3.0.0 中提供的 I/O 优化存储，您只需为正在使用的存储和实例付费。存储成本高于标准存储，而且您无需为所使用的 I/O 支付任何费用。如果您的 I/O 使用率很高，则预调配 IOPs 存储可以显著降低成本。

  I/O 优化存储符合 I/O 密集型图形工作负载的需求，且成本可预测，I/O 延迟低，I/O 吞吐量一致。每 30 天只能在 I/O 优化存储类型和标准存储类型之间切换一次。

  有关 I/O 优化存储的定价信息，请参阅 Neptune 定价页面。以下部分介绍如何为 Neptune 数据库集群设置 I/O 优化存储。

为 Neptune 数据库集群选择 I/O 优化存储

默认情况下，Neptune 数据库集群使用标准存储。您可以在创建数据库集群时对其启用 I/O 优化存储，如下所示：

以下是一个示例，说明如何在使用 AWS CLI创建集群时启用 I/O 优化存储：

```
aws neptune create-db-cluster \
  --database-name (name for the new database) \
  --db-cluster-identifier (an ID for the cluster) \
  --engine neptune \
  --engine-version 1.3.0.0 \
  --storage-type iopt1
```

然后，您自动创建的任何实例都启用 I/O 优化存储：

```
aws neptune create-db-instance \
  --db-cluster-identifier (the ID of the new cluster) \
  --db-instance-identifier (an ID for the new instance) \
  --engine neptune \
  --db-instance-class db.r5.large
```

您也可以修改现有数据库集群，以对其启用 I/O 优化存储，如下所示：

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (the ID of a cluster without I/O-Optimized storage) \
  --storage-type iopt1 \
  --apply-immediately
```

您可以将备份快照还原到启用了 I/O 优化存储的数据库集群：

```
aws neptune restore-db-cluster-from-snapshot \
  --db-cluster-identifier (an ID for the restored cluster) \
  --snapshot-identifier (the ID of the snapshot to restore from) \
  --engine neptune \
  --engine-version 1.3.0.0 \
  --storage-type iopt1
```

您可以通过任何 `describe-` 调用来确定集群是否在使用 I/O 优化存储。如果启用了 I/O 优化存储，则该调用将返回设置为 `iop1` 的存储类型字段。

创建新的 Neptune 数据库集群

创建新 Amazon Neptune 数据库集群的最简单方法是使用可以为您创建所有必需资源的 AWS CloudFormation 模板，而不必手动完成所有操作。该 AWS CloudFormation 模板为您执行大部分设置，包括创建亚马逊弹性计算云 (Amazon EC2) 实例：

使用模板启动新的 Neptune 数据库集群 AWS CloudFormation

1. 创建一个拥有使用 Neptune 数据库集群所需权限的新 IAM 用户，如IAM 用户权限中所述。
2. 设置使用 AWS CloudFormation 模板所需的其他先决条件，如中所述用于设置 Nep AWS CloudFormation tune 的先决条件。
3. 调用 AWS CloudFormation 堆栈，如中所述使用 AWS CloudFormation 堆栈创建 Neptune 数据库集群。

您还可以创建跨多个的 Neptune 全局数据库 AWS 区域，从而实现低延迟的全局读取，并在极少数中断影响整个数据库的情况下提供快速恢复。 AWS 区域

有关使用手动创建 Amazon Neptune 集群的信息 AWS Management Console，请参阅。使用 AWS Management Console启动 Neptune 数据库集群

您也可以使用 AWS CloudFormation 模板创建 Lambda 函数以与 Neptune 配合使用（参见）。使用 AWS CloudFormation 创建要在 Neptune 中使用的 Lambda 函数

有关管理 Neptune 中的集群和实例的一般信息，请参阅管理您的 Amazon Neptune 数据库。

用于设置 Nep AWS CloudFormation tune 的先决条件

在使用 AWS CloudFormation 模板创建 Amazon Neptune 集群之前，您需要具备以下条件：

- Amazon EC2 密钥对。
- 使用所需的权限 AWS CloudFormation。

使用创建用于启动 Neptune 集群的 Amazon EC2 密钥对 AWS CloudFormation

要使用 AWS CloudFormation 模板启动 Neptune 数据库集群，您必须在创建堆栈的区域中提供 Amazon EC2Key 对（及其关联的 PEM 文件）。 AWS CloudFormation

如果您需要创建密钥对，请参阅 Amazon EC2 用户指南中的使用 Amazon EC2 创建密钥对，或参阅 Amazon EC2 用户指南中的使用 Amazon EC2 创建密钥对以了解相关说明。

添加 IAM 策略以授予使用 AWS CloudFormation 模板所需的权限

首先，您需要设置一个拥有使用 Neptune 所需权限的 IAM 用户，如创建具有 Neptune 权限的 IAM 用户中所述。

然后，您需要向该用户添加 AWS 托管策略。`AWSCloudFormationReadOnlyAccess`

最后，您需要创建以下客户管理型策略并将其添加到该用户：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
          "rds:CreateDBCluster",
          "rds:CreateDBInstance"
      ],
      "Resource": [
          "arn:aws:rds:*:*:*"
      ],
      "Condition": {
          "StringEquals": {
              "rds:DatabaseEngine": ["graphdb","neptune"]
          }
      }
    },
```

```
    {
      "Action": [
        "rds:AddRoleToDBCluster",
        "rds:AddSourceIdentifierToSubscription",
        "rds:AddTagsToResource",
        "rds:ApplyPendingMaintenanceAction",
        "rds:CopyDBClusterParameterGroup",
        "rds:CopyDBClusterSnapshot",
        "rds:CopyDBParameterGroup",
        "rds:CreateDBClusterParameterGroup",
        "rds:CreateDBClusterSnapshot",
        "rds:CreateDBParameterGroup",
        "rds:CreateDBSubnetGroup",
        "rds:CreateEventSubscription",
        "rds:DeleteDBCluster",
        "rds:DeleteDBClusterParameterGroup",
        "rds:DeleteDBClusterSnapshot",
        "rds:DeleteDBInstance",
        "rds:DeleteDBParameterGroup",
        "rds:DeleteDBSubnetGroup",
        "rds:DeleteEventSubscription",
        "rds:DescribeAccountAttributes",
        "rds:DescribeCertificates",
        "rds:DescribeDBClusterParameterGroups",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBClusterSnapshotAttributes",
        "rds:DescribeDBClusterSnapshots",
        "rds:DescribeDBClusters",
        "rds:DescribeDBEngineVersions",
        "rds:DescribeDBInstances",
        "rds:DescribeDBLogFiles",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBParameters",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSubnetGroups",
        "rds:DescribeEngineDefaultClusterParameters",
        "rds:DescribeEngineDefaultParameters",
        "rds:DescribeEventCategories",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeEvents",
        "rds:DescribeOptionGroups",
        "rds:DescribeOrderableDBInstanceOptions",
        "rds:DescribePendingMaintenanceActions",
        "rds:DescribeValidDBInstanceModifications",
```

```
        "rds:DownloadDBLogFilePortion",
        "rds:FailoverDBCluster",
        "rds:ListTagsForResource",
        "rds:ModifyDBCluster",
        "rds:ModifyDBClusterParameterGroup",
        "rds:ModifyDBClusterSnapshotAttribute",
        "rds:ModifyDBInstance",
        "rds:ModifyDBParameterGroup",
        "rds:ModifyDBSubnetGroup",
        "rds:ModifyEventSubscription",
        "rds:PromoteReadReplicaDBCluster",
        "rds:RebootDBInstance",
        "rds:RemoveRoleFromDBCluster",
        "rds:RemoveSourceIdentifierFromSubscription",
        "rds:RemoveTagsFromResource",
        "rds:ResetDBClusterParameterGroup",
        "rds:ResetDBParameterGroup",
        "rds:RestoreDBClusterFromSnapshot",
        "rds:RestoreDBClusterToPointInTime"
      ],
      "Effect": "Allow",
      "Resource": [
          "*"
      ]
    },
    {
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcs",
        "kms:ListAliases",
        "kms:ListKeyPolicies",
        "kms:ListKeys",
        "kms:ListRetirableGrants",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "sns:ListSubscriptions",
        "sns:ListTopics",
        "sns:Publish"
```

```
      ],
      "Effect": "Allow",
      "Resource": [
          "*"
      ]
    },
    {
      "Action": "iam:PassRole",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
            "iam:passedToService": "rds.amazonaws.com"
        }
      }
    },
    {
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "rds.amazonaws.com"
        }
      }
    }
  ]
}
```

> **Note**
>
> 以下权限仅在删除堆栈时需要：iam:DeleteRole、iam:RemoveRoleFromInstanceProfile、iam:DeleteRolePolicy、i
> 和 ec2:DeleteVpcEndpoints。
> 另请注意，ec2:*Vpc 授予 ec2:DeleteVpc 权限。

使用 AWS CloudFormation 堆栈创建 Neptune 数据库集群

您可以使用 AWS CloudFormation 模板来设置 Neptune 数据库集群。

1. 要在 AWS CloudFormation 主机上启动 AWS CloudFormation 堆栈，请选择下表中的 “启动堆栈” 按钮之一。

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	
美国东部（俄亥俄州）	视图	在 Designer 中查看	
美国西部（加利福尼亚北部）	视图	在 Designer 中查看	
美国西部（俄勒冈州）	视图	在 Designer 中查看	
加拿大（中部）	视图	在 Designer 中查看	
南美洲（圣保罗）	视图	在 Designer 中查看	
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	
欧洲地区（爱尔兰）	视图	在 Designer 中查看	
欧洲地区（伦敦）	视图	在 Designer 中查看	
欧洲地区（巴黎）	视图	在 Designer 中查看	
欧洲地区（法兰克福）	视图	在 Designer 中查看	

区域	查看	在 Designer 中查看	发布
中东（巴林）	视图	在 Designer 中查看	Launch Stack
中东（阿联酋）	视图	在 Designer 中查看	Launch Stack
以色列（特拉维夫）	视图	在 Designer 中查看	Launch Stack
非洲（开普敦）	视图	在 Designer 中查看	Launch Stack
亚太地区（香港）	视图	在 Designer 中查看	Launch Stack
Asia Pacific (Tokyo)	视图	在 Designer 中查看	Launch Stack
亚太地区（首尔）	视图	在 Designer 中查看	Launch Stack
亚太地区（新加坡）	视图	在 Designer 中查看	Launch Stack
亚太地区（悉尼）	视图	在 Designer 中查看	Launch Stack
亚太地区（孟买）	视图	在 Designer 中查看	Launch Stack
中国（北京）	视图	在 Designer 中查看	Launch Stack
中国（宁夏）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国西部）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国东部）	视图	在 Designer 中查看	Launch Stack

2. 在 Select Template 页面上，选择 Next。

3. 在“指定细节”页面上，为 EC2SSH KeyPair 名称选择密钥对。

   要访问 EC2 实例，此密钥对是必需的。确保您具有所选密钥对的 PEM 文件。

4. 选择下一步。
5. 在选项页面上，选择下一步。
6. 在审核页面上，选中第一个复选框以确认 AWS CloudFormation 将创建 IAM 资源。选中第二个复选框以确认新堆栈的 `CAPABILITY_AUTO_EXPAND`。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` 明确确认在创建堆栈时将扩展宏，而无需事先审核。用户通常通过处理的模板创建更改集，以便在实际创建堆栈之前对宏所做的更改进行审核。有关更多信息，请参阅 AWS CloudFormation CreateStack API。

   然后选择创建。

> **Note**
>
> 您也可以使用 AWS CloudFormation 模板升级数据库集群的引擎版本。

设置您的 Amazon Neptune 数据库集群所在的 Amazon VPC

Amazon Neptune 数据库集群只能 在 Amazon Virtual Private Cloud (Amazon VPC) 中创建。其端点可在该 VPC 内访问。

根据您想要访问数据库集群的方式，您可以通过多种不同的方式来设置 VPC。

在配置 Neptune 数据库集群所在的 VPC 时，请记住以下几点：

- 您的 VPC 必须至少有两个子网。这些子网必须位于两个不同的可用区 (AZ) 中。通过跨至少两个可用区分配您的集群实例，Neptune 有助于确保数据库集群中始终有可用的实例，即使在不太可能发生可用区故障的情况下也是如此。Neptune 数据库集群的集群卷始终跨三个可用区以提供持久存储，数据丢失的可能性极低。
- 每个子网中的 CIDR 块必须足够大，以便提供 Neptune 在维护活动、失效转移和扩展期间可能需要的 IP 地址。

- 该 VPC 必须具有一个数据库子网组，其中包含您已创建的子网组。Neptune 选择子网组中的一个子网和该子网中的一个 IP 地址，以与数据库集群中的每个数据库实例关联。然后，数据库实例与子网位于同一个可用区中。
- VPC 应已启用 DNS（包括 DNS 主机名和 DNS 解析）。
- 您的 VPC 必须具有允许访问数据库集群的 VPC 安全组。
- Neptune VPC 中的租赁应设置为默认。

向 Neptune 数据库集群所在的 VPC 添加子网

子网是您的 VPC 内的 IP 地址范围。您可以将 Neptune 数据库集群或 EC2 实例等资源启动到特定的子网中。在创建子网时，指定子网的 IPv4 CIDR 块，它是 VPC CIDR 块的子集。每个子网都必须完全位于一个可用区 (AZ) 内，不能跨越多个可用区。通过在单独的可用区内启动实例，您可以保护您的应用程序不受其中一个可用区中的故障影响。有关更多信息，请参阅 VPC 子网文档。

Neptune 数据库集群需要至少两个 VPC 子网。

将子网添加到 VPC

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择 Subnets(子网)。
3. 在 VPC 控制面板中，选择子网，然后选择创建子网。
4. 在创建子网页面上，选择要在其中创建子网的 VPC。
5. 在子网设置下，进行以下选择：
    a. 在子网名称下输入新子网的名称。
    b. 为子网选择可用区 (AZ)，或者将选择保留为无首选项。
    c. 在 IPv4 CIDR 块下输入子网的 IP 地址块。
    d. 如果需要，可以向子网添加标签。
    e. 选择
6. 如果要同时创建另一个子网，请选择添加新子网。
7. 选择创建子网以创建新的子网。

在 VPC 中创建子网组

创建子网组。

创建 Neptune 子网组

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 选择子网组，然后选择创建数据库子网组。
3. 为新子网组输入名称和描述（描述为必需项）。
4. 在 VPC 下，选择您希望此子网组所在的 VPC。
5. 在可用区下，选择您希望此子网组所在的可用区。
6. 在子网下，将此可用区中的一个或多个子网添加到该子网组。
7. 选择创建以创建新的子网组。

使用 VPC 控制台创建安全组

安全组提供访问 VPC 中的 Neptune 数据库集群的权限。它们充当关联数据库集群的防火墙，在实例级别控制入站和出站流量。默认情况下，创建数据库实例时具有防火墙和默认安全组，用来阻止对其进行任何访问。要启用访问权限，您必须拥有带其它规则的 VPC 安全组。

以下过程显示了如何添加自定义 TCP 规则，该规则指定 Amazon EC2 实例用于访问 Neptune 数据库集群的端口范围和 IP 地址。您可以使用分配到 EC2 实例而不是其 IP 地址的 VPC 安全组。

在控制台上为 Neptune 创建 VPC 安全组

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在控制台的右上角，选择要在其中为 Neptune 创建 VPC 安全组的 AWS 区域。该区域的 Amazon VPC 资源列表应显示至少有一个 VPC 和多个子网。如果没有显示，则说明您在该区域中没有默认 VPC。
3. 在导航窗格中的安全下，选择安全组。
4. 选择创建安全组。在创建安全组窗口中，输入安全组名称、描述和您的 Neptune 数据库集群所在 VPC 的标识符。
5. 为要连接到 Neptune 数据库集群的 Amazon EC2 实例的安全组添加入站规则：

a. 在入站规则区域中，选择添加规则。
b. 在类型列表中，将自定义 TCP 保留为选中状态。
c. 在端口范围框中，输入 Neptune 的默认端口值 8182。
d. 在源下，输入您要从中访问 Neptune 的 IP 地址范围（CIDR 值），或者选择现有安全组名称。
e. 如果需要添加更多 IP 地址或不同端口范围，请再次选择添加规则。

6. 如果需要，您还可以在“出站规则”区域中添加一条或多条出站规则。
7. 完成后，选择 Create security group (创建安全组)。

在创建新的 Neptune 数据库集群时，您可以使用这个新的 VPC 安全组。

如果您使用默认 VPC，则已为您创建跨越该 VPC 的所有子网的默认子网组。在 Neptune 控制台中选择创建数据库时，除非您指定其它 VPC，否则将使用默认 VPC。

请确保您的 VPC 中具有 DNS 支持

域名系统 (DNS) 是 Internet 中名称使用的标准，以将名称解析到各自相应的 IP 地址。DNS 主机名称可以唯一的命名计算机，它由主机名称和域名组成。DNS 服务器会将 DNS 主机名称解析到其相应的 IP 地址。

检查以确保在您的 VPC 中同时启用了 DNS 主机名和 DNS 解析。VPC 网络属性 `enableDnsHostnames` 和 `enableDnsSupport` 必须设置为 `true`。要查看和修改这些属性，请转到 VPC 控制台：https://console.aws.amazon.com/vpc/ 的 。

有关更多信息，请参阅将 DNS 与您的 VPC 一起使用。

> **Note**
>
> 如果您使用的是 Route 53，请确认您的配置不会覆盖您的 VPC 中的 DNS 网络属性。

连接到您的 Amazon Neptune 图形

一旦您创建了 Neptune 数据库集群，下一步就是设置您想要连接到它的方式。

设置 curl 或 awscurl 以与您的 Neptune 端点进行通信

如本文档中的许多示例所示，拥有用于向 Neptune 数据库集群提交查询的命令行工具非常方便。未启用 IAM 身份验证时，curl 命令行工具是与 Neptune 端点通信的上佳选择。从 7.75.0 开始的版本支持 `--aws-sigv4` 选项，用于在启用 IAM 身份验证时对请求进行签名。

对于启用了 IAM 身份验证的端点，也可以使用 awscurl，它使用的语法与 `curl` 几乎完全相同，但支持按 IAM 身份验证的要求对请求进行签名。由于 IAM 身份验证提供了额外的安全性，因此通常最好启用它。

有关如何使用 `curl`（或 `awscurl`）的信息，请参阅 curl 手册页面和电子书 Everything curl。

要使用 HTTPS 进行连接（Neptune 需要），`curl` 需要访问相应的证书。只要 `curl` 可以找到相应的证书，它即可像处理 HTTP 连接一样处理 HTTPS 连接，而无需额外的参数。对于 `awscurl` 也是如此。本文档中的示例基于该场景。

要了解如何获取此类证书以及如何将其正确格式化为 `curl` 可以使用的证书颁发机构 (CA) 证书存储，请参阅 `curl` 文档中的 SSL 证书验证。

然后，您使用 `CURL_CA_BUNDLE` 环境变量指定此 CA 证书存储的位置。在 Windows 上，`curl` 自动在名为 `curl-ca-bundle.crt` 的文件中查找它。首先在与 `curl.exe` 相同的目录中查找，然后在路径的其他位置查找。有关更多信息，请参阅 SSL 证书验证。

连接到 Neptune 数据库集群的不同方法

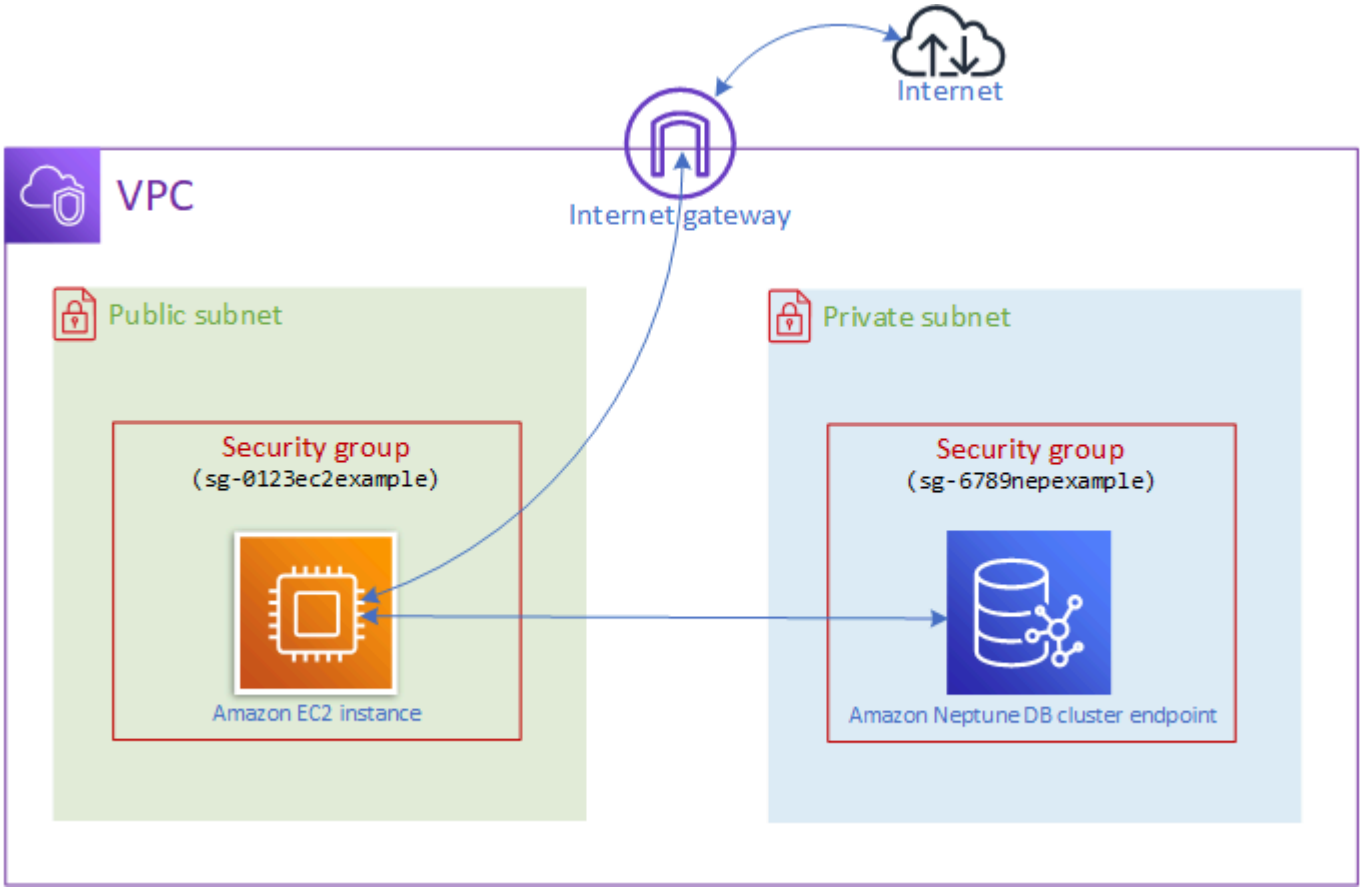
Amazon Neptune 数据库集群只能 在 Amazon Virtual Private Cloud (Amazon VPC) 中创建。除非您为数据库集群启用和设置 Neptune 公共端点，否则只能在该 VPC 内访问其端点。

有以下几种不同的方法可以设置对 VPC 中 Neptune 数据库集群的访问：

- 从同一 VPC 中的 Amazon EC2 实例进行连接
- 从另一个 VPC 中的 Amazon EC2 实例进行连接
- 从私有网络连接

从同一 VPC 中的 Amazon EC2 实例连接到 Neptune 数据库集群

连接到 Neptune 数据库的最常见方法之一是从与 Neptune 数据库集群位于同一 VPC 中的 Amazon EC2 实例进行连接。例如，EC2 实例可能正在运行与互联网连接的 Web 服务器。在这种情况下，只有 EC2 实例可以访问 Neptune 数据库集群，而互联网只能访问 EC2 实例：



要启用此配置，您需要设置正确的 VPC 安全组和子网组。Web 服务器托管在公有子网中以便它可访问公共互联网，而您的 Neptune 集群实例托管在私有子网中以确保其安全。请参阅 设置您的 Amazon Neptune 数据库集群所在的 Amazon VPC。

为了使 Amazon EC2 实例连接到（例如，端口 8182）上的 Neptune 端点，您需要设置一个安全组来执行此操作。如果您的 Amazon EC2 实例使用的是安全组（例如，名为 `ec2-sg1`），则需要创建另一个 Amazon EC2 安全组（比方说 `db-sg1`），此安全组具有用于端口 8182 的入站规则并使用 `ec2-sg1` 作为其源。然后，将 `db-sg1` 添加到您的 Neptune 集群以允许连接。

创建 Amazon EC2 实例后，您可以使用 SSH 登录该实例并连接到您的 Neptune 数据库集群。有关使用 SSH 连接到 EC2 实例的信息，请参阅 A mazon EC2 用户指南中的连接到您的 Linux 实例。

如果您使用 Linux 或 macOS 命令行连接到 EC2 实例，则可以将来自 ss hAcces s 项目的 SSH 命令粘贴到堆栈的输出部分。 AWS CloudFormation 您在当前目录中必须具有 PEM 文件并且 PEM 文件权限必须设置为 400 (`chmod 400 keypair.pem`)。

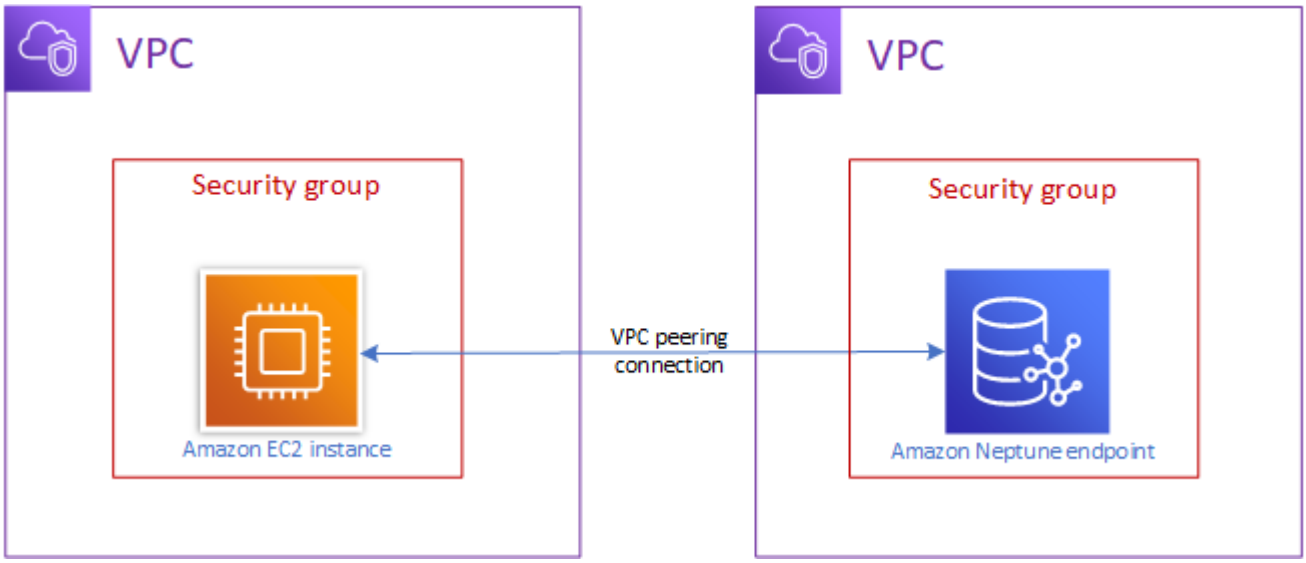
创建包含公有子网和私有子网的 VPC

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在的右上角 AWS Management Console，选择要在其中创建 VPC 的区域。
3. 在 VPC 控制面板上，选择启动 VPC 向导。
4. 完成创建 VPC 页面的 VPC 设置区域：
   a. 在 Resources to create（要创建的资源）下，选择 VPC, subnets, etc.（VPC、子网等）。
   b. 保留默认名称标签不变，或输入您选择的名称，或者取消选中自动生成复选框以禁用名称标签生成。
   c. 将 IPv4 CIDR 块值保留为 `10.0.0.0/16`。
   d. 将 IPv6 CIDR 块值保留为无 IPv6 CIDR 块。
   e. 将租赁保留为默认值。
   f. 将可用区 (AZ) 的数量保留为 2。
   g. 除非需要一个或多个 NAT 网关，否则将 NAT 网关 ($) 设置为无。
   h. 除非您将使用 Amazon S3，否则将 VPC 端点设置为无。
   i. 启用 DNS 主机名和启用 DNS 解析应已选中。
5. 选择创建 VPC。

从另一个 VPC 中的 Amazon EC2 实例访问数据库集群

Amazon Neptune 数据库集群只能在 Amazon Virtual Private Cloud (Amazon VPC) 中创建，并且其端点只能在该 VPC 内访问，通常是从在该 VPC 中运行的 Amazon Elastic Compute Cloud (Amazon EC2) 实例进行访问。

当您的数据库集群与您用来访问它的 EC2 实例位于不同的 VPC 中时，您可以使用 VPC 对等连接进行连接：



VPC 对等连接是指两个 VPC 之间的网络连接，用于在它们之间以私有方式路由流量，这样，任何一个 VPC 中的实例都可以像在同一网络中一样进行通信。您可以在账户中的 VPC 之间、您账户中的 VPC 与其他账户中的 VPC 之间，或者与其他区域的 VPC 之间创建 VPC 对等连接。 AWS AWS AWS

AWS 使用 VPC 的现有基础设施创建 VPC 对等连接。它既不是网关，也不是 AWS 站点到站点 VPN 连接，也不依赖单独的物理硬件。它没有单点通信故障，也没有带宽瓶颈。

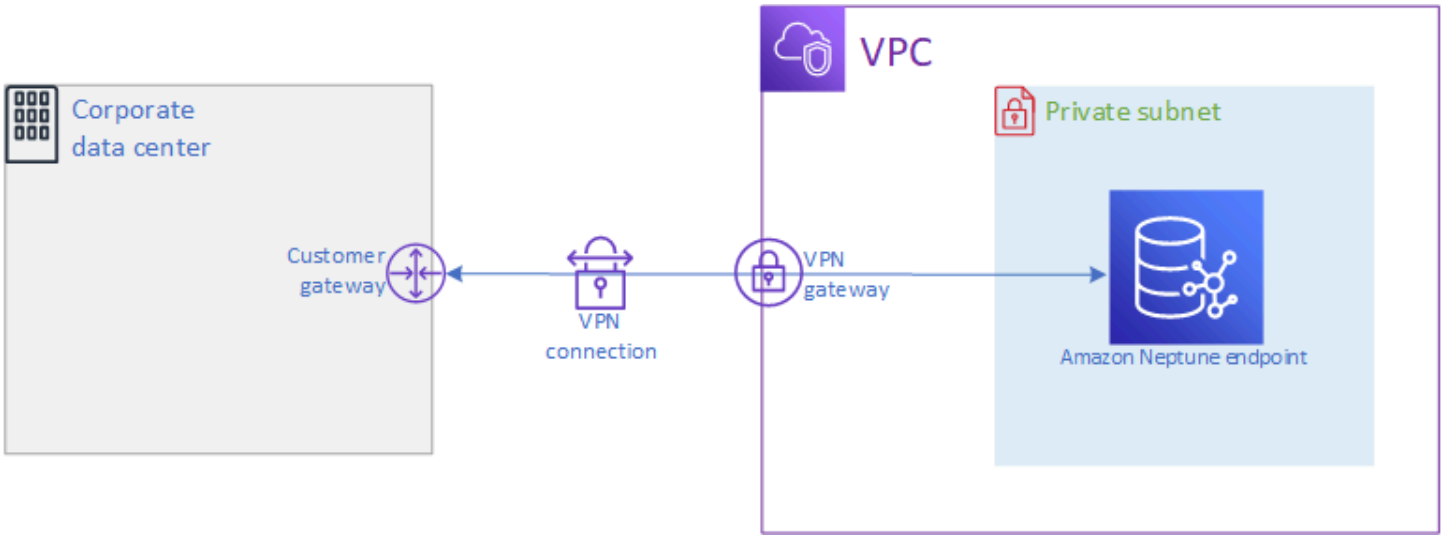
有关如何使用 VPC 对等连接的更多信息，请参阅 Amazon VPC 对等连接指南。

从私有网络访问您的数据库集群

您可以通过两种不同的方式从私有网络访问 Neptune 数据库集群：

- 使用 AWS Site-to-Site VPN 连接。
- 使用 AWS Direct Connect 连接。

上面的链接包含有关这些连接方法以及如何设置它们的信息。 AWS 站点到站点连接的配置可能如下所示：



在 Amazon Neptune 中保护数据

有多种方式可保护您的 Amazon Neptune 集群。

使用 IAM policy 限制对 Neptune 数据库集群的访问

要控制谁可以对 Neptune 数据库集群和数据库实例执行 Neptune 管理操作，请使用 AWS Identity and Access Management (IAM)。

当你使用 IAM 账户访问 Neptune 控制台时，必须先 AWS Management Console 使用你的 IAM 账户登录，然后才能通过 https://console.aws.amazon.com/neptune/home 打开 Neptune 控制台。

当您 AWS 使用 IAM 凭证进行连接时，您的 IAM 账户必须拥有 IAM 策略，这些策略可授予执行 Neptune 管理操作所需的权限。有关更多信息，请参阅 使用不同类型的 IAM policy 控制对 Neptune 的访问权限。

使用 VPC 安全组限制对 Neptune 数据库集群的访问

Neptune 数据库集群必须在 Amazon Virtual Private Cloud (Amazon VPC) 中创建。要控制哪些设备和 EC2 实例能够建立与 VPC 中 Neptune 数据库集群的数据库实例端点和端口的连接，请使用 VPC 安全组。有关 VPCs 的更多信息，请参阅 使用 VPC 控制台创建安全组。

使用 IAM 身份验证限制对 Neptune 数据库集群的访问

如果您在 Neptune 数据库集群中启用 AWS Identity and Access Management (IAM) 身份验证，则必须先对访问该数据库集群的任何人进行身份验证。有关设置 IAM 身份验证的信息，请参阅亚马逊 Neptune 中的 AWS Identity and Access Management (IAM) 概述。

有关使用临时证书进行身份验证的信息，包括、和 Amazon EC2 的示例，请参阅the section called “临时证书”。 AWS CLI AWS Lambda

以下链接提供了有关将 IAM 身份验证和各种查询语言结合使用来连接到 Neptune 的更多信息：

将 Gremlin 和 IAM 身份验证结合使用

- the section called “Gremlin 控制台”
- the section called “Gremlin Java”
- the section called “Python 示例”

> **Note**
>
> 本示例适用于 Gremlin 和 SPARQL。

将 openCypher 和 IAM 身份验证结合使用

- the section called “Gremlin 控制台”
- the section called “Gremlin Java”
- the section called “Python 示例”

> **Note**
>
> 本示例适用于 Gremlin 和 SPARQL。

将 SPARQL 和 IAM 身份验证结合使用

- the section called “SPARQL Java（RDF4J 和 Jena）”
- the section called “Python 示例”

> **Note**
>
> 本示例适用于 Gremlin 和 SPARQL。

访问 Neptune 图形入门

创建 Neptune 数据库集群并与其建立连接后，下一步就是与其通信，以便加载数据、进行查询等。为此，大多数人使用 `curl` 或 `awscurl` 命令行工具。

设置 `curl` 以与您的 Neptune 端点进行通信

如本文档中的多个示例所示，curl 命令行工具是用于与您的 Neptune 端点进行通信的方便选项。有关该工具的信息，请参阅 curl 手册页面和电子书 Everything curl。

要使用 HTTPS 进行连接（正如我们所建议以及 Neptune 在大多数区域中所要求的），`curl` 需要对相应证书拥有访问权限。要了解如何获取这些证书以及如何将其正确格式化为 `curl` 可以使用的证书颁发机构 (CA) 证书存储，请参阅 `curl` 文档中的 SSL 证书验证。

然后，您使用 `CURL_CA_BUNDLE` 环境变量指定此 CA 证书存储的位置。在 Windows 上，`curl` 自动在名为 `curl-ca-bundle.crt` 的文件中查找它。首先在与 `curl.exe` 相同的目录中查找，然后在路径的其他位置查找。有关更多信息，请参阅 SSL 证书验证。

只要 `curl` 可以找到相应的证书，它即可像处理 HTTP 连接一样处理 HTTPS 连接，而无需额外的参数。本文档中的示例基于该场景。

使用查询语言访问 Neptune 数据库集群中的图形数据

连接后，您可以使用 Gremlin 和 openCypher 查询语言来创建和查询属性图，或者使用 SPARQL 查询语言来创建和查询包含 RDF 数据的图形。

Neptune 支持的图形查询语言

- Gremlin 是用于属性图的图形遍历语言。Gremlin 中的查询是由离散步骤组成的遍历，每个步骤都沿着一个边缘到达一个节点。有关更多信息，请参阅 Apache TinkerPop 3 上的 Gremlin 文档。

  Gremlin 的 Neptune 实施与其它实施有一些差别，尤其是在您使用 Gremlin-Groovy 时（作为序列化文本发送的 Gremlin 查询）。有关更多信息，请参阅 Amazon Neptune 中的 Gremlin 标准合规性。
- openCypher 是一种用于属性图的声明式查询语言，最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。其语法在 Cypher 查询语言参考版本 9 中介绍。
- SPARQL 是一种用于 RDF 数据的声明性查询语言，基于由万维网联盟 (W3C) 标准化的图形模式匹配，并在 SPARQL 1.1 概述和 SPARQL 1.1 查询语言规范中描述。

> **Note**
>
> 可以同时使用 Gremlin 和 openCypher 在 Neptune 中访问属性图数据，而不使用 SPARQL。同样，您只能使用 SPARQL 访问 RDF 数据，而不能使用 Gremlin 或 openCypher。

使用 Gremlin 访问 Amazon Neptune 中的图形

你可以使用 Gremlin 控制台在 REPL（read-eval-print 循环）环境中尝试 TinkerPop 图形和查询。

以下教程为您演练使用 Gremlin 控制台将顶点、边缘、属性以及更多内容添加到 Neptune 图形的过程，重点讲述 Neptune 特定的 Gremlin 实施的不同之处。

> **Note**
>
> 此示例假定您已完成以下内容：
>
> - 您已使用 SSH 连接到 Amazon EC2 实例。
> - 您已创建了一个 Neptune 集群，如创建数据库集群中所述。
> - 您已安装了 Gremlin 控制台，如 安装 Gremlin 控制台 中所述。

使用 Gremlin 控制台

1. 将目录更改为解压缩 Gremlin 控制台文件所在的文件夹。

   ```
   cd apache-tinkerpop-gremlin-console-3.6.5
   ```

2. 输入以下命令以运行 Gremlin 控制台。

   ```
   bin/gremlin.sh
   ```

   您应看到以下输出：

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

```
gremlin>
```

您现在位于 gremlin> 提示符处。您在此提示符处输入剩余步骤。

3. 在 gremlin> 提示符处，输入以下命令以连接到 Neptune 数据库实例。

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

4. 在 gremlin> 提示符处，输入以下命令以切换到远程模式。这会将所有 Gremlin 查询发送到远程连接。

```
:remote console
```

5. 添加带有标签和属性的顶点。

```
g.addV('person').property('name', 'justin')
```

顶点分配有包含 GUID 的 string ID。Neptune 中的所有顶点 ID 均为字符串。

6. 添加具有自定义 ID 的顶点。

```
g.addV('person').property(id, '1').property('name', 'martin')
```

id 属性未使用引号引起来。这是顶点 ID 的关键字。顶点 ID 具有一个字符串，其中包含数字 1。

普通属性名称必须包含在引号中。

7. 更改属性或添加属性（如果属性不存在）。

```
g.V('1').property(single, 'name', 'marko')
```

此处介绍如何更改上一步中顶点的 name 属性。这会从 name 属性删除现有值。

如果您未指定 single，则会改为将值附加到 name 属性（如果尚未附加）。

8. 添加属性，但在属性已有值时附加属性。

```
g.V('1').property('age', 29)
```

Neptune 使用集基数作为默认操作。

此命令添加值为 29 的 age 属性，但不替换任何现有值。

如果 age 属性已有值，此命令将 29 附加到属性。例如，如果 age 属性是 27，则新值将为 `[ 27, 29 ]`。

9. 添加多个顶点。

```
g.addV('person').property(id, '2').property('name', 'vadas').property('age',
 27).iterate()
g.addV('software').property(id, '3').property('name', 'lop').property('lang',
 'java').iterate()
g.addV('person').property(id, '4').property('name', 'josh').property('age',
 32).iterate()
g.addV('software').property(id, '5').property('name', 'ripple').property('lang',
 'java').iterate()
g.addV('person').property(id, '6').property('name', 'peter').property('age', 35)
```

您可以同时将多个语句添加到 Neptune。

语句可以使用换行符 (`'\n'`)、空格 (`' '`)、分号 (`'; '`) 或什么都不使用（例如，`g.addV('person').iterate()g.V()` 是有效的）来分隔。

> **Note**
>
> Gremlin 控制台对每个换行符 (`'\n'`) 发送单独的命令，因此在该例中，它们每一个都是一项单独的事务。此示例的每个命令位于单独一行中，以便阅读。删除换行符 (`'\n'`) 可以将其通过 Gremlin 控制台作为单个命令发送。

除了最后一条语句之外的所有语句必须以终止步骤结尾，例如 `.next()` 或 `.iterate()`，否则语句不会运行。Gremlin 控制台不需要这些终止步骤。在不需要对结果进行序列化时使用 `.iterate`。

一起发送的所有语句包括在单个事务处理中，并且一起成功或失败。

10. 添加边缘。

```
g.V('1').addE('knows').to(__.V('2')).property('weight', 0.5).iterate()
g.addE('knows').from(__.V('1')).to(__.V('4')).property('weight', 1.0)
```

这里介绍了两种不同的方法来添加边缘。

11. 添加现代图形的其余部分。

```
g.V('1').addE('created').to(__.V('3')).property('weight', 0.4).iterate()
g.V('4').addE('created').to(__.V('5')).property('weight', 1.0).iterate()
g.V('4').addE('knows').to(__.V('3')).property('weight', 0.4).iterate()
g.V('6').addE('created').to(__.V('3')).property('weight', 0.2)
```

12. 删除顶点。

```
g.V().has('name', 'justin').drop()
```

删除 name 属性等于 justin 的顶点。

> **Important**
>
> 到此为止，你就有了完整的 Apache M TinkerPop odern 图表。 TinkerPop 文档遍历部分中的示例使用现代图。

13. 运行遍历。

```
g.V().hasLabel('person')
```

返回所有 person 顶点。

14. 使用值运行遍历 (valueMap())。

```
g.V().has('name', 'marko').out('knows').valueMap()
```

返回 marko“知道”的所有顶点的键值对。

15. 指定多个标签。

```
g.addV("Label1::Label2::Label3")
```

Neptune 对于一个顶点支持多个标签。创建标签时，您可以指定多个标签，同时使用 :: 分隔它们。

此示例添加具有三个不同标签的顶点。

hasLabel 步骤与具有以下任一三个标签的此顶点相匹配：hasLabel("Label1")、hasLabel("Label2") 和 hasLabel("Label3")。

`::` 分隔符仅用于此用途。

您不能在 `hasLabel` 步骤中指定多个标签。例如，`hasLabel("Label1::Label2")` 与任何内容都不匹配。

16. 指定时间/日期。

```
g.V().property(single, 'lastUpdate', datetime('2018-01-01T00:00:00'))
```

Neptune 不支持 Java 日期。改用 `datetime()` 函数。`datetime()` 接受符合 ISO8061 的 `datetime` 字符串。

支持以下格式：`YYYY-MM-DD`, `YYYY-MM-DDTHH:mm`、`YYYY-MM-DDTHH:mm:SS` 和 `YYYY-MM-DDTHH:mm:SSZ`。

17. 删除顶点、属性或边缘。

```
g.V().hasLabel('person').properties('age').drop().iterate()
g.V('1').drop().iterate()
g.V().outE().hasLabel('created').drop()
```

下面是几个删除示例。

> **Note**
>
> `.next()` 步骤不可与 `.drop()` 一起使用。请改用 `.iterate()`。

18. 完成后，输入以下命令以退出 Gremlin 控制台。

```
:exit
```

> **Note**
>
> 使用分号 (`;`) 或换行符 (\n) 分隔每个语句。
> 在最后遍历之前的每个遍历必须以要执行的 `iterate()` 结尾。仅返回最后遍历中的数据。

使用 openCypher 访问 Amazon Neptune 中的图形

要开始使用 OpenCypher，请查看或使用 Nept openCypher une 图形笔记本存储库中的 OpenCypher 笔记本。 GitHub

使用 RDF 和 SPARQL 访问 Amazon Neptune 中的图形

SPARQL 是一种用于资源描述框架 (RDF) 的查询语言，这是一种专为 Web 设计的图形数据格式。Amazon Neptune 与 SPARQL 1.1 兼容。这表示您可以连接到 Neptune 数据库实例并使用 SPARQL 1.1 查询语言规范中所述的查询语言查询图形。

SPARQL 中的查询包含一个用于指定要返回的变量的 `SELECT` 子句和一个用于指定要在图形中匹配的数据的 `WHERE` 子句。如果您不熟悉 SPARQL 查询，请参阅 SPARQL 1.1 查询语言中的编写简单查询。

针对 Neptune 数据库实例的 SPARQL 查询的 HTTP 端点为 `https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`。

连接到 SPARQL

1. 你可以从堆栈输出部分的项目中获取 Neptune 集群SparqlEndpoint的 SPARQL 终端节点。 AWS CloudFormation
2. 输入以下内容以使用 HTTP POST 和 curl 命令提交 SPARQL **UPDATE**。

   ```
   curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://
   test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
   ```

   上述示例将以下三元组插入 SPARQL 默认图形中：`<https://test.com/s> <https://test.com/p> <https://test.com/o>`
3. 输入以下内容以使用 HTTP POST 和 curl 命令提交 SPARQL **QUERY**。

   ```
   curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10'
    https://your-neptune-endpoint:port/sparql
   ```

   上述示例使用限制为 10 的 `?s ?p ?o` 查询最多返回图形中的 10 个三元组 (主-谓-宾)。要查询其他内容，请将其替换为其他 SPARQL 查询。

> **Note**
>
> 对于 SELECT 和 ASK 查询，响应的默认 MIME 类型为 application/sparql-results+json。
> 对于 CONSTRUCT 和 DESCRIBE 查询，响应的默认 MIME 类型为 application/n-quads。
> 有关所有可用的 MIME 类型的列表，请参阅 SPARQL HTTP API。

将数据加载到 Neptune 中

Amazon Neptune 提供将数据从外部文件直接加载到 Neptune 数据库实例中的过程。可使用此过程代替执行大量 INSERT 语句、addV 和 addE 步骤或其他 API 调用。

下面是指向额外加载信息的链接。

- 加载数据的方法 – 加载数据
- 批量加载程序支持的数据格式 – the section called “数据格式”
- 加载示例 – the section called “加载示例”

监控 Amazon Neptune

Amazon Neptune 支持以下监控方法。

- 亚马逊 CloudWatch — Amazon Neptune 会自动向警报发送指标 CloudWatch 并支持 CloudWatch 警报。有关更多信息，请参阅 the section called “使用 CloudWatch”。
- AWS CloudTrail— 亚马逊 Neptune 支持使用 API 日志记录。 CloudTrail有关更多信息，请参阅 the section called “使用记录 Neptune API 调用 AWS CloudTrail”。
- 标记 – 使用标签向 Neptune 资源添加元数据并基于标签跟踪使用情况。有关更多信息，请参阅 the section called “给 Neptune 资源加标签”。
- 审计日志文件 – 使用 Neptune 控制台来查看、下载或监视数据库日志文件。有关更多信息，请参阅 the section called “使用 Neptune 审计日志”。
- 实例状态 – 检查 Neptune 实例的图形数据库引擎的运行状况，查明安装了引擎的哪个版本，并使用实例状态 API 获取其它引擎状态信息。

Neptune 中的故障排除和最佳实践

以下链接可能有助于解决 Amazon Neptune 的问题。

- 最佳实践 – 有关常见问题解决方案和性能建议，请参阅最佳实践。
- 服务错误 – 有关管理 API 和图形数据库连接的错误列表，请参阅Neptune 错误。
- 服务限制 – 有关 Neptune 限制的信息，请参阅Neptune 限制。
- 引擎版本 – 有关图形引擎版本的信息（包括发行说明），请参阅引擎版本。
- 支持论坛 – 要加入有关 Neptune 的讨论，请参阅 Amazon Neptune 论坛。
- 定价 – 有关使用 Amazon Neptune 的成本的信息，请参阅 Amazon Neptune 定价。
- AWS S@@ u pport — 要获得专家的帮助和指导，请参阅AWS Support。

创建 Neptune 全球数据库

Amazon Neptune 全球数据库跨多个 AWS 区域，从而实现低延迟的全局读取，并在中断影响整个 AWS 区域的极少数情况下提供快速恢复。

一个 Neptune 全球数据库由一个区域中的一个主数据库集群和不同区域中最多有五个辅助数据库集群组成。

只能在主区域进行写入。辅助区域仅支持读取。每个辅助区域最多可以有 16 个读取器实例。

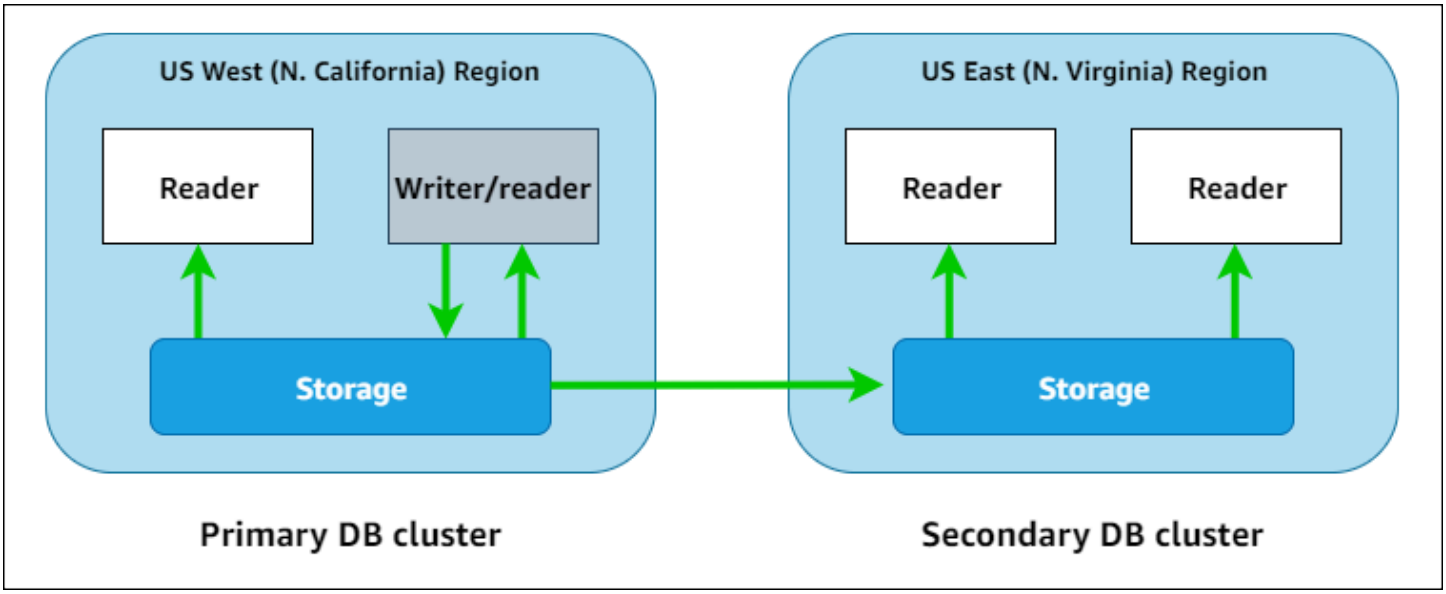
主题

Amazon Neptune 中的全球数据库概览

使用 Neptune 全球数据库，您可以在跨多个 AWS 区域的单个数据库上运行您的全球分布式应用程序。

Neptune 全球数据库由主 AWS 区域中一个用于写入数据的数据库集群和辅助 AWS 区域中最多五个只读数据库集群组成。当您对主数据库集群执行写入操作时，Neptune 会将写入的数据复制到使用专用基础设施的所有辅助数据库集群，延迟通常不到一秒钟。

下图显示了跨两个 AWS 区域的全球数据库示例。



您可以通过添加一个或多个只读副本实例来独立扩展每个辅助集群，以处理只读工作负载。

要执行写入操作，必须连接到主数据库集群的数据库集群端点。只有主集群才能执行写入操作。然后，如上图所示，复制由集群存储卷而不是数据库引擎执行。

Neptune 全球数据库专为遍布全球的应用程序而设计。只读辅助数据库集群支持更靠近应用程序用户的读取操作。

Neptune 全球数据库支持两种不同的失效转移方法：

- 要从主区域的中断中恢复，请使用手动计划外分离和提升流程，即分离一个辅助集群，将其转换为独立集群，然后将其提升为新的主集群。
- 对于计划内操作程序（例如维护），请使用托管式计划内失效转移，将主集群重新定位到其辅助区域之一，而不会造成数据丢失。

在 Amazon Neptune 中使用全球数据库的优势

使用全球数据库，您可以获得以下优势：

- 全球读取，本地延迟 – 如果您在世界各地设有办事处，全球数据库允许您辅助区域的办公室在本地延迟的情况下访问自己所在区域的数据。
- 可扩展辅助 Neptune 数据库集群 — 您可以通过添加只读副本数据库实例来扩展辅助集群。因为辅助集群是只读的，所以每个集群最多可以支持 16 个只读副本，而不是通常的 15 个限制。

- 快速复制到辅助数据库集群 — 从主数据库集群到辅助数据库集群的复制速度很快，延迟通常不到一秒，对主数据库集群的性能影响很小。由于复制是在存储级别执行的，因此数据库实例资源完全可用于应用程序读取和写入工作负载。
- 从区域范围内的中断中恢复 — 与传统复制解决方案相比，辅助数据库集群使您能够更快地将主集群移动到一个新的区域，RTO 更低且数据丢失更少（RPO 更低）。

Amazon Neptune 中全球数据库的限制

以下限制目前适用于 全局数据库：

- Neptune 全球数据库仅在以下 AWS 区域可用：
  - 美国东部（弗吉尼亚州北部）：us-east-1
  - 美国东部（俄亥俄州）：us-east-2
  - 美国西部（北加利福尼亚）：us-west-1
  - 美国西部（俄勒冈州）：us-west-2
  - 欧洲地区（爱尔兰）：eu-west-1
  - 欧洲地区（伦敦）：eu-west-2
  - 亚太地区（东京）：ap-northeast-1
- Neptune 全球数据库不支持自动扩缩辅助数据库集群。
- 在对该全球数据库执行主要版本升级时，无法将自定义参数组应用于全球数据库集群。相反，在全球集群的每个区域中创建自定义参数组，然后在升级后手动将它们应用于区域集群。
- 您无法单独停止或启动全球数据库中的数据库集群。
- 在某些情况下，辅助数据库集群中的只读副本可能会重新启动。如果主集群的写入器实例重新启动或失效转移，则辅助区域中的所有实例也将重新启动。随后辅助集群将不可用，直到其所有实例与主数据库集群的写入器实例恢复同步。

在 Amazon Neptune 中设置全球数据库

您可以通过以下方式之一创建 Neptune 全球数据库：

- 使用所有新的数据库集群和实例创建全球数据库。
- 使用现有 Neptune 数据库集群作为主集群创建全球数据库。

主题

Amazon Neptune 中全球数据库的配置要求

Neptune 全球数据库跨至少两个 AWS 区域。主 AWS 区域包含具有一个写入器实例的 Neptune 数据库集群。一到五个辅助 AWS 区域中的每一个区域都包含一个只读 Neptune 数据库集群，该集群完全由只读副本实例组成。至少需要一个辅助 AWS 区域。

组成全球数据库的 Neptune 数据库集群具有以下特定要求：

- 数据库实例类要求 — 全球数据库需要针对内存密集型工作负载进行优化的 `r5` 或 `r6g` 数据库实例类，例如 `db.r5.large` 实例类型。
- AWS 区域要求 — 全球数据库需要位于一个 AWS 区域中的主 Neptune 数据库集群，以及位于另一个区域中的至少一个辅助 Neptune 数据库集群。您最多可以创建五个辅助只读 Neptune 数据库集群，每个集群必须在不同的区域中。换句话说，Neptune 全球数据库中没有两个 Neptune 数据库集群可位于同一个 AWS 区域中。
- 引擎版本要求 — 全球数据库中的所有数据库集群使用的 Neptune 引擎版本应相同，并且必须大于或等于 `1.2.0.0`。如果您在创建新的全球数据库、集群或实例时未指定引擎版本，则将使用最新的引擎版本。

> **Important**
>
> 尽管可以为全球数据库中的每个数据库集群单独配置数据库集群参数组，但最好在集群之间保持设置的一致性，以避免在必须将辅助集群提升为主集群时出现意外行为变化。例如，对于所有数据库集群中的对象索引、流等使用相同设置。

使用 AWS CLI 在 Amazon Neptune 中创建全球数据库

> **Note**
>
> 本节中的示例遵循使用反斜杠 (\) 作为行扩展符的 UNIX 惯例。对于 Windows，请用尖号 (^) 代替反斜杠。

使用 AWS CLI 创建全球数据库

1. 首先使用 `create-global-cluster` AWS CLI 命令（用于封装 CreateGlobalCluster API）创建一个空的全球数据库。指定要作为主区域的 AWS 区域的名称，将 Neptune 设置为数据库引擎，也可选择指定要使用的引擎版本（必须为 1.2.0.0 或更高版本）：

```
aws neptune create-global-cluster
  --region (primary region, such as us-east-1) \
  --global-cluster-identifier (ID for the global database) \
  --engine neptune \
  --engine-version (engine version; this is optional)
```

2. 全球数据库可能需要几分钟才能可用，因此在转入下一步之前，请使用 `describe-global-clusters` CLI 命令（用于封装 DescribeGlobalClusters API）检查全球数据库是否可用：

```
aws neptune describe-global-clusters \
  --region (primary region) \
  --global-cluster-identifier (global database ID)
```

3. 在 Neptune 全球数据库变为可用后，您可以创建一个新的 Neptune 数据库集群作为其主集群：

```
aws neptune create-db-cluster \
  --region (primary region) \
  --db-cluster-identifier (ID for the primary DB cluster) \
  --engine neptune \
  --engine-version (engine version; must be >= 1.2.0.0) \
  --global-cluster-identifier (global database ID)
```

4. 使用 `describe-db-clusters` AWS CLI 命令确认新的数据库集群已准备就绪，可以添加其主数据库实例：

```
aws neptune describe-db-clusters \
  --region (primary region) \
```

```
  --db-cluster-identifier (primary DB cluster ID)
```

当响应显示 "Status": "available" 时，继续执行下一步。

5. 使用 create-db-instance AWS CLI 命令为主集群创建主数据库实例。您必须使用内存优化型 r5 或 r6g 实例类型之一，例如 db.r5.large。

```
aws neptune create-db-instance \
  --region (primary region) \
  --db-cluster-identifier (primary cluster ID) \
  --db-instance-class (instance class) \
  --db-instance-identifier (ID for the DB instance) \
  --engine neptune \
  --engine-version (optional: engine version)
```

> **Note**
>
> 如果您计划使用 Neptune 批量加载程序向新的主数据库集群添加数据，请在添加辅助区域之前执行此操作。这比在全球数据库完全设置完毕后执行批量加载更快、更具成本效益。

现在，向新的全球数据库添加一个或多个辅助区域，如使用 AWS CLI 添加辅助区域中所述。

将现有数据库集群转换为全球数据库

要将现有数据库集群转换为全球数据库，请使用 create-global-cluster AWS CLI 命令创建一个与现有数据库集群位于相同 AWS 区域中的新全球数据库，并将其 --source-db-cluster-identifier 参数设置为位于此处的现有集群的 Amazon 资源名称 (ARN)：

```
aws neptune create-global-cluster \
  --region (region where the existing cluster is located) \
  --global-cluster-identifier (provide an ID for the new global database) \
  --source-db-cluster-identifier (the ARN of the existing DB cluster) \
  --engine neptune \
  --engine-version (engine version; this is optional)
```

现在，向新的全球数据库添加一个或多个辅助区域，如使用 AWS CLI 添加辅助区域中所述。

使用从快照还原的数据库集群作为主集群

您可以将从快照还原的数据库集群转换成 Neptune 全球数据库。完成还原后，将它创建的数据库集群转换为新的全球数据库的主集群，如上所述。

在 Amazon Neptune 中向主区域添加辅助全球数据库区域

Neptune 全球数据库至少需要一个与主数据库集群不在相同 AWS 区域中的辅助 Neptune 数据库集群。您最多可以将五个辅助数据库集群附加到主数据库集群。

您添加的每个辅助数据库集群都会将主集群上可拥有的最大只读副本实例数减少一个。例如，如果有 4 个辅助集群，则主集群上可以拥有的最大只读副本实例数为 15 - 4 = 11。这意味着，如果主数据库集群和一个辅助集群中已经有 14 个读取器实例，则无法添加另一个辅助集群。

在 Neptune 中使用 AWS CLI 向全球数据库添加辅助区域

使用 AWS CLI 将辅助 AWS 区域添加到 Neptune 全球数据库

1. 使用 `create-db-cluster` AWS CLI 命令在与主集群不同的区域创建新的数据库集群，并设置其 `--global-cluster-identifier` 参数以指定全球数据库的 ID：

```
aws neptune create-db-cluster \
  --region (the secondary region) \
  --db-cluster-identifier (ID for the new secondary DB cluster) \
  --global-cluster-identifier (global database ID)
  --engine neptune \
  --engine-version (optional: engine version)
```

2. 使用 `describe-db-clusters` AWS CLI 命令确认新的数据库集群已准备就绪，可以添加其主数据库实例：

```
aws neptune describe-db-clusters \
  --region (primary region) \
  --db-cluster-identifier (primary DB cluster ID)
```

   当响应显示 `"Status": "available"` 时，继续执行下一步。

3. 使用 `create-db-instance` AWS CLI 命令，并使用 `r5` 或 `r6g` 实例类中的实例类型，为主集群创建主数据库实例：

```
aws neptune create-db-instance \
```

```
  --region (secondary region) \
  --db-cluster-identifier (secondary cluster ID) \
  --db-instance-class (instance class) \
  --db-instance-identifier (ID for the DB instance) \
  --engine neptune \
  --engine-version (optional: engine version)
```

> **Note**
>
> 如果您不打算在辅助区域中处理大量读取请求，而主要关心的是将数据可靠地备份到辅助区域，则可以创建一个没有数据库实例的辅助数据库集群。这样可以节省资金，因为这样您只需为辅助集群的存储付费，而 Neptune 会将此存储与主数据库集群中的存储保持同步。

连接到 Neptune 全球数据库

连接到 Neptune 全球数据库的方式取决于您是需要写入数据库还是从数据库读取：

- 对于只读请求或查询，请连接到 AWS 区域中 Neptune 集群的读取器端点。
- 要运行突变查询，请连接到主数据库集群的集群端点，该集群可能位于与应用程序不同的 AWS 区域中。

管理 Amazon Neptune 全球数据库

除了托管式计划内失效转移以外，您可对构成 Neptune 全球数据库的各个集群执行大多数的管理操作。托管式计划内失效转移过程仅适用于 Neptune 全球数据库，而不适用于单个 Neptune 数据库集群。要了解更多信息，请参阅 执行 Neptune 全球数据库的托管式计划内失效转移。

要从主区域的计划外停机中恢复 Neptune 全球数据库，请参阅在计划外停机时分离并提升 Neptune 全球数据库。

尽管您可以为全球数据库中的每个 Neptune 集群独立配置数据库集群参数组，但最好在所有集群之间保持设置一致，以避免在辅助集群提升为主集群时出现意外的行为变化。例如，对于所有数据库集群中的对象索引、流等使用相同设置。

从 Neptune 全球数据库中移除数据库集群

出于多种原因，您可能要从全球数据库中移除数据库集群。例如：

- 如果主集群已降级或处于隔离状态，则可以将其从全球数据库中移除，这样它就会变成一个可用于创建新的全球数据库的独立预调配集群（请参阅在计划外停机时分离并提升 Neptune 全球数据库）。
- 如果您要删除全球数据库，首先必须从全球数据库移除（分离）所有关联集群，仅保留主集群（请参阅删除 Neptune 全球数据库）。

您可以使用 remove-from-global-cluster CLI 命令（封装 RemoveFromGlobalCluster API）将 Neptune 数据库集群从全球数据库分离：

```
aws neptune remove-from-global-cluster \
  --region (region of the cluster to remove) \
  --global-cluster-identifier (global database ID) \
  --db-cluster-identifier (ARN of the cluster to remove)
```

然后，分离的数据库集群就会变成独立的数据库集群。

删除 Neptune 全球数据库

您不能通过单个步骤删除全球数据库及其关联集群。相反，您必须逐一删除其组件：

1. 如移除集群中所述，从全球数据库中分离所有辅助数据库集群。如果您愿意，您现在可以单独删除它们。
2. 从全球数据库分离主数据库集群。
3. 从主集群中删除所有只读副本数据库实例。
4. 从主集群删除主（写入器）数据库实例。如果您在控制台上执行此操作，它也会删除数据库集群。
5. 删除全球数据库本身。要使用 AWS CLI 执行此操作，请使用 delete-global-cluster CLI 命令（封装 DeleteGlobalCluster API），如下所示：

   ```
   aws neptune delete-global-cluster \
     --region (region of the DB cluster to delete) \
     --global-cluster-identifier (global database ID)
   ```

修改 Neptune 全球数据库

可以为全球数据库中的每个 Neptune 数据库集群单独配置数据库集群参数组，但最好在集群之间保持设置的一致性，以避免在必须将辅助集群提升为主集群时出现意外行为变化。

您可以使用 modify-global-cluster CLI 命令（封装 ModifyGlobalCluster API）修改全球数据库本身的设置。例如，您可以更改全球数据库标识符，同时关闭删除保护，如下所示：

```
aws neptune modify-global-cluster \
  --region (region of the DB cluster to modify) \
  --global-cluster-identifier (current global database ID) \
  --new-global-cluster-identifier (new global database ID to assign) \
  --deletion-protection false
```

在 Neptune 全球数据库中使用失效转移

Neptune 全球数据库提供的失效转移功能比独立 Neptune 数据库集群提供的失效转移功能更全面。使用全球数据库，您可以对灾难进行规划并很快地从灾难中恢复。通常使用对恢复时间目标 (RTO) 和恢复点目标 (RPO) 的评测来评测灾难恢复：

- 恢复时间目标 (RTO) — 灾难后系统恢复工作状态的速度。换言之，RTO 用于衡量停机时间。对于 Neptune 全球数据库，RTO 大约为数分钟。
- 恢复点目标 (RPO) — 数据丢失的时间段。对于 Neptune 全球数据库，RPO 通常以秒为单位进行测量（请参阅执行 Neptune 全球数据库的托管式计划内失效转移）。

对于 Neptune 全球数据库，失效转移方法有两种不同的方法：

- 分离并提升（手动计划外恢复）— 要从计划外停机中恢复或进行灾难恢复测试（DR 测试），您可以对全球数据库中的辅助数据库集群之一执行跨区域分离并提升。此手动过程的 RTO 取决于您执行 分离并提升 中列出的任务的速度。RPO 通常为数秒，但这取决于发生故障时整个网络的存储复制滞后时间。
- 托管式计划内失效转移 — 此方法适用于操作维护和其它计划内操作过程，例如将全球数据库的主数据库集群迁移到辅助区域之一。由于此过程会在进行任何其它更改之前将辅助数据库集群与主数据库集群同步，因此 RPO 实际上为 0（也即不会造成数据丢失）。请参阅执行 Neptune 全球数据库的托管式计划内失效转移。

在计划外停机时分离并提升 Neptune 全球数据库

在极少数情况下，Neptune 全球数据库在主 AWS 区域中发生意外中断，主 Neptune 数据库集群及其写入器节点将变得不可用，且主集群和辅助集群之间的复制将停止。为了最大限度地减少由此产生的停机时间 (RTO) 和数据丢失 (RPO)，请快速执行跨区域分离和提升以重建全球数据库。

> **Tip**
>
> 在使用此过程之前，最好先了解一下该过程，并制定一个计划，以便在出现区域范围问题的第一个迹象时迅速采取行动。
>
> - 定期使用 Amazon CloudWatch 来跟踪辅助集群的滞后时间，以便在需要失效转移时，您可以确定滞后时间最小的辅助区域。
> - 确保测试您的计划，以检查过程是否完整和准确。
> - 使用模拟环境以确保您的员工接受过培训，准备好在必要时快速执行灾难恢复失效转移。

在主区域发生计划外停机后失效转移到辅助集群

1. 停止在主数据库集群上发出突变查询和其它写入操作。
2. 在辅助 AWS 区域中确定一个数据库集群，以用作全球数据库的新主数据库集群。如果全球数据库有两个或更多辅助 AWS 区域，请选择滞后时间最小的辅助集群。
3. 从 Neptune 全球数据库中分离您选择的辅助数据库集群。

   从 Neptune 全球数据库中移除辅助数据库集群会立即停止将数据从主数据库集群复制到该辅助数据库集群的过程，并会将其提升为拥有完全读/写功能的独立数据库集群。全球数据库中的任何其它辅助集群仍可用，并且可以接受来自应用程序的读取调用。

   在重新创建 Neptune 全球数据库之前，您还必须分离其它辅助集群，以避免集群间的数据不一致（请参阅移除集群）。
4. 重新配置应用程序，以使用新的端点将所有写入操作发送到您选择成为新主集群的独立 Neptune 数据库集群。如果您在创建 Neptune 全球数据库时接受了默认名称，则可以在应用程序中从集群的端点字符串中移除 `-ro` 以更改端点。

   例如，辅助集群的端点 `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.neptune.amazonaws.com` 将在该集群从全球数据库分离时变为 `my-global.cluster-aaaaaabbbbbb.us-west-1.neptune.amazonaws.com`。

   在下一步中，当您开始向此 Neptune 数据库集群添加区域时，该 Neptune 数据库集群将成为新的 Neptune 全球数据库的主集群。
5. 向数据库集群添加 AWS 区域。执行此操作后，从主数据库集群到辅助数据库集群的复制过程将会开始。请参阅在 Amazon Neptune 中向主区域添加辅助全球数据库区域。
6. 根据需要添加更多 AWS 区域，以重新创建为了支持应用程序所需的拓扑。

确保在进行这些更改之前、期间和之后，将应用程序写入发送到正确的 Neptune 数据库集群。这样做可以避免 Neptune 全球数据库中数据库集群之间的数据不一致（这些问题称为脑裂问题）。

执行 Neptune 全球数据库的托管式计划内失效转移

托管式计划内失效转移允许您将 Neptune 全球数据库的主集群重新定位到您选择的不同 AWS 区域。一些组织希望定期轮换其主集群位置。

> **Note**
>
> 此处介绍的托管式计划内失效转移适合在运行正常的 Neptune 全球数据库上使用。要从计划外停机中恢复或进行灾难恢复 (DR) 测试，请改为按照分离并提升过程进行操作。

在托管式计划内失效转移期间，主集群会将失效转移到您选择的辅助区域，同时保留全球数据库的现有复制拓扑。在托管式计划内失效转移过程开始之前，全球数据库会将所有辅助集群与其主集群同步。确保所有集群都同步后，托管的计划内故障转移才会开始。主区域中的数据库集群变为只读的，所选辅助集群将其一个只读实例提升为完全写入器状态，从而允许该集群担任主集群的角色。由于所有辅助集群在过程开始时都与主集群同步，因此新的主集群将继续执行全球数据库的操作，而不会丢失任何数据。数据库仅在短时间内不可用，而主集群和所选的辅助集群将担任其新角色。

要优化应用程序可用性，您可以在非高峰时间段（向主数据库集群写入操作最少的时候）执行失效转移。此外，在开始失效转移之前，请采取以下步骤：

- 尽可能使应用程序离线，以减少对主集群的写入。
- 检查全球数据库中所有辅助 Neptune 数据库集群的滞后时间，然后选择总体滞后时间最少的辅助集群成为主集群。使用 Amazon CloudWatch 查看所有辅助数据库集群的 `NeptuneGlobalDBProgressLag` 指标。此指标会指出辅助数据库集群滞后于主数据库集群的时间（以毫秒为单位）。它的值与 Neptune 完成失效转移所需的时间成正比。换言之，滞后值越大，失效转移停机时间越长，因此请选择滞后值最小的辅助数据库集群。参阅 Neptun CloudWatch e 指标了解更多信息。

在托管式计划内失效转移期间，所选的辅助数据库集群将提升为新角色（即主数据库集群），但它不会继承主数据库集群的完全配置。配置不匹配可能会导致性能问题、工作负载不兼容和其他异常行为。要避免此类问题，请在失效转移之前解决全球数据库集群之间的以下几种配置差异：

- 在新的主数据库集群中配置参数以匹配当前主数据库集群。

- 配置监控工具、选项和警报 — 配置将成为新的主数据库集群的数据库集群，该集群与当前主数据库具有相同的日志记录功能、警报等。
- 配置与其它 AWS 服务的集成 — 如果 Neptune 全球数据库与 AWS 服务集成，如 AWS Identity and Access Management (IAM)、Amazon S3 或 AWS Lambda，请确保根据需要配置这些服务，以便与新的主数据库集群集成。

当失效转移过程完成并且提升的数据库集群已准备好处理全球数据库的写入操作时，请确保更改您的应用程序以使用新的主数据库的新端点。

使用 AWS CLI 启动托管式计划内失效转移

使用 failover-global-cluster CLI 命令（封装 FailoverGlobalCluster API）对您的 Neptune 全球数据库进行失效转移：

```
aws neptune failover-global-cluster \
  --region (the region where the primary cluster is located) \
  --global-cluster-identifier (global database ID) \
  --target-db-cluster-identifier (the ARN of the secondary DB cluster to promote)
```

> **Note**
>
> failover-global-cluster API 在预览版中不可用。它将成为 GA 版本的一部分。

使用 CloudWatch 指标监控 Neptune 全球数据库

Neptune 支持以下 CloudWatch 指标，您可以使用这些指标来监控 Neptune 全球数据库：

- **GlobalDbDataTransferBytes** – 在 Neptune 全球数据库中，从主要 AWS 区域传输到辅助 AWS 区域的重做日志数据的字节数。
- **GlobalDbReplicatedWriteIO** – 这是在全球数据库中从主要 AWS 区域复制到辅助 AWS 区域中的集群卷的写入 I/O 操作数。

  Neptune 全球数据库中每个数据库集群的账单计算使用 VolumeWriteIOPS 指标来考虑在该集群中执行的写入。对于主数据库集群，账单计算使用 NeptuneGlobalDbReplicatedWriteIO 考虑向辅助数据库集群的跨区域复制。
- **GlobalDbProgressLag** – 对于用户事务和系统事务，辅助集群落后于主集群的毫秒数。

指标	维度	发布位置	单位
GlobalDbDataTransferBytes	SourceRegion、DBClusterIdentifier	辅助	字节
GlobalDbReplicatedWriteIO	SourceRegion、DBClusterIdentifier	辅助	计数
GlobalDbProgressLag	DBClusterIdentifier、SecondaryRegion：在主数据库集群中；DBClusterIdentifier、SourceRegion：在辅助数据库集群中	主、辅助	毫秒

Amazon Neptune 特征概述

本部分提供特定 Neptune 特征的概述，包括：

- Neptune 与查询语言标准的符合性。
- Neptune 的图形数据模型。
- 对 Neptune 事务语义的解释。
- Neptune 集群和实例简介。
- Neptune 的存储、可靠性和可用性。
- 对 Neptune 端点的解释。
- Neptune 的自定义查询 ID 如何让您检查查询状态。
- 使用 Neptune 的实验室模式启用实验特征。
- Neptune 的 DFE 引擎的描述。
- Neptune 的 JDBC 连接。
- Neptune 引擎版本列表以及如何更新引擎。

> **Note**
>
> 本节不介绍使用可用于访问 Neptune 图形中数据的查询语言。
> 有关如何使用 Gremlin 连接到正在运行的 Neptune 数据库集群的信息，请参阅使用 Gremlin 访问 Neptune 图形。
> 有关如何使用 openCypher 连接到正在运行的 Neptune 数据库集群的信息，请参阅使用 openCypher 访问 Neptune 图形。
> 有关如何使用 SPARQL 连接到正在运行的 Neptune 数据库集群的信息，请参阅使用 SPARQL 访问 Neptune 图形。

主题

- 关于 Amazon Neptune 标准合规性的说明
- Neptune 图形数据模型
- Neptune 查找缓存可以加快读取查询的速度
- Neptune 中的事务语义
- Amazon Neptune 数据库集群和实例

- Amazon Neptune 存储、可靠性和可用性
- 连接到 Amazon Neptune 端点
- 将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中
- Neptune 实验室模式
- Amazon Neptune 替代查询引擎 (DFE)
- 管理 Neptune DFE 要使用的统计数据
- 获取有关图形的快速摘要报告
- Amazon Neptune JDBC 连接
- Amazon Neptune 引擎更新

关于 Amazon Neptune 标准合规性的说明

在大多数情况下，Amazon Neptune 符合实施 Gremlin 和 SPARQL 图形查询语言的适用标准。

这些章节介绍了这些标准以及 Neptune 扩展或偏离这些标准的领域。

主题

Amazon Neptune 中的 Gremlin 标准合规性

以下各节概述了 Gremlin 的 Neptune 实现以及它与 Apache 实现有何不同。 TinkerPop

Neptune 在其引擎中原生实现了一些 Gremlin 步骤，并使用 Apache TinkerPop Gremlin 实现来处理其他步骤（参见）。Amazon Neptune 中的原生 Gremlin 步骤支持

> **Note**
>
> 有关 Gremlin 控制台和 Amazon Neptune 中所示的这些实施差别的一些具体示例，请参阅“快速入门”的the section called “使用 Gremlin”部分。

主题

- 顶点属性 ID
- 顶点属性的基数
- 更新顶点属性
- 标签
- 转义字符
- Groovy 限制
- 序列化
- Lambda 步骤
- 不受支持的 Gremlin 方法
- 不受支持的 Gremlin 步骤
- Neptune 中的 Gremlin 图形特征

Gremlin 的适用标准

- Gremlin 语言由 Apache TinkerPop 文档和 Gremlin 的 Apache TinkerPop 实现定义，而不是由正式规范定义。
- 对于数字格式，Gremlin 遵循 IEEE 754 标准（IEEE 754-2019 - IEEE 浮点算法标准。有关更多信息，请参阅维基百科 IEEE 第 754 页）。

脚本中的变量和参数

就预绑定变量而言，遍历对象 g 在 Neptune 中是预先绑定的，并且不支持 graph 对象。

尽管 Neptune 在脚本中不支持 Gremlin 变量或参数化，但您可能经常在互联网上遇到包含变量声明的 Gremlin Server 示例脚本，例如：

```
String query = "x = 1; g.V(x)";
List<Result> results = client.submit(query).all().get();
```

还有许多在提交查询时利用参数化（或绑定）的示例，例如：

```
Map<String,Object> params = new HashMap<>();
params.put("x",1);
String query = "g.V(x)";
List<Result> results = client.submit(query).all().get();
```

参数示例通常与在可能的情况下未进行参数化会导致性能损失的警告相关联。你可能会遇到很多这样的例子 TinkerPop ，而且它们在参数化的必要性方面听起来都很有说服力。

但是，变量声明功能和参数化功能（以及警告）仅在使用 Gremlin Server 时才适用于该服务器。TinkerPop `GremlinGroovyScriptEngine`当 Gremlin 服务器使用 Gremlin 的 `gremlin-language` ANTLR 语法来解析查询时，它们不适用。ANTLR 语法不支持变量声明或参数化，因此，在使用 ANTLR 时，您不必担心参数化失败。由于ANTLR语法是一个较新的组成部分 TinkerPop，因此您在互联网上可能遇到的较旧内容通常无法反映出这种区别。

Neptune 在其查询处理引擎中使用 ANTLR 语法，而不是 `GremlinGroovyScriptEngine`，因此它不支持变量、参数化或 `bindings` 属性。因此，与参数化失败相关的问题在 Neptune 中不适用。使用 Neptune，只需按通常发生参数化的位置提交查询就完全安全了。因此，可以简化前面的示例，而不会造成任何性能损失，如下所示：

```
String query = "g.V(1)";
List<Result> results = client.submit(query).all().get();
```

TinkerPop 枚举

Neptune 对于枚举值不支持完全限定的类名。例如，您在 Groovy 请求中必须使用 `single`，不能使用 `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single`。

枚举类型由参数类型决定。

下表显示了允许的枚举值和相关的 TinkerPop 完全限定名称。

允许的值	类
`id`, `key`, `label`, `value`	org.apache.tinkerpop.gremlin.structure.T
`T.id`, `T.key`, `T.label`, `T.value`	org.apache.tinkerpop.gremlin.structure.T
`set`, `single`	org.apache.tinkerpop.gremlin.structure。VertexProperty.Cardinality
`asc`, `desc`, `shuffle`	org.apache.tinkerpop.gremlin.process.traversal.Order
`Order.asc` ，`Order.desc` ，`Order.shuffle`	org.apache.tinkerpop.gremlin.process.traversal.Order

`global`, `local`	org.apache.tinkerpop.gremlin.process.traversal.Scope
`Scope.global` , `Scope.local`	org.apache.tinkerpop.gremlin.process.traversal.Scope
`all`, `first`, `last`, `mixed`	org.apache.tinkerpop.gremlin.process.traversal.Pop
`normSack`	org.apache.tinkerpop.gremlin.process.traversal。 SackFunctions. Barrier
`addAll`, `and`, `assign`, `div`, `max`, `min`, `minus`, `mult`, `or`, `sum`, `sumLong`	org.apache.tinkerpop.gremlin.process.traversal.Operator
`keys`, `values`	org.apache.tinkerpop.gremlin.structure.Column
`BOTH`, `IN`, `OUT`	org.apache.tinkerpop.gremlin.structure.Direction
`any`, `none`	org.apache.tinkerpop.gremlin.process.traversal.step。 TraversalOption家长。选择

Java 代码

Neptune 不支持调用由支持的 Gremlin API 以外的任意 Java 或 Java 库调用定义的方法。例如，不允许 `java.lang.*`、`Date()` 和 `g.V().tryNext().orElseGet()`。

元素的属性

Neptune 不支持 TinkerPop 3.7.0 中引入的用于返回元素属性的`materializeProperties`标志。因此，Neptune 仍然只会将顶点或边作为参照返回，只返回它们的和。`id` `label`

脚本执行

所有查询必须以遍历对象 g 开头。

在字符串查询提交中，可以使用分号 (`;`) 或换行符 (\n) 进行分隔来发布多个遍历。若要执行，除最后一个语句以外的每个语句都必须以 `.iterate()` 步骤结尾。仅返回最后遍历数据。请注意，这不适用于 GLV ByteCode 查询提交。

会话

Neptune 中的会话持续时间限制为仅 10 分钟。有关更多信息，请参阅基于 Gremlin 脚本的会话和TinkerPop会话参考。

事务

Neptune 在每个 Gremlin 遍历开始时打开新的事务，并在遍历成功完成后结束事务。事务会在出现错误时回滚。

用分号 (;) 或换行符 (\n) 分隔的多个语句包含在单个事务中。最后一个以外的每个语句都必须以要执行的 `next()` 步骤结尾。仅返回最后遍历数据。

不支持使用 `tx.commit()` 和 `tx.rollback()` 的手动事务逻辑。

> **Important**
>
> 这仅适用于您将 Gremlin 查询作为文本字符串发送的方法（请参阅Gremlin 事务）。

顶点和边缘 ID

Neptune Gremlin 顶点和边缘 ID 必须属于 `String` 类型。这些 ID 字符串支持 Unicode 字符，大小不能超过 55MB。

支持用户提供的 ID，但它们在正常使用中是可选的。如果您在添加顶点或边缘时不提供 ID，Neptune 会生成一个 UUID 并将其转换为字符串，格式如下：`"48af8178-50ce-971a-fc41-8c9a954cea62"`。这些 UUID 不符合 RFC 标准，因此，如果您需要标准 UUID，则应在外部生成它们，并在添加顶点或边缘时提供它们。

> **Note**
>
> Neptune Load 命令要求您使用 ~id 字段采用 Neptune CSV 格式提供 ID。

用户提供的 ID

用户提供的 ID 在 Neptune Gremlin 中得到允许，且具有以下规定。

- 提供的 ID 是可选的。
- 仅支持顶点和边缘。

- 仅支持 `String` 类型。

要使用自定义 ID 创建新顶点，请将 `property` 步骤与 `id` 关键字一起使用：`g.addV().property(id, 'customid')`。

> **Note**
>
> 请勿为 `id` 关键字加上引号。它指的是 `T.id`。

所有顶点 ID 必须是唯一的，并且所有边缘 ID 也必须是唯一的。但是，Neptune 的确允许顶点和边缘具有相同的 ID。

如果您尝试使用 `g.addV()` 创建新顶点并且已存在具有该 ID 的顶点，则此操作将失败。此情况的例外是，如果为顶点指定新标签，该操作将成功，但会将新标签和指定的任何其他属性添加到现有顶点。不会覆盖任何内容。未创建新顶点。顶点 ID 不会更改并会保持唯一。

例如，以下 Gremlin 控制台命令将成功：

```
gremlin> g.addV('label1').property(id, 'customid')
gremlin> g.addV('label2').property(id, 'customid')
gremlin> g.V('customid').label()
==>label1::label2
```

顶点属性 ID

顶点属性 ID 会自动生成且可以在查询时显示为正数或负数。

顶点属性的基数

Neptune 支持集基数和单一基数。如果未指定，则集基数处于选中状态。这意味着，如果您设置一个属性值，它会向该属性添加新值，但是仅当它未显示在一组值中时。这是 Set 的 Gremlin 枚举值。

不支持 `List`。有关属性基数的更多信息，请参阅 Gremlin 中的 Vertex 主题。 JavaDoc

更新顶点属性

要更新属性值而无需向一组值添加其他值，请在 `property` 步骤中指定 `single` 基数。

```
g.V('exampleid01').property(single, 'age', 25)
```

这将删除该属性的所有现有值。

标签

Neptune 对于一个顶点支持多个标签。创建标签时，您可以指定多个标签，同时使用 `::` 分隔它们。例如，`g.addV("Label1::Label2::Label3")` 添加具有三个不同标签的顶点。`hasLabel` 步骤与具有以下任一三个标签的此顶点相匹配：`hasLabel("Label1")`、`hasLabel("Label2")` 和 `hasLabel("Label3")`。

> **Important**
>
> `::` 分隔符仅用于此用途。您不能在 `hasLabel` 步骤中指定多个标签。例如，`hasLabel("Label1::Label2")` 与任何内容都不匹配。

转义字符

Neptune 解析所有转义字符，如 Apache Groovy 语言文档的转义特殊字符部分中所述。

Groovy 限制

Neptune 不支持不以 g 开头的 Groovy 命令。这包括数学（例如：1+1）、系统调用（例如：`System.nanoTime()`）和变量定义（例如：1+1）。

> **Important**
>
> Neptune 不支持完全限定类名称。例如，您在 Groovy 请求中必须使用 `single`，不能使用 `org.apache.tinkerpop.gremlin.structure.VertexProperty.Cardinality.single`。

序列化

Neptune 根据请求的 MIME 类型支持以下串行化。

MIME 类型	序列化	配置
`application/vnd.gremlin-v1.0+gryo`	`GryoMessageSerializerV1`	`ioRegistries: [org.apache.tinkerpop.gremlin.tinker`

		graph.structure.TinkerIoRegistryV1]
application/vnd.gremlin-v1.0+gryo-stringd	GryoMessageSerializerV1	serializeResultToString: true}
application/vnd.gremlin-v3.0+gryo	GryoMessageSerializerV3	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3]
application/vnd.gremlin-v3.0+gryo-stringd	GryoMessageSerializerV3	serializeResultToString: true
application/vnd.gremlin-v1.0+json	GraphSONMessageSerializerGremlinV1	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV1]
application/vnd.gremlin-v2.0+json	GraphSONMessageSerializerV2 （仅适用于 WebSockets ）	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV2]
application/vnd.gremlin-v3.0+json	GraphSONMessageSerializerV3	
application/json	GraphSONMessageSerializerV3	ioRegistries: [org.apache.tinkerpop.gremlin.tinkergraph.structure.TinkerIoRegistryV3]

```
application/vnd.gr      GraphBinaryMessage
aphbinary-v1.0          SerializerV1
```

虽然 Neptune 支持这些不同的序列化器类型，但它们的使用指南却相当简单。如果您通过 HTTP 连接到 Neptune，请优先使用替代版本 GraphSon 3 中的嵌入式类型，这样使用起来会变得复杂。`application/vnd.gremlin-v3.0+json;types=false`如果您使用的是 Apache TinkerPop 驱动程序，则可能不需要像使用默认驱动程序那样做出任何选择。`application/vnd.graphbinary-v1.0`由于遗留原因，其余格式仍然存在。

> **Note**
>
> 此处显示的序列化器表指的是从 TinkerPop 3.7.0 开始的命名。如果您想了解有关此更改的更多信息，请参阅TinkerPop 升级文档。Gryo 序列化支持在 3.4.3 中已被弃用，并在 3.6.0 中正式删除。如果您明确使用Gryo或默认使用Gryo的驱动程序版本，则应切换到 GraphBinary 或升级驱动程序。

Lambda 步骤

Neptune 不支持 Lambda 步骤。

不受支持的 Gremlin 方法

Neptune 不支持以下 Gremlin 方法：

- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.progra`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.sideEf`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.from(o`
- `org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversal.to(org`

例如，不允许以下遍历：`g.V().addE('something').from(__.V().next()).to(__.V().next())`。

> **Important**
>
> 这仅 适用于您将 Gremlin 查询作为文本字符串 发送的方法。

不受支持的 Gremlin 步骤

Neptune 不支持以下 Gremlin 步骤：

- Neptune 中仅部分支持 Gremlin io( ) 步骤。它可以在读取上下文中使用，如在 `g.io(`*`url`*`).read()` 中，但不能用于写入。

Neptune 中的 Gremlin 图形特征

Gremlin 的 Neptune 实现不公开 `graph` 对象。下表列出了 Gremlin 特征，并指明了 Neptune 是否支持这些特征。

Neptune 对 **`graph`** 特征的支持

Neptune 图形特征（如果支持）与 `graph.features()` 命令返回的特征相同。

图形特征	是否启用？
`Transactions`	true
`ThreadedTransactions`	false
`Computer`	false
`Persistence`	true
`ConcurrentAccess`	true

Neptune 对变量特征的支持

变量特征	是否启用？
`Variables`	false
`SerializableValues`	false
`UniformListValues`	false
`BooleanArrayValues`	false

DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	false
ByteValues	false
DoubleValues	false
FloatValues	false
IntegerValues	false
LongValues	false
MapValues	false
MixedListValues	false
StringValues	false
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Neptune 对顶点特征的支持

顶点特征	是否启用？
MetaProperties	false
DuplicateMultiProperties	false
AddVertices	true
RemoveVertices	true

MultiProperties	true
UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Neptune 对顶点属性特征的支持

顶点属性特征	是否启用？
UserSuppliedIds	false
AddProperty	true
RemoveProperty	true
NumericIds	true
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false
Properties	true
SerializableValues	false

UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true
DoubleValues	true
FloatValues	true
IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Neptune 对边缘特征的支持

边缘特征	是否启用？
AddEdges	true
RemoveEdges	true

UserSuppliedIds	true
AddProperty	true
RemoveProperty	true
NumericIds	false
StringIds	true
UuidIds	false
CustomIds	false
AnyIds	false

Neptune 对边缘属性特征的支持

边缘属性特征	是否启用？
Properties	true
SerializableValues	false
UniformListValues	false
BooleanArrayValues	false
DoubleArrayValues	false
IntegerArrayValues	false
StringArrayValues	false
BooleanValues	true
ByteValues	true
DoubleValues	true
FloatValues	true

IntegerValues	true
LongValues	true
MapValues	false
MixedListValues	false
StringValues	true
ByteArrayValues	false
FloatArrayValues	false
LongArrayValues	false

Amazon Neptune 中的 SPARQL 标准合规性

在列出了适用的 SPARQL 标准之后，以下各节提供有关 Neptune 的 SPARQL 实现如何扩展或偏离这些标准的具体细节。

主题

- SPARQL 的适用标准
- Neptune SPARQL 中的默认命名空间前缀
- SPARQL 默认图形和命名图形
- Neptune 支持的 SPARQL XPath 构造函数
- 用于查询和更新的默认基本 IRI
- Neptune 中的 xsd:dateTime 值
- Neptune 的特殊浮点值处理
- Neptune 的任意长度值限制
- Neptune 扩展了 SPARQL 中的等于比较
- 处理 Neptune SPARQL 中超出范围的文本

在实施 SPARQL 图形查询语言时，Amazon Neptune 符合以下标准。

SPARQL 的适用标准

- SPARQL 根据 2013 年 3 月 21 日发布的 W3C SPARQL 1.1 查询语言建议定义。
- SPARQL 更新协议和查询语言由 W3C SPARQL 1.1 更新规范定义。
- 对于数字格式，SPARQL 遵循 W3C XML 架构定义语言 (XSD) 1.1 第 2 部分：数据类型规范，该规范与 IEEE 754 规范一致（IEEE 754-2019-IEEE 浮点算法标准。有关更多信息，请参阅维基百科 IEEE 第 754 页）。但是，IEEE 754-1985 版本之后引入的功能不包括在该规范中。

Neptune SPARQL 中的默认命名空间前缀

默认情况下，Neptune 定义了以下前缀以用于 SPARQL 查询。有关更多信息，请参阅 SPARQL 规范中的前缀名称。

- `rdf` – `http://www.w3.org/1999/02/22-rdf-syntax-ns#`
- `rdfs` – `http://www.w3.org/2000/01/rdf-schema#`
- `owl` – `http://www.w3.org/2002/07/owl#`
- `xsd` – `http://www.w3.org/2001/XMLSchema#`

SPARQL 默认图形和命名图形

Amazon Neptune 将每个三元组与一个命名图形相关联。默认图形定义为所有命名图形的并集。

查询的默认图形

如果提交 SPARQL 查询而未通过 GRAPH 关键字或 FROM NAMED 之类的构造明确指定图形，Neptune 将始终考虑数据库实例中的所有三元组。例如，以下查询从 Neptune SPARQL 端点返回所有三元组：

```
SELECT * WHERE { ?s ?p ?o }
```

显示在多个图形中的三元组将仅返回一次。

有关默认图形规范的信息，请参阅 SPARQL 1.1 查询语言规范的 RDF 数据集一节。

为加载、插入或更新指定命名图形

如果加载、插入或更新三元组时未指定命名图形，Neptune 使用 URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` 定义的回退命名图形。

使用基于三元组的格式发出 Neptune Load 请求时，可使用 parserConfiguration: namedGraphUri 参数指定命名图形以用于所有三元组。有关 Load 命令语法的信息，请参阅the section called “加载程序命令”。

> **Important**
>
> 如果您不使用此参数，并且未指定命名图形，将使用回退 URI：http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph。

如果通过 SPARQL UPDATE 加载三元组而未显式提供命名图形目标，也将使用此回退命名图形。

可使用基于四元组的 N-Quads 格式为数据库中的每个三元组指定一个命名图形。

> **Note**
>
> 您可以使用 N-Quads 将命名图形保留为空。在本例中，将使用 http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph。
> 使用 namedGraphUri 解析程序配置选项，可以覆盖 N-Quads 的默认命名图形。

Neptune 支持的 SPARQL XPath 构造函数

SPARQL 标准允许 SPARQL 引擎支持一组可扩展的 XPath 构造函数。Neptune 目前支持以下构造函数，其中 xsd 前缀定义为 http://www.w3.org/2001/XMLSchema#：

- xsd:boolean
- xsd:integer
- xsd:double
- xsd:float
- xsd:decimal
- xsd:long
- xsd:unsignedLong

用于查询和更新的默认基本 IRI

由于 Neptune 集群有多个不同的端点，因此，将查询或更新的请求 URL 用作基本 IRI 可能会在解析相对 IRI 时导致意外结果。

从引擎版本 1.2.1.0 开始，如果请求中没有显式基本 IRI，Neptune 将使用 `http://aws.amazon.com/neptune/default/` 作为基本 IRI。

在以下请求中，基本 IRI 是请求的一部分：

```
BASE <http://example.org/default/>
INSERT DATA { <node1> <id> "n1" }

BASE <http://example.org/default/>
SELECT * { <node1> ?p ?o }
```

结果将是：

```
?p                                                  ?o
http://example.org/default/id                       n1
```

但是，此请求中不包括基本 IRI：

```
INSERT DATA { <node1> <id> "n1" }

SELECT * { <node1> ?p ?o }
```

在这种情况下，结果将是：

```
?p                                                  ?o
http://aws.amazon.com/neptune/default/id            n1
```

Neptune 中的 xsd:dateTime 值

出于性能原因，Neptune 始终将日期/时间值存储为协调世界时 (UTC) 形式。这可让直接比较非常有效。

此外，这意味着，如果您输入一个指定特定时区的 `dateTime` 值，则 Neptune 会将此值转换为 UTC 形式并丢弃该时区信息。当您稍后检索该 `dateTime` 值时，它会以 UTC 形式表示，而不是原始时区的时间，并且您再也无法确定原始时区。

Neptune 的特殊浮点值处理

Neptune 按如下所示处理 SPARQL 中的特殊浮点值。

在 Neptune 中处理 SPARQL NaN

在 Neptune 中，SPARQL 可以接受查询中的 NaN 值。信号和安静 NaN 值之间没有区别。Neptune 将所有 NaN 值都视为安静值。

从语义上讲，无法进行 NaN 比较，因为没有什么大于、小于或等于 NaN。这意味着比较一侧的 NaN 值理论上永远不会匹配另一侧的任何 内容。

但是，XSD 规范确实将两个 `xsd:double` 或 `xsd:float` NaN 值视为相等。对于 IN 筛选条件、筛选表达式中的等于运算符以及完全匹配语义（在三元模式的对象位置具有 NaN），Neptune 遵循此规则。

Neptune 中的 SPARQL 无限值处理

在 Neptune 中，SPARQL 可以接受查询中的 INF 或 -INF 值。INF 的比较大于任何其它数值，而 -INF 的比较小于任何其它数值。

不管值的类型如何，带匹配符号的两个 INF 值的比较结果是相等（例如，浮点 -INF 等于双精度 -INF）。

当然，无法进行 NaN 比较，因为没有什么内容大于、小于或等于 NaN。

Neptune 中的 SPARQL 负零处理

Neptune 将负零值标准化为无符号零。您可以在查询中使用负零值，但它们不会原样记录在数据库中，并且在比较时与无符号零相等。

Neptune 的任意长度值限制

Neptune 将 SPARQL 中的 XSD 整数、浮点数和十进制值的存储大小限制为 64 位。使用较大的值会导致 `InvalidNumericDataException` 错误。

Neptune 扩展了 SPARQL 中的等于比较

SPARQL 标准定义了值表达式的三元逻辑，其中值表达式可求值为 `true`、`false` 或 `error`。SPARQL 1.1 规范中定义的术语相等的默认语义适用于 FILTER 条件下的 = 和 != 比较，在比较规范的运算符表中不是明确可比较的数据类型时，会产生 `error`。

此行为可能会导致不直观的结果，如以下示例所示。

数据：

```
<http://example.com/Server/1> <http://example.com/ip> "127.0.0.1"^^<http://example.com/
datatype/IPAddress>
```

查询 1：

```
SELECT * WHERE {
    <http://example.com/Server/1> <http://example.com/ip> ?o .
    FILTER(?o = "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

查询 2：

```
SELECT * WHERE {
    <http://example.com/Server/1> <http://example.com/ip> ?o .
    FILTER(?o != "127.0.0.2"^^<http://example.com/datatype/IPAddress>)
}
```

借助 Neptune 在版本 1.0.2.1 之前使用的默认 SPARQL 语义，两个查询均返回空结果。原因是，`?o = "127.0.0.2"^^<http://example.com/IPAddress>` 在为 `?o := "127.0.0.1"^^<http://example.com/IPAddress>` 评估时生成 `error` 而不是 `false`，因为没有为自定义数据类型 `<http://example.com/IPAddress>` 指定明确的比较规则。因此，第二个查询中的否定版本也会生成 `error`。在这两个查询中，`error` 会导致筛选掉候选解决方案。

从版本 1.0.2.1 开始，Neptune 根据规范，扩展了 SPARQL 不等于运算符。请参阅 SPARQL 1.1 关于运算符可扩展性的部分，该部分允许引擎定义如何对用户定义的数据类型和不可比较的内置数据类型进行比较的其他规则。

利用此选项，Neptune 现在对在运算符映射表中未显式定义的任意两种数据类型进行比较时，如果文本值和数据类型在语法上相等，则视为 `true`，否则视为 false。在任何情况下都不会产生 `error`。

使用这些新语义，第二个查询将返回 `"127.0.0.1"^^<http://example.com/IPAddress>` 而不是空的结果。

处理 Neptune SPARQL 中超出范围的文本

XSD 语义定义每个数字类型及其值空间（`integer` 和 `decimal` 除外）。这些定义将每种类型限制为一个值范围。例如，`xsd:byte` 范围是从 -128 到 +127（含）。超出此范围的任何值都被视为无效。

如果您尝试在类型的值空间之外分配文字值（例如，如果您尝试将的文字值设置为 999），Neptune 会按原样接受该 out-of-range 值，而不对其进行四舍五入或`xsd:byte`截断。但它不会将该值保留为数字值，因为给定的类型无法表示数字值。

也就是说，即使 `"999"^^xsd:byte` 是一个超出定义的 `xsd:byte` 值范围的值，Neptune 也可以接受它。但是，在数据库中保留该值后，它只能在三元模式的宾语位置用于完全匹配语义。无法对其执行范围过滤器，因为 out-of-range 字面值不被视为数值。

SPARQL 1.1 规范以 `numeric-operator-numeric`、`string-operator-string`、`literal-operator-literal` 等形式定义范围运算符。Neptune 无法执行诸如 `invalid-literal-operator-numeric-value` 之类的范围比较运算符。

亚马逊 Neptune 中符合 OpenCypher 规范

Amazon Neptune 版本的 openCypher 通常支持当前 openCypher 规范（即 Cypher 查询语言参考版本 9）中定义的子句、运算符、表达式、函数和语法。下面列出了 Neptune 对 openCypher 的支持的局限性和差异。

> **Note**
>
> Cypher 的当前 Neo4j 实现包含上述 openCypher 规范中未包含的功能。如果您要将当前 Cypher 代码迁移到 Neptune，请参阅Neptune 与 Neo4j 的兼容性和重写 Cypher 查询以在 Neptune 上的 openCypher 中运行以了解更多信息。

Neptune 中对 openCypher 子句的支持

除非另有说明，否则 Neptune 支持以下子句：

- `MATCH` – 支持，但除了目前不支持 *`shortestPath()`* 和 *`allShortestPaths()`* 之外。
- `OPTIONAL MATCH`
- *`MANDATORY MATCH`* –Neptune 目前不支持。但是，Neptune 确实支持在 `MATCH` 查询中使用自定义 ID 值。
- `RETURN` – 支持，但与 `SKIP` 或 `LIMIT` 的非静态值结合使用时除外。例如，以下内容目前不起作用：

```
MATCH (n)
RETURN n LIMIT toInteger(rand())    // Does NOT work!
```

- WITH – 支持，但与 SKIP 或 LIMIT 的非静态值结合使用时除外。例如，以下内容目前不起作用：

```
MATCH (n)
WITH n SKIP toInteger(rand())
WITH count() AS count
RETURN count > 0 AS nonEmpty    // Does NOT work!
```

- UNWIND
- WHERE
- ORDER BY
- SKIP
- LIMIT
- CREATE – Neptune 允许您在 CREATE 查询中创建自定义 ID 值。
- DELETE
- SET
- REMOVE
- MERGE – Neptune 支持在 MERGE 查询中使用自定义 ID 值。
- *CALL[YIELD...]* –Neptune 目前不支持。
- UNION, UNION ALL – 支持只读查询，但目前不支持突变查询。

Neptune 中对 openCypher 运算符的支持

除非另有说明，否则 Neptune 支持以下运算符：

一般运算符

- DISTINCT
- 用于访问嵌套文本映射的属性的 **.** 运算符。

数学运算

- **+** 加法运算符。
- **-** 减法运算符。
- ***** 乘法运算符。

- **/** 除法运算符。
- **%** 模除运算符。
- ### **^** 幂运算符。

比较运算符

- **=** 加法运算符。
- **<>** 不等于运算符。
- 支持 **<** 小于运算符，除非其中一个参数是路径、列表或映射。
- 支持 **>** 大于运算符，除非其中一个参数是路径、列表或映射。
- 支持 **<=** less-than-or-equal-to 运算符，除非其中一个参数是路径、列表或地图。
- 支持 **>=** greater-than-or-equal-to 运算符，除非其中一个参数是路径、列表或地图。
- `IS NULL`
- `IS NOT NULL`
- 如果要搜索的数据是字符串，则支持 `STARTS WITH`。
- 如果要搜索的数据是字符串，则支持 `ENDS WITH`。
- 如果要搜索的数据是字符串，则支持 `CONTAINS`。

布尔运算符

- `AND`
- `OR`
- `XOR`
- `NOT`

字符串运算符

- **+** 联接运算符。

列表运算符

- **+** 联接运算符。
- `IN`（检查列表中是否存在某个项目）

Neptune 中对 openCypher 表达式的支持

除非另有说明，否则 Neptune 支持以下表达式：

- CASE
- Neptune 目前不支持使用 **[]** 表达式来访问节点、关系或映射中动态计算的属性键。例如，以下内容不起作用：

```
MATCH (n)
WITH [5, n, {key: 'value'}] AS list
RETURN list[1].name
```

Neptune 中对 openCypher 函数的支持

除非另有说明，否则 Neptune 支持以下函数：

谓词函数

- exists()

标量函数

- coalesce()
- endNode()
- epochmillis()
- head()
- id()
- last()
- length()
- randomUUID()
- properties()
- removeKeyFromMap
- size() – 此重载方法目前仅适用于模式表达式、列表和字符串
- startNode()
- timestamp()

- toBoolean()
- toFloat()
- toInteger()
- type()

聚合函数

- avg()
- collect()
- count()
- max()
- min()
- percentileDisc()
- stDev()
- percentileCont()
- stDevP()
- sum()

列出函数

- join()（将列表中的字符串联接成单个字符串）
- keys()
- labels()
- nodes()
- range()
- relationships()
- reverse()
- tail()

数学函数 - 数字

- abs()

- ceil()
- floor()
- rand()
- round()
- sign()

数学函数 - 对数

- e()
- exp()
- log()
- log10()
- sqrt()

数学函数 - 三角函数

- acos()
- asin()
- atan()
- atan2()
- cos()
- cot()
- degrees()
- pi()
- radians()
- sin()
- tan()

字符串函数

- join()（将列表中的字符串联接成单个字符串）

- left()
- lTrim()
- replace()
- reverse()
- right()
- rTrim()
- split()
- substring()
- toLower()
- toString()
- toUpper()
- trim()

用户定义的函数

Neptune 中目前不支持#######。

Neptune 特定的 openCypher 实现细节

以下各节描述了 openCypher 的 Neptune 实现可能与 openCypher 规范不同或超出该规范的方式。

Neptune 中的可变长度路径 (VLP) 求值

可变长度路径 (VLP) 求值会发现图形中节点之间的路径。查询中的路径长度可以不受限制。为了防止循环，openCypher 规范规定，每个解最多只能遍历每个边缘一次。

对于 VLP，Neptune 实现与 openCypher 规范的不同之处在于，它仅支持属性相等筛选条件的常量值。执行以下查询：

```
MATCH (x)-[:route*1..2 {dist:33, code:x.name}]->(y) return x,y
```

由于 x.name 属性相等筛选条件值不是常量，因此，此查询会导致 UnsupportedOperationException，并显示消息：Property predicate over variable-length relationships with non-constant expression is not supported in this release.

Neptune OpenCypher 实现中的时间支持（海王星数据库 1.3.1.0 及更低版本）

Neptune 目前为 openCypher 中的时间函数提供有限支持。它支持对时间类型使用 `DateTime` 数据类型。

`datetime()` 函数可用于获取当前的 UTC 日期和时间，如下所示：

```
RETURN  datetime() as res
```

可以从存储在 Neptune 中的数据转换日期和时间值，如下所示：

```
MATCH (n) RETURN datetime(n.createdDate)
```

日期和时间值可以从 "dateTtime" 格式的字符串中解析，其中 date 和 time 都以下面支持的形式之一表示：

支持的日期格式

- `yyyy-MM-dd`
- `yyyyMMdd`
- `yyyy-MM`
- `yyyy-DDD`
- `yyyyDDD`
- `yyyy`

支持的时间格式

- `HH:mm:ssZ`
- `HHmmssZ`
- `HH:mm:ssZ`
- `HH:mmZ`
- `HHmmZ`
- `HHZ`
- `HHmmss`
- `HH:mm:ss`

- HH:mm
- HHmm
- HH

例如：

```
RETURN datetime('2022-01-01T00:01')      // or another example:
RETURN datetime('2022T0001')
```

请注意，Neptune openCypher 中的所有日期/时间值均作为 UTC 值存储和检索。

Neptune openCypher 使用 `statement` 时钟，这意味着在查询的整个持续时间内使用相同的时刻。同一事务中的不同查询可能会使用不同的时刻。

Neptune 不支持在对 `datetime()` 的调用中使用函数。例如，以下内容不起作用：

```
CREATE (:n {date:datetime(tostring(2021))})  // ---> NOT ALLOWED!
```

Neptune 确实支持将 `datetime` 转换为 `epochmillis` 的 `epochmillis()` 函数。例如：

```
MATCH (n) RETURN epochMillis(n.someDateTime)
1698972364782
```

Neptune 目前对 `DateTime` 对象不支持其它函数和操作，例如加法和减法。

Neptune OpenCypher 实现中的时间支持（Neptune Analytics 和 Neptune Database 1.3.2.0 及更高版本）

以下日期时间功能 OpenCypher 适用于 Neptune Analytics。或者，您可以使用 labmode 参数`DatetimeMillisecond=enabled`在 Neptune 引擎发行版 1.3.2.0 及更高版本上启用以下日期时间功能。有关在实验室模式下使用此功能的更多详细信息，请参阅扩展的日期时间支持。

- Support 以毫秒为单位。即使毫秒为 0，日期时间文字也将始终以毫秒为单位返回。（之前的行为是截断毫秒。）

```
CREATE (:event {time: datetime('2024-04-01T23:59:59Z')})

# Returning the date returns with 000 suffixed representing milliseconds
```

```
MATCH(n:event)
RETURN n.time as datetime

{
  "results" : [ {
    "n" : {
      "~id" : "0fe88f7f-a9d9-470a-bbf2-fd6dd5bf1a7d",
      "~entityType" : "node",
      "~labels" : [ "event" ],
      "~properties" : {
        "time" : "2024-04-01T23:59:59.000Z"
      }
    }
  } ]
}
```

- 支持通过存储的属性或中间结果调用 datetime () 函数。例如，在此功能之前，无法进行以下查询。

  日期时间 () 而不是属性：

```
// Create node with property 'time' stored as string
CREATE (:event {time: '2024-04-01T23:59:59Z'})

// Match and return this property as datetime
MATCH(n:event)
RETURN datetime(n.time) as datetime
```

  日期时间 () 而不是中间结果：

```
// Parse datetime from parameter
UNWIND $list as myDate
RETURN datetime(myDate) as d
```

- 现在也可以保存上述情况下创建的日期时间属性。

  将日期时间从一个属性的字符串属性保存到另一个属性：

```
// Create node with property 'time' stored as string
CREATE (:event {time: '2024-04-01T23:59:59Z', name: 'crash'})

// Match and update the same property to datetime type
MATCH(n:event {name: 'crash'})
SET n.time = datetime(n.time)
```

```
// Match and update another node's property
MATCH(e:event {name: 'crash'})
MATCH(n:server {name: e.servername})
SET n.time = datetime(e.time)
```

使用带有 datetime 属性的参数批量创建节点：

```
// Batch create from parameter
UNWIND $list as events
CREATE (n:crash) {time: datetime(events.time)}
// Parameter value
{
  "x":[
    {"time":"2024-01-01T23:59:29", "name":"crash1"},
    {"time":"2023-01-01T00:00:00Z", "name":"crash2"}
  ]
}
```

- 支持 ISO8601 日期时间格式的更大子集。请参阅下面的。

支持的格式

日期时间值的格式为 [日期] T [时间] [时区]，其中 T 是分隔符。如果未提供明确的时区，则假定 UTC (Z) 为默认时区。

时区

支持的时区格式有：

- +/-HH: mm
- +/-hhhmm
- +/-HH

日期时间字符串中是否存在时区是可选的。如果时区偏移量为 0，则可以使用 Z 代替上面的时区后缀来指示 UTC 时间。支持的时区范围为-14:00 到 + 14:00。

Date

如果不存在时区，或者时区为 UTC (Z)，则支持的日期格式如下：

> **Note**
>
> DDD 是指序数日期，它代表一年中从 001 到 365（闰年为 366）之间的某一天。例如，2024-002 代表 2024 年 1 月 2 日。

- `yyyy-MM-dd`
- `yyyyMMdd`
- `yyyy-MM`
- `yyyyMM`
- `yyyy-DDD`
- `yyyyDDD`
- `yyyy`

如果选择了 Z 以外的时区，则支持的日期格式仅限于以下几种：

- `yyyy-MM-dd`
- `yyyy-DDD`
- `yyyyDDD`

支持的日期范围为 1400-01-01 到 9999-12-31。

时间

如果不存在 timezaone，或者时区为 UTC (Z)，则支持的时间格式为：

- `HH:mm:ss.SSS`
- `HH:mm:ss`
- `HHmmss.SSS`
- `HHmmss`
- `HH:mm`
- `HHmm`
- `HH`

如果选择了 Z 以外的时区，则支持的时间格式仅限于以下几种：

- `HH:mm:ss`
- `HH:mm:ss.SSS`

Neptune openCypher 语言语义的差异

Neptune 将节点和关系 ID 表示为字符串而不是整数。ID 等于通过数据加载程序提供的 ID。如果该列有命名空间，则命名空间加上 ID。因此，`id` 函数返回的是字符串而不是整数。

`INTEGER` 数据类型限制为 64 位。使用 `TOINTEGER` 函数将较大的浮点值或字符串值转换为整数时，负值会被截断为 `LLONG_MIN`，正值被截断为 `LLONG_MAX`。

例如：

```
RETURN TOINTEGER(2^100)
>  9223372036854775807

RETURN TOINTEGER(-1 * 2^100)
>  -9223372036854775808
```

Neptune 特定的 **`join()`** 函数

Neptune 实现了一个在 openCypher 规范中不存在的 `join()` 函数。它根据字符串文本列表和字符串分隔符创建字符串文本。此函数采用两个参数：

- 第一个参数是字符串文本列表。
- 第二个参数是分隔符字符串，可以包含零个、一个或多个字符。

例如：

```
join(["abc", "def", "ghi"], ", ")    // Returns "abc, def, ghi"
```

Neptune 特定的 **`removeKeyFromMap()`** 函数

Neptune 实现了一个在 openCypher 规范中不存在的 `removeKeyFromMap()` 函数。它从映射中移除指定的键并返回生成的新映射。

此函数采用两个参数：

- 第一个参数是从中移除键的映射。
- 第二个参数是从映射中移除的键。

当您想通过展开映射列表来设置节点或关系的值时，`removeKeyFromMap()` 函数特别有用。例如：

```
UNWIND [{`~id`: 'id1', name: 'john'}, {`~id`: 'id2', name: 'jim'}] as val
CREATE (n {`~id`: val.`~id`})
SET n = removeKeyFromMap(val, '~id')
```

节点和关系属性的自定义 ID 值

从引擎版本 1.2.0.2 开始，Neptune 扩展了 openCypher 规范，以便您现在可以在 `CREATE`、`MERGE` 和 `MATCH` 子句中为节点和关系指定 `id` 值。这使您可以分配用户友好的字符串（而不是系统生成的 UUID）来识别节点和关系。

> **Warning**
>
> openCypher 规范的这一扩展不向后兼容，因为 `~id` 现在被视为保留的属性名称。如果您已经在数据和查询中将 `~id` 用作属性，则需要将现有属性迁移到新的属性键并移除旧的属性键。请参阅 如果您目前正在将 ~id 用作属性，该怎么办。

以下示例展示了如何创建具有自定义 ID 的节点和关系：

```
CREATE (n {`~id`: 'fromNode', name: 'john'})
  -[:knows {`~id`: 'john-knows->jim', since: 2020}]
  ->(m {`~id`: 'toNode', name: 'jim'})
```

如果您尝试创建已在使用的自定义 ID，Neptune 会引发 `DuplicateDataException` 错误。

以下是在 `MATCH` 子句中使用一个自定义 ID 的示例：

```
MATCH (n {`~id`: 'id1'})
RETURN n
```

以下是在 `MERGE` 子句中使用多个自定义 ID 的示例：

```
MATCH (n {name: 'john'}), (m {name: 'jim'})
```

```
MERGE (n)-[r {`~id`: 'john->jim'}]->(m)
RETURN r
```

如果您目前正在将 **~id** 用作属性，该怎么办

在引擎版本 1.2.0.2 中，openCypher 子句中的 ~id 键现在视为 id 而不是属性。这意味着，如果您有一个名为 ~id 的属性，则无法对其进行访问。

如果您使用的是 ~id 属性，那么在升级到引擎版本 1.2.0.2 或更高版本之前，您要做的就是先将现有 ~id 属性迁移到新的属性键，然后移除 ~id 属性。例如，下面的查询：

- 为所有节点创建一个名为“newId”的新属性，
- 将“~id”属性的值复制到“newId”属性中，
- 并从数据中移除“~id”属性

```
MATCH (n)
WHERE exists(n.`~id`)
SET n.newId = n.`~id`
REMOVE n.`~id`
```

对于数据中具有 ~id 属性的任何关系，都需要做同样的事情。

您还必须更改您正在使用的任何引用 ~id 属性的查询。例如，此查询：

```
MATCH (n)
WHERE n.`~id` = 'some-value'
RETURN n
```

...会改成这样：

```
MATCH (n)
WHERE n.newId = 'some-value'
RETURN n
```

Neptune openCypher 和 Cypher 之间的其它区别

- Neptune 仅对于 Bolt 协议支持 TCP 连接。 WebSockets不支持螺栓连接。
- Neptune openCypher 会移除 Unicode 在 `trim()`、`ltrim()` 和 `rtrim()` 函数中定义的空格。

- 在 Neptune openCypher 中，对于较大的双精度值，`tostring(double)` 不会自动切换到 E 表示法。
- 尽管 openCypher CREATE 不会创建多值属性，但它们可以存在于使用 Gremlin 创建的数据中。如果 Neptune openCypher 遇到多值属性，则会任意选择其中一个值，从而产生不确定的结果。

Neptune 图形数据模型

Amazon Neptune 图形数据的基本单位是一个四个位置的（四元组）元素，类似于资源描述框架 (RDF) 四元组。以下是 Neptune 四元组的四个位置：

- `subject  (S)`
- `predicate  (P)`
- `object  (O)`
- `graph  (G)`

每个四元组都是一个语句，对一个或多个资源进行断言。一个语句可以断言两个资源之间是否存在关系，或者可以将一个属性（键/值对）附加到某个资源。您可以将四元组谓词值通常视为语句的动词。它描述了正在定义的关系或属性的类型。对象是关系的目标，或者是属性的值。示例如下：

- 可以通过将源顶点标识符存储在 S 位置、将目标顶点标识符存储在 O 位置并将边缘标签存储在 P 位置来表示两个顶点之间的关系。
- 可以通过将元素标识符存储在 S 位置、将属性键存储在 P 位置并将属性值存储在 O 位置来表示属性。

图形位置 G 在不同堆栈中的使用方式不同。对于 Neptune 中的 RDF 数据，G 位置包含命名图形标识符。对于 Gremlin 中的属性图形，它用于有边缘时存储边缘 ID 值。在所有其他情况下，它默认为固定值。

一组共享资源标识符的四元组语句创建一个图形。

面向用户的值的字典

Neptune 不会将大多数面向用户的值直接存储在其维护的各种索引中。相反，它将这些值分别存储在字典中，并在索引中用 8 字节的标识符替换它们。

- 所有会进入 S、P 或 G 索引的面向用户的值都以这种方式存储在字典中。
- 在 O 索引中，数值直接存储在索引中（内联）。这包括 `date` 和 `datetime` 值（以从纪元开始的毫秒数表示）。
- 进入 O 索引中的所有其它面向用户的值都存储在字典中，并在索引中用 ID 表示。

字典包含面向用户的值与 `value_to_id` 索引中 8 字节 ID 的正向映射。

它将存储 8 字节 ID 与两个索引之一中的值的反向映射，具体取决于值的大小：

- `id_to_value` 索引将 ID 映射到在内部编码后小于 767 字节的面向用户的值。
- `id_to_blob` 索引将 ID 映射到更大的面向用户的值。

如何在 Neptune 中为语句编制索引

当您查询四元组的图形时，对于每个四元组位置，您可以指定一个值约束，也可以不指定。查询将返回与您指定的值约束相匹配的所有四元组。

Neptune 使用索引来解析查询。正如 Andreas Harth 和 Stefan Decker 在其 2005 年的论文针对从 Web 查询 RDF 优化索引结构中观察到的，对于 4 个四元组位置，有 16（2 的 4 次方）种可能的访问模式。您可以有效地查询所有 16 种模式，而无需使用 6 个四元组语句索引进行扫描和筛选。每个四元组语句索引都使用一个由以不同顺序连接的四个位置值组成的键。

```
     Access Pattern                                          Index key order
-------------------------------------------------------    ---------------
 1.  ????  (No constraints; returns every quad)                SPOG
 2.  SPOG  (Every position is constrained)                     SPOG
 3.  SPO?  (S, P, and O are constrained; G is not)             SPOG
 4.  SP??  (S and P are constrained; O and G are not)          SPOG
 5.  S???  (S is constrained; P, O, and G are not)             SPOG
 6.  S??G  (S and G are constrained; P and O are not)          SPOG

 7.  ?POG  (P, O, and G are constrained; S is not)             POGS
 8.  ?PO?  (P and O are constrained; S and G are not)          POGS
 9.  ?P??  (P is constrained; S, O, and G are not)             POGS

10.  ?P?G  (P and G are constrained; S and O are not)          GPSO
11.  SP?G  (S, P, and G are constrained; O is not)             GPSO
12.  ???G  (G is constrained; S, P, and O are not)             GPSO

13.  S?OG  (S, O, and G are constrained; P is not)             OGSP
14.  ??OG  (O and G are constrained; S and P are not)          OGSP
15.  ??O?  (O is constrained; S, P, and G are not)             OGSP

16.  S?O?  (S and O are constrained; P and G are not)          OSGP
```

默认情况下，Neptune 只创建并维护这六个索引中的三个：

- SPOG – 使用 Subject + Predicate + Object + Graph 组成的密钥。
- POGS – 使用 Predicate + Object + Graph + Subject 组成的密钥。
- GPSO – 使用 Graph + Predicate + Subject + Object 组成的密钥。

这三个索引处理许多最常见的访问模式。仅维护三个完整语句索引而不是六个会显著减少支持无需扫描和筛选的快速访问所需的资源。例如，只要绑定了位置的前缀（例如顶点或顶点和属性标识符），SPOG 索引就允许高效查找。当仅绑定了存储在 P 位置中的边缘或属性标签时，POGS 索引才允许高效访问。

用于查找语句的低级别 API 获取语句模式，此时部分位置已知，而其余位置留下由索引搜索来发现。通过根据语句索引之一的索引键顺序，将已知位置复合到键前缀，Neptune 执行范围扫描来检索与已知位置匹配的所有语句。

不过，Neptune 默认情况下没有创建的语句索引之一是反向遍历 OSGP 索引，该索引可跨对象和主题收集谓词。相反，默认情况下，Neptune 在用于执行 {all P x POGS} 联合扫描的单独索引中跟踪不同谓词。当您使用 Gremlin 时，谓词对应于属性或边缘标签。

如果图形中的不同谓词数太大，默认 Neptune 访问策略可能会效率低下。例如在 Gremlin 中，没有给出边缘标签的 in() 步骤或内部使用 in() 的任何步骤（例如 both() 或 drop()）可能会效率非常低下。

使用实验室模式启用 OSGP 索引创建

如果您的数据模型创建了大量不同的谓词，则可能会遇到性能下降和运营成本较高的问题，在 Neptune 默认维护的三个索引之外，还可以使用实验室模式来启用 OSGP 索引，从而显著改进这些问题。

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.1.0.200463.0 开始推出。

启用 OSGP 索引可能有一些缺点：

- 插入速率最多可能会降低 23%。
- 使用的存储最多会增加 20%。
- 均匀接触所有索引的读取查询（这是非常少有的情况）的延迟可能会增加。

但是，一般来说，为具有大量不同谓词的数据库集群启用 OSGP 索引是值得的。基于对象的搜索（例如，查找指向某个顶点的所有传入边缘，或者连接到指定对象的所有主题）将变得高效，因此删除顶点的效率也会更高。

> **Important**
>
> 您只能首先在空数据库集群中启用 OSGP 索引，然后再将任何数据加载到其中。

Neptune 数据模型中的 Gremlin 语句

在 SPOG 模型中，Gremlin 属性图数据使用三个语句类来表示，即：

- 顶点标签语句
- 边缘语句
- 属性语句

有关如何在 Gremlin 查询中使用它们的说明，请参阅了解 Gremlin 查询在 Neptune 中的工作方式。

Neptune 查找缓存可以加快读取查询的速度

Amazon Neptune 实现了查找缓存，该缓存使用 R5d 实例的基于 NVMe 的 SSD 来提高频繁、重复查找属性值或 RDF 文本的查询的读取性能。查找缓存会临时将这些值存储在 NVMe SSD 卷中，在那里可以快速访问它们。

从Amazon Neptune 引擎版本 1.0.4.2.R2（2021 年 6 月 1 日）开始，此特征可用。

如果需要从集群存储卷而不是内存中检索属性值或文本，则返回大量顶点和边缘或许多 RDF 三元组的属性的读取查询可能会有很高的延迟。示例包括长时间运行的读取查询，这些查询会从身份图形中返回大量全名，或者从欺诈检测图形中返回大量的 IP 地址。随着查询返回的属性值或 RDF 文本数量增加，可用内存会减少，查询执行可能会显著降级。

Neptune 查找缓存的用例

只有当读取查询返回非常大量的顶点和边缘或 RDF 三元组的属性时，查找缓存才会有所帮助。

为了优化查询性能，Amazon Neptune 使用 R5d 实例类型为此类属性值或文本创建大型缓存。这样，从缓存中检索它们要比从集群存储卷中检索它们快得多。

根据经验，只有满足以下所有三个条件时，才值得启用查找缓存：

- 您会一直在观察到读取查询的延迟时间增加。
- 在运行读取查询时，您还会发现`BufferCacheHitRatio`CloudWatch 指标有所下降（请参阅使用亚马逊监控 Neptune CloudWatch）。
- 在呈现结果之前，读取查询会花费大量时间来实体化返回值（有关确定要为查询实体化多少个属性值的方法，请参阅下面的 Gremlin-Profile 示例）。

> **Note**
>
> 此特征仅在上述特定情况下有用。例如，查找缓存根本无助于聚合查询。除非您运行的查询会受益于查找缓存，否则没有理由使用 R5d 实例类型来代替等效且成本更低的 R5 实例类型。

如果您使用的是 Gremlin，则可以使用 Gremlin `profile` API 来评测查询的实体化成本。在“索引操作”下，它显示了在执行过程中实体化的项数量：

```
Index Operations
```

```
Query execution:
    # of statement index ops: 3
    # of unique statement index ops: 3
    Duplication ratio: 1.0
    # of terms materialized: 5273
Serialization:
    # of statement index ops: 200
    # of unique statement index ops: 140
    Duplication ratio: 1.43
    # of terms materialized: 32693
```

实体化的非数字项的数量与 Neptune 必须执行的项查找数量成正比。

使用查找缓存

查找缓存仅在 R5d 实例类型上可用，默认情况下会自动启用。Neptune R5d 实例具有与 R5 实例相同的规格，外加高达 1.8TB 的基于 NVMe 的本地 SSD 存储。查找缓存特定于实例，受益的工作负载可以专门定向到 Neptune 集群中的 R5d 实例，而其它工作负载可以定向到 R5 或其它实例类型。

要在 Neptune 实例上使用查找缓存，只需将该实例升级为 R5d 实例类型即可。当您这样做时，Neptune 会自动将 neptune_lookup_cache 数据库集群参数设置为 `'enabled'`，并在该特定实例上创建查找缓存。然后，您可以使用 实例状态 API 确认缓存已启用。

同样，要在给定实例上禁用查找缓存，请将该实例从 R5d 实例类型缩减到等效的 R5 实例类型。

启动 R5d 实例时，查找缓存处于启用状态并处于冷启动模式，这意味着它是空的。Neptune 在处理查询时首先在查找缓存中检查属性值或 RDF 文本，如果它们尚不存在，则添加它们。这会逐渐预热缓存。

当您将需要属性值或 RDF 文本查找的读取查询定向到 R5d 读取器实例时，在缓存预热时读取性能会略有下降。但是，当缓存预热后，读取性能会显著提高，并且您还可能会看到与查找命中缓存而非集群存储相关的 I/O 成本下降。内存利用率也有所提高。

如果您的写入器实例是 R5d，则它会在每次写入操作时自动预热其查找缓存。这种方法确实会稍微增加写入查询的延迟，但可以更高效地预热查找缓存。然后，如果您将需要属性值或 RDF 文本查找的读取查询定向到写入器实例，则可以立即开始获得改进的读取性能，因为已在此处缓存了这些值。

此外，如果您在 R5d 写入器实例上运行批量加载程序，则可能会注意到由于缓存的原因，其性能略有下降。

由于查找缓存特定于每个节点，因此主机替换会将缓存重置为冷启动。

通过将 neptune_lookup_cache 数据库集群参数设置为 'disabled'，您可以暂时禁用数据库集群中所有实例上的查找缓存。但是，一般而言，通过将特定实例从 R5d 缩减到 R5 实例类型来禁用这些实例上的缓存更有意义。

Neptune 中的事务语义

Amazon Neptune 旨在支持针对数据图形的高度并发的在线事务处理 (OLTP) 工作负载。适用于 RDF 的 W3C SPARQL 查询语言规范和 Apache TinkerPop Gremlin Graph Traversal Language 文档并未定义并发查询处理的事务语义。由于 ACID 支持和良好定义的事务保证非常重要，因此，我们实施了严格的语义来帮助避免数据异常。

本部分定义这些语义，并介绍它们如何应用于 Neptune 中的一些常用案例。

主题

隔离级别定义

ACID 中的“I”代表隔离。事务的隔离程度决定了其他并发事务对其所操作的数据的影响程度。

SQL:1992 标准制定了用于描述隔离级别的词汇表。它定义了两个并发事务 Tx1 和 Tx2 之间可能发生的三种类型的交互（称为现象）：

- Dirty read – 当 Tx1 修改某个项目，然后 Tx2 在 Tx1 提交更改前读取该项目时，将发生这种情况。之后，如果 Tx1 始终未成功提交该更改或将其回滚，则 Tx2 读取的值从未进入数据库。
- Non-repeatable read – 当 Tx1 读取某个项目，然后 Tx2 修改或删除该项目并提交更改，随后 Tx1 尝试重新读取该项目时，将发生这种情况。此时，Tx1 将读取到与以前不同的值，或发现该项目已不存在。
- Phantom read – 当 Tx1 读取一组满足搜索条件的项目，然后 Tx2 添加一个满足搜索条件的新项目，随后 Tx1 重复该搜索时，将发生这种情况。此时，Tx1 将获得与之前不同的项目集。

这三种交互类型中的任何一种都可能导致数据库中的最终数据不一致。

SQL:1992 标准定义了四个隔离级别，这些隔离级别在三种交互类型及其可能产生的不一致方面具有不同的保证。在所有四个级别上，都可以保证事务完整执行或根本不执行：

- READ UNCOMMITTED – 允许所有三种交互类型（即脏读、不可重复读以及幻读）。
- READ COMMITTED – 不可能出现脏读，但可能出现不可重复读和幻读。

- `REPEATABLE READ` – 不可能出现脏读和不可重复读，但仍可能出现幻读。
- `SERIALIZABLE` – 三种交互现象均不会发生。

多版本并发控制 (MVCC) 允许另一种隔离，即快照隔离。这保证事务对事务开始时就存在的数据快照进行操作，并且没有任何其他事务可以更改该快照。

Neptune 中的事务隔离级别

Amazon Neptune 为只读查询和突变查询实现了不同的事务隔离级别。基于以下标准，SPARQL 和 Gremlin 查询被划分为只读查询或更改查询：

- 在 SPARQL 中，读取查询（`SELECT`、`ASK`、`CONSTRUCT` 和 `DESCRIBE`，如 SPARQL 1.1 查询语言规范中所定义）和突变查询（`INSERT` 和 `DELETE`，如 SPARQL 1.1 更新规范中所定义）之间有明显的区别。

  请注意，Neptune 将一起提交的多个突变查询（例如，在 `POST` 消息中，以分号分隔）视为单个事务。它们作为原子单位保证成功或失败，在失败的情况下，会回滚部分更改。
- 但是，在 Gremlin 中，Neptune 根据查询是否包含操纵数据的任何查询路径步骤（例如 `addE()`、`addV()`、`property()` 或 `drop()`）将查询分类为只读查询或突变查询。如果查询包含任何此类路径步骤，则将其分类为更改查询并执行。

还可以在 Gremlin 中使用长期会话。有关更多信息，请参阅 基于 Gremlin 脚本的会话。在这些会话中，所有查询（包括只读查询）都是在与写入器端点上的突变查询相同的隔离条件下执行的。

在 openCypher 中使用 bolt 读写会话，所有查询（包括只读查询）都是在与突变查询相同的隔离条件下在写入器端点上执行的。

主题

- Neptune 中的只读查询隔离
- Neptune 中的突变查询隔离
- 使用锁定等待超时解决冲突
- 范围锁定和虚假冲突

Neptune 中的只读查询隔离

Neptune 根据快照隔离语义计算只读查询。也就是说，只读查询以逻辑方式对在查询评估开始时拍摄的数据库一致性快照进行操作。然后，Neptune 可以保证不会发生以下任何现象：

- `Dirty reads` – Neptune 中的只读查询绝不会看到并发事务中未提交的数据。
- `Non-repeatable reads` – 多次读取相同数据的只读事务将始终返回相同的值。
- `Phantom reads` – 只读事务绝不会读取到在该事务开始后添加的数据。

由于快照隔离是使用多版本并发控制 (MVCC) 实现的，只读查询不需要锁定数据，因此不会阻止更改查询。

只读副本仅接受只读查询，因此所有针对只读副本的查询均按照 SNAPSHOT 隔离语义执行。

查询只读副本时，唯一需要考虑的其他问题是，写入副本和只读副本之间可能会有较小的复制滞后。这意味着对写入器进行的更新可能需要一个较短的时间才能传播到您正在读取的只读副本。实际复制时间取决于针对主实例的写入负载。Neptune 架构支持低延迟复制，复制延迟是在 Amazon 指标中进行衡量的。 CloudWatch

但是，由于隔离级别为 SNAPSHOT，读取查询看到的始终是数据库的一致性状态（即使不是最新的状态）。

如果需要强力保证查询看到的是之前更新的结果，请将查询发送到写入器终端节点本身而不是只读副本。

Neptune 中的突变查询隔离

在更改查询中进行的读取按照 `READ COMMITTED` 事务隔离执行，这排除了脏读的可能性。除了为 `READ COMMITTED` 事务隔离提供通常的保证之外，Neptune 还提供不会发生 `NON-REPEATABLE` 或 `PHANTOM` 读取的强力保证。

这些强力保证是通过在读取数据时锁定记录和记录范围实现的。这可防止并发事务在被读取后的索引范围内进行插入或删除，从而保证可重复读取。

> **Note**
>
> 但是，并发更改事务 `Tx2` 可在更改事务 `Tx1` 开始后开始，并且可在 `Tx1` 锁定并读取数据前提交更改。在这种情况下，`Tx1` 将看到 `Tx2` 的更改，就像 `Tx2` 在 `Tx1` 开始之前已经完成一样。由于这仅适用于已提交的更改，因此绝不会发生 `dirty read`。

要了解 Neptune 用于突变查询的锁定机制，首先了解 Neptune 图形数据模型和索引策略的细节会有所帮助。Neptune 使用三个索引来管理数据，即 SPOG、POGS 和 GPSO。

为了实现 READ COMMITTED 事务级别的可重复读取，Neptune 将对正在使用的索引进行范围锁定。例如，如果更改查询读取名为 person1 的顶点的所有属性和出边，则该节点将在读取数据前锁定由 SPOG 索引中的 S=person1 前缀定义的整个范围。

使用其他索引时，将应用相同的机制。例如，当更改事务使用 POGS 索引在所有源-目标顶点对中查找给定边缘标签时，将锁定 P 位置中该边缘标签的范围。任何并发事务，不管是只读查询还是更改查询，都仍可在锁定范围内执行读取。但是，涉及在锁定的前缀范围内插入或删除新记录的任何更改都需要排他锁，并且将被阻止。

换句话说，当更改事务已读取索引范围时，可以强力保证在该读取事务结束之前，任何并发事务都不会修改该范围。这可保证不会出现 non-repeatable reads。

使用锁定等待超时解决冲突

如果第二个事务试图在第一个事务已锁定的范围内修改记录，Neptune 会立即检测到该冲突并阻止第二个事务。

如果未检测到依赖性死锁，Neptune 将自动应用锁定等待超时机制，被阻止的事务等待最多 60 秒，以便持有该锁的事务完成并释放锁。

- 如果锁定等待超时在释放锁前到期，则回滚被阻止的事务。
- 如果在锁定等待超时之内释放了锁，则第二个事务将被解除阻止，并且可以成功完成而无需重试。

但是，如果 Neptune 检测到两个事务之间存在依赖性死锁，则无法自动协调冲突。在这种情况下，Neptune 将立即取消并回滚这两个事务之一，而不会启动锁定等待超时。Neptune 会尽最大努力回滚插入或删除的记录数最少的事务。

范围锁定和虚假冲突

Neptune 使用间隙锁定来进行范围锁定。间隙锁定是对索引记录之间间隙的锁定，或者是对第一条索引记录之前或最后一条索引记录之后间隙的锁定。

Neptune 使用所谓的字典表将数字 ID 值与特定的字符串文本关联起来。以下是此类 Neptune 字典的示例状态：表：

String	ID
type	1

String	ID
default_graph	2
person_3	3
person_1	5
knows	6
person_2	7
age	8
edge_1	9
lives_in	10
New York	11
人员	12
Place	13
edge_2	14

上面的字符串属于属性图模型，但这些概念同样适用于所有 RDF 图形模型。

SPOG (Subject-Predicate-Object_Graph) 索引的相应状态如下图左侧所示。右侧显示了相应的字符串，以帮助理解索引数据的含义。

S (ID)	P (ID)	O (ID)	G (ID)		S（字符串）	P（字符串）	O（字符串）	G（字符串）
3	1	12	2		person_3	type	人员	default_graph
5	1	12	2		person_1	type	人员	default_graph

S (ID)	P (ID)	O (ID)	G (ID)		S（字符串）	P（字符串）	O（字符串）	G（字符串）
5	6	3	9		person_1	knows	person_3	edge_1
5	8	40	2		person_1	age	40	default_graph
5	10	11	14		person_1	lives_in	New York	edge_2
7	1	12	2		person_2	type	人员	default_graph
11	1	13	2		New York	type	Place	default_graph

现在，如果突变查询读取名为 `person_1` 的顶点的所有属性和外出边缘，则该节点将在读取数据前锁定由 SPOG 索引中的 `S=person_1` 前缀定义的整个范围。范围锁定将在所有匹配的记录和第一条不匹配的记录上设置间隙锁定。匹配的记录将被锁定，不匹配的记录不会被锁定。Neptune 会按如下方式放置间隙锁定：

- `5 1 12 2` （间隙 1）
- `5 6 3 9` （间隙 2）
- `5 8 40 2` （间隙 3）
- `5 10 11 14` （间隙 4）
- `7 1 12 2` （间隙 5）

这将锁定以下记录：

- `5 1 12 2`
- `5 6 3 9`
- `5 8 40 2`
- `5 10 11 14`

在此状态下，以下操作被合理阻止：

- 为 S=person_1 插入新的属性或边缘。不同于 type 或新边缘的新属性必须进入间隙 2、间隙 3、间隙 4 或间隙 5，所有这些都被锁定。
- 删除任何现有记录。

同时，一些并发操作会被错误地阻止（生成虚假冲突）：

- S=person_3 的任何属性或边缘插入都会被阻止，因为它们必须进入间隙 1。
- 任何分配了一个 3 到 5 之间的 ID 的新顶点插入都将被阻止，因为它必须进入间隙 1。
- 任何分配了一个 5 到 7 之间的 ID 的新顶点插入都将被阻止，因为它必须进入间隙 5。

间隙锁定不够精确，无法锁定一个特定谓词的间隙（例如，锁定谓词 S=5 的 gap5）。

范围锁定只放在读取发生的索引中。在上述情况下，记录仅锁定在 SPOG 索引中，而不锁定在 POGS 或 GPSO 中。根据访问模式，可以对所有索引执行查询读取，访问模式可以使用 explain API 列出（适用于 Sparql 和 Gremlin）。

> **Note**
>
> 也可以使用间隙锁定来安全地并发更新底层索引，这也可能导致虚假冲突。这些间隙锁定的放置与隔离级别或事务执行的读取操作无关。

虚假冲突不仅发生在并发事务由于间隙锁定而发生冲突时，而且在某些情况下，当事务在任何类型的失败后重试时，也会发生。如果失败所触发的回滚仍在进行中，并且之前为该事务采取的锁定尚未完全释放，则重试将遇到虚假冲突并失败。

在高负载下，您通常会发现 3-4% 的写入查询由于虚假冲突而失败。对于外部客户端，此类虚假冲突很难预测，应使用重试来处理。

Neptune 事务语义示例

以下示例说明了 Amazon Neptune 中事务语义的不同用例。

主题

- 示例 1 – 仅在不存在属性时插入属性
- 示例 2 – 断言某个属性值是全局唯一的

示例 1 – 仅在不存在属性时插入属性

假设您要确保某个属性仅设置一次。例如，假设有多个查询同时尝试为某人指定信用评分。您只希望插入该属性的一个实例，而其他查询因该属性已设置而失败。

```
# GREMLIN:
g.V('person1').hasLabel('Person').coalesce(has('creditScore'), property('creditScore',
 'AAA+'))

# SPARQL:
INSERT { :person1 :creditScore "AAA+" .}
WHERE  { :person1 rdf:type :Person .
         FILTER NOT EXISTS { :person1 :creditScore ?o .} }
```

Gremlin `property()` 步骤插入具有给定键和值的属性。`coalesce()` 步骤在第一步中执行第一个参数，如果失败，则执行第二步：

在为给定 `person1` 顶点的 `creditScore` 属性插入值之前，事务必须尝试读取 `person1` 可能不存在的 `creditScore` 值。此尝试的读取将锁定 SPOG 索引中 `S=person1` 和 `P=creditScore` 的 SP 范围，其中 `creditScore` 值要么存在，要么会被写入。

进行该范围锁定可防止任何并发事务同时插入 `creditScore` 值。当存在多个并行事务时，某一时刻它们中只有一个可以更新该值。这可排除创建多个 `creditScore` 属性的异常。

示例 2 – 断言某个属性值是全局唯一的

假设您要插入一个使用社会保险号作为主键的人员。您希望更改查询，以确保在全局范围内，数据库中没有其他人具有相同的社会保险号：

```
# GREMLIN:
g.V().has('ssn', 123456789).fold()
  .coalesce(__.unfold(),
            __.addV('Person').property('name', 'John Doe').property('ssn', 123456789'))

# SPARQL:
```

```
INSERT { :person1 rdf:type :Person .
         :person1 :name "John Doe" .
         :person1 :ssn 123456789 .}
WHERE  { FILTER NOT EXISTS { ?person :ssn 123456789 } }
```

该示例与上一个示例相似。主要区别在于范围锁定是在 POGS 索引而不是 SPOG 索引上进行的。

执行查询的事务必须读取模式 ?person :ssn 123456789，其中绑定了 P 和 O 位置。范围锁定是在 P=ssn 和 O=123456789 的 POGS 索引上进行的。

- 如果存在该模式，则不采取任何操作。
- 如果不存在该模式，锁将阻止任何并发事务也插入该社会保险号

示例 3 – 如果其它属性具有指定值，则更改属性

假设游戏中的各种事件将一个人从第一关移动到第二关，并为他们分配一个设置为零的新 level2Score 属性。您需要确保此类事务的多个并发实例无法创建第二关得分属性的多个实例。Gremlin 和 SPARQL 中的查询可能如下所示。

```
# GREMLIN:
g.V('person1').hasLabel('Person').has('level', 1)
 .property('level2Score', 0)
 .property(Cardinality.single, 'level', 2)

# SPARQL:
DELETE { :person1 :level 1 .}
INSERT { :person1 :level2Score 0 .
         :person1 :level 2 .}
WHERE  { :person1 rdf:type :Person .
         :person1 :level 1 .}
```

在 Gremlin 中，指定了 Cardinality.single 时，property() 步骤将添加新属性，或将现有属性值替换为指定的新值。

对属性值的任何更新（例如将 level 从 1 增加到 2）都实现为删除当前记录并插入具有新属性值的新记录。在这种情况下，将删除第 1 关的记录，并重新插入第 2 关的记录。

在增加 level2Score 并将 level 从 1 更新为 2 之前，事务必须先验证 level 值当前等于 1。为此，它需要对 SPOG 索引中的 S=person1、P=level 和 O=1 的 SPO 前缀进行范围锁定。该锁可防止并发事务删除版本 1 三元组，因此，不会发生冲突性的并发更新。

示例 4 – 替换现有属性

某些事件可能会将某人的信用评分更新为新值（此处为 BBB）。但是，您想要确保这种类型的并发事件无法为某人创建多个信用评分属性。

```
# GREMLIN:
g.V('person1').hasLabel('Person')
 .sideEffect(properties('creditScore').drop())
 .property('creditScore', 'BBB')

# SPARQL:
DELETE { :person1 :creditScore ?o .}
INSERT { :person1 :creditScore "BBB" .}
WHERE  { :person1 rdf:type :Person .
         :person1 :creditScore ?o .}
```

这种情况与示例 3 相似，不同之处在于，Neptune 不是锁定 SPO 前缀，而是仅使用 S=person1 和 P=creditScore 来锁定 SP 前缀。这可防止并发事务插入或删除 person1 对象具有 creditScore 属性的任何三元组。

示例 5 – 避免悬垂属性或边缘

对实体的更新不应造成悬垂元素，即与无类型的实体相关联的属性或边缘。仅 SPARQL 存在该问题；Gremlin 具有内置约束，可避免造成悬垂元素。

```
# SPARQL:
tx1: INSERT { :person1 :age 23 } WHERE { :person1 rdf:type :Person }
tx2: DELETE { :person1 ?p ?o }
```

INSERT 查询必须读取并使用 SPOG 索引中的 S=person1、P=rdf:type 和 O=Person 锁定 SPO 前缀。该锁可防止 DELETE 查询并行成功。

在 DELETE 查询尝试删除 :person1 rdf:type :Person 记录和 INSERT 查询读取该记录并在 SPOG 索引中该记录的 SPO 上创建范围锁的争用过程中，可能产生以下结果：

- 如果 INSERT 查询在 DELETE 查询读取并删除 :person1 的所有记录之前提交，则将从数据库中完全删除 :person1，包括新插入的记录。
- 如果 DELETE 查询在 INSERT 查询尝试读取 :person1 rdf:type :Person 记录之前提交，则读取内容将包括已提交的更改。也就是说，它找不到任何 :person1 rdf:type :Person 记录，因此成为一个空操作。

- 如果 INSERT 查询在 DELETE 查询之前读取，则 `:person1 rdf:type :Person` 三元组将被锁定，并且 DELETE 查询将被阻止，直到 INSERT 查询提交为止，就像前面的第一种情况一样。
- 如果 DELETE 在 INSERT 查询之前读取，并且 INSERT 查询尝试读取并锁定该记录的 SPO 前缀，则会检测到冲突。这是因为三元组已标记为等待删除，因此 INSERT 将失败。

在上述所有不同的可能事件序列中，均未创建任何悬垂边缘。

异常处理和重试

当由于无法解决的冲突或锁定等待超时而取消事务时，Amazon Neptune 将使用 `ConcurrentModificationException` 进行响应。有关更多信息，请参阅 引擎错误代码。作为最佳实践，客户端应始终捕获并处理这些异常。

在许多情况下，当 `ConcurrentModificationException` 实例数量很少时，基于指数回退的重试机制能够很好地处理它们。在这种重试方法中，最大重试次数和等待时间通常取决于事务的最大大小和持续时间。

但是，如果您的应用程序具有很高的并发更新工作负载，并且您观察到大量 `ConcurrentModificationException` 事件，则可以修改应用程序来减少冲突性并发修改的数量。

例如，请考虑这样的应用程序：它经常对一组顶点进行更新，并使用多个并发线程进行更新来优化写入吞吐量。如果每个线程连续执行更新一个或多个节点属性的查询，则并发更新同一节点可能会产生 `ConcurrentModificationException`。这反过来会降低写入性能。

如果您能够将可能相互冲突的更新更改为顺序执行，则可大大降低发生此类冲突的可能性。例如，如果您可以确保对给定节点的所有更新查询都在同一线程上进行（例如使用基于哈希的分配），则可以确保它们一个接一个地执行，而不是并发执行。虽然对相邻节点进行范围锁定仍然可能导致 `ConcurrentModificationException`，但是您消除了对同一节点的并发更新。

Amazon Neptune 数据库集群和实例

Amazon Neptune 数据库集群通过查询管理对数据的访问。集群包括：

- 一个主数据库实例。
- 最多 15 个只读副本数据库实例。

集群中的所有实例共享相同的底层托管式存储卷，该存储卷专为实现可靠性和高可用性而设计。

您可以通过 Neptune 端点连接到数据库集群中的数据库实例。

Neptune 数据库集群中的主数据库实例

主数据库实例协调针对数据库集群底层存储卷的所有写入操作。它还支持读取操作。

一个 Neptune 数据库集群中只能有一个主数据库实例。如果主实例变得不可用，Neptune 会自动失效转移到其中一个具有您可以指定的优先级的只读副本实例。

Neptune 数据库集群中的只读副本数据库实例

在为数据库集群创建主实例后，您可以在数据库集群中创建最多 15 个只读副本实例，以便为只读查询提供支持。

Neptune 只读副本数据库实例非常适用于扩展读取容量，因为它们完全专用于集群卷上的读取操作。所有写入操作均由主实例进行管理。每个只读副本数据库实例都有自己的端点。

由于集群存储卷由集群中的所有实例共享，因此，所有只读副本实例都会返回相同的查询结果数据，而复制滞后很小。此滞后通常远远少于主实例写入更新后的 100 毫秒，但当写入操作量非常大时，此滞后可能更长一些。

在不同的可用区中提供一个或多个只读副本实例可以提高可用性，因为只读副本充当主实例的失效转移目标。也就是说，如果主实例失败，Neptune 将只读副本实例提升为主实例。当这种情况发生时，在提升的实例重启时会出现短暂的中断，在此期间，对主实例的读写请求将失败，同时引发异常。

相比之下，如果您的数据库集群不包含任何只读副本实例，则当主实例出现故障时，您的数据库集群将保持不可用状态，直到重新创建该实例。与提升只读副本相比，重新创建主实例所需的时间要长得多。

为确保高可用性，我们建议您创建一个或多个只读副本实例，这些实例的数据库实例类与主实例相同，并且与主实例位于不同的可用区。请参阅 Neptune 数据库集群的容错能力。

使用控制台，您只需在创建数据库集群时指定多可用区，即可创建多可用区部署。如果数据库集群位于单个可用区中，您可以通过在不同可用区中添加 Neptune 副本来使其成为多可用区数据库集群。

> **Note**
>
> 您无法为未加密的 Neptune 数据库集群创建加密的只读副本实例，也无法为加密的 Neptune 数据库集群创建未加密的只读副本实例。

有关如何创建 Neptune 只读副本数据库实例的详细信息，请参阅使用控制台创建 Neptune 读取器实例。

调整 Neptune 数据库集群中数据库实例的大小

根据 CPU 和内存要求调整 Neptune 数据库集群中实例的大小。实例上的 vCPU 数量决定了处理传入查询的查询线程的数量。实例上的内存量决定了缓冲区缓存的大小，缓冲区缓存用于存储从底层存储卷提取的数据页的副本。

每个 Neptune 数据库实例的查询线程数等于该实例上 vCPU 数量的两倍。例如，具有 16 个 vCPU 的 `r5.4xlarge` 有 32 个查询线程，因此可以同时处理 32 个查询。

在所有查询线程都被占用时到达的其它查询将放入服务器端队列，并在查询线程变为可用时以 FIFO 方式进行处理。此服务器端队列可以容纳大约 8000 个待处理请求。装满后，Neptune 会用 `ThrottlingException` 来响应其它请求。您可以使用`MainRequestQueuePendingRequests` CloudWatch 指标监控待处理请求的数量，也可以使用带参数的 Gremlin 查询状态端点。`includeWaiting`

从客户端的角度来看，查询执行时间除了实际执行查询所花费的时间外，还包括在队列中花费的任何时间。

理想情况下，利用主数据库实例上所有查询线程的持续并发写入负载会显示 90% 或更高的 CPU 利用率，这表明服务器上的所有查询线程都积极参与执行有用的工作。但是，即使在持续并发写入负载下，实际的 CPU 利用率也通常会稍低一些。这通常是因为查询线程正在等待针对底层存储卷的 I/O 操作完成。Neptune 使用仲裁写入，这种写入在三个可用区中生成六份数据副本，而这六个存储节点中有四个必须确认写入才能被视为持久。当查询线程从存储卷等待此仲裁时，它会被搁置，从而降低 CPU 利用率。

如果您有一个串行写入负载，即您正在执行一个接一个的写入，并等待第一个写入完成后再开始下一个写入，则可以预计 CPU 使用率会更低。确切的数量将是 vCPU 和查询线程数量的函数（查询线程越多，每个查询的总 CPU 量就越少），等待 I/O 会导致一定程度减少。

有关如何最好地调整数据库实例大小的更多信息，请参阅选择正确的 Neptune 数据库实例类型。有关每种实例类型的定价，请参阅 Neptune 定价页面。

在 Neptune 中监控数据库实例性能

您可以使用 Neptune 中的 CloudWatch 指标来监控数据库实例的性能并跟踪客户端观察到的查询延迟。请参阅 CloudWatch 用于监控 Neptune 中的数据库实例性能。

Amazon Neptune 存储、可靠性和可用性

Amazon Neptune 使用分布式共享存储架构，这一架构可随着数据库存储需求增长而自动扩展。

Neptune 数据存储在集群卷中，该集群卷是使用基于非易失性存储规范 (NVMe) SSD 驱动器的单个虚拟卷。集群卷由一组称为分段的逻辑块组成。每个分段都分配了 10GB 的存储空间。每个分段中的数据复制到六个副本中，然后分配给数据库集群所在 AWS 区域中的三个可用区 (AZ)。

创建 Neptune 数据库集群时，会为其分配单个 10GB 的分段。随着数据量增加并超过当前分配的存储空间，Neptune 通过添加新的分段来自动扩展集群容量。在除中国以外的所有支持区域，Neptune 集群的最大容量可以增长到 128 TiB (TiB)，中国除外，该集群的容量限制在 6 GovCloud 4 TiB 以内。但是，对于早于版本：1.0.2.2（2020 年 3 月 9 日）的引擎版本，所有区域中的集群卷大小限制为 64TiB。

数据库集群卷包含所有用户数据、索引和字典（如Neptune 图形数据模型部分所述）以及内部元数据，例如内部事务日志。所有这些图形数据，包括索引和内部日志，不能超过集群卷的最大大小。

I/O 优化存储选项

Neptune 提供两种存储定价模式：

- 标准存储 - 标准存储为 I/O 使用率为中到低的应用程序提供经济实惠的数据库存储。
- I/O 优化存储 - 使用 I/O 优化存储，您只需为正在使用的存储付费，成本高于标准存储，而且无需为所使用的 I/O 支付任何费用。

  I/O 优化存储符合 I/O 密集型图形工作负载的需求，且成本可预测，I/O 延迟低，I/O 吞吐量一致。

  有关更多信息，请参阅 I/O 优化型存储。

Neptune 存储分配

即使 Neptune 集群卷可以增长到 128TiB（或在少数区域可增长到 64TiB），您也只需为实际分配的空间付费。分配的总空间由存储高水位决定，这是集群卷在它存在期间的任何时候分配给集群卷的最大容量。

这意味着，即使从集群卷中移除了用户数据（例如通过 `g.V().drop()` 等此类删除查询），分配的总空间仍保持不变。Neptune 确实会自动优化未使用的分配空间，以供将来重复使用。

除用户数据外，另外两种类型的内容会占用内部存储空间，即字典数据和内部事务日志。尽管字典数据与图形数据一起存储，但即使它支持的图形数据已被删除，它也会无限期地保存，这意味着如果重新引

入数据，条目可以重复使用。内部日志数据存储在单独的内部存储空间中，该存储空间具有自己的高水位。内部日志过期后，其占用的存储空间可以重新用于其它日志，但不能用于图形数据。为日志分配的内部空间量包含在该VolumeBytesUsedCloudWatch 指标报告的总空间中。

查看存储最佳实践以了解如何将分配的存储空间保持在最低限度并重用空间。

Neptune 存储账单

如上所述，存储费用根据存储高水位进行计费。尽管数据复制成六个副本，但您只需为一个数据副本付费。

您可以通过监控VolumeBytesUsed CloudWatch 指标来确定数据库集群当前的存储最高水位是多少（请参阅使用亚马逊监控 Neptune CloudWatch）。

可能影响 Neptune 存储成本的其它因素包括数据库快照和备份，它们作为备份存储单独计费，并基于 Neptune 存储成本（请参阅用于管理 Neptune 备份存储的 CloudWatch 指标）。

但是，如果您创建数据库的克隆，则该克隆指向的集群卷与数据库集群本身使用的集群卷相同，因此对于原始数据没有额外的存储费用。对克隆的后续更改将使用该copy-on-write 协议，并且确实会导致额外的存储成本。

有关 Neptune 定价的更多信息，请参阅 Amazon Neptune 定价。

Neptune 存储最佳实践

由于某些类型的数据会消耗 Neptune 中的永久存储空间，因此请使用以下最佳做法来避免存储增长大幅激增：

- 在设计图形数据模型时，请尽可能避免使用本质上是临时性的属性键和面向用户的值。
- 如果您计划更改数据模型，则在使用快速重置 API 清除现有数据库集群中的数据之前，不要使用新模型将数据加载到该数据库集群中。最好的办法通常是将使用新模型的数据加载到新的数据库集群上。
- 对大量数据进行操作的事务会生成相应大量的内部日志，这可能会永久增加内部日志空间的高水位。例如，删除数据库集群中所有数据的单个事务可能会生成庞大的内部日志，这需要分配大量内部存储空间，从而永久减少图形数据的可用空间。

  为避免这种情况，请将大型事务拆分为较小的事务并在两者之间留出时间，以便关联的内部日志有机会过期并释放其内部存储以供后续日志重用。

- 为了监控 Neptune 集群容量的增长，您可以对该指标设置 CloudWatch 警报。`VolumeBytesUsed` CloudWatch 如果数据已达到集群卷的最大大小，则此功能将特别有用。有关更多信息，请参阅使用 Amazon CloudWatch 警报。

当您有大量未使用的已分配空间时，缩小数据库集群使用的存储空间的唯一方法是导出图形中的所有数据，然后将其重新加载到新的数据库集群中。有关从数据库集群导出数据的简便方法，请参阅 Neptune 的数据导出服务和实用程序；有关将数据导入回 Neptune 的简单方法，请参阅 Neptune 的批量加载程序。

> **Note**
>
> 创建和还原快照不会减少分配给数据库集群的存储量，因为快照会保留集群底层存储的原始映像。如果大量已分配的存储未被使用，则缩小已分配存储量的唯一方法是导出图形数据并将其重新加载到新的数据库集群中。

Neptune 存储可靠性和高可用性

Amazon Neptune 的设计具有可靠、持久和容错的特点。

Neptune 数据的六个副本跨三个可用区 (AZ) 进行维护，这一事实确保了数据的存储非常耐用，而且数据丢失的可能性非常低。无论可用区中是否有数据库实例，数据都会跨可用区自动复制，并且复制的数量与集群中的数据库实例数量无关。

这意味着您可以快速添加只读副本，因为 Neptune 不会创建图形数据的新副本。相反，只读副本连接到已包含您的数据的集群卷。同样，移除只读副本不会移除任何底层数据。

只有在删除集群卷的所有数据库实例后，才能删除集群卷及其数据。

Neptune 还会自动检测构成集群卷的分段中的故障。当分段中的数据副本损坏时，Neptune 立即修复该分段，同时使用同一分段中的其它数据副本来确保修复的数据是最新的。因此，Neptune 避免了数据丢失，并减少了从磁盘故障中 point-in-time 恢复所需的执行恢复。

连接到 Amazon Neptune 端点

Amazon Neptune 使用数据库实例的集群而不是单个实例。每个 Neptune 连接均由特定的数据库实例处理。在连接到 Neptune 集群时，您指定的主机名和端口将指向名为端点的中间处理程序。端点是包含主机地址和端口的 URL。Neptune 端点使用加密的传输层安全性协议/安全套接字层 (TLS/SSL) 连接。

Neptune 使用端点机制来抽象这些连接，以便您不必对主机名进行硬编码，也不必在某些数据库实例不可用时编写自己的逻辑来重新路由连接。

通过使用端点，您可以根据用例将每个连接映射到相应的实例或实例组。自定义端点允许您连接到数据库实例的子集。Neptune 数据库集群中提供以下端点：

Neptune 集群端点

集群端点是 Neptune 数据库集群的一个端点，连接到该数据库集群的当前主数据库实例。每个 Neptune 数据库集群都具有集群端点和一个主数据库实例。

集群终端节点为数据库集群的读取/写入连接提供故障转移支持。对数据库集群上的所有写入操作使用集群终端节点，这些操作包括插入、更新、删除和数据定义语言 (DDL) 更改。您还可以对读取操作（如查询）使用集群端点。

如果数据库集群的当前主数据库实例失败，Neptune 将自动失效转移到新的主数据库实例。在故障转移期间，数据库集群将继续为从新的主数据库实例到集群终端节点的请求提供服务，对服务造成的中断最少。

以下示例介绍 Neptune 数据库集群中的集群端点。

```
mydbcluster.cluster-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune 读取器端点

读取器端点是 Neptune 数据库集群的一个端点，连接到该数据库集群的可用 Neptune 副本之一。每个 Neptune 数据库集群都具有一个读取器端点。如果有多个 Neptune 副本，则读取器端点会将每个连接请求定向到 Neptune 副本之一。

读取器终端节点为数据库集群的只读连接提供轮询路由。对读取操作（如 查询）使用读取器端点。

除非您拥有单实例集群（没有只读副本的集群），否则您无法将读取器终端节点用于写入操作。当且仅当在这种情况下，读取器可以用于写入操作以及读取操作。

读取器终端节点轮询路由的运行方式是更改 DNS 条目指向的主机。每次解析 DNS 时，您会获得不同的 IP 并且针对这些 IP 打开连接。建立连接之后，对于该连接的所有请求将发送到同一个主机。客户端必须创建新连接并再次解析 DNS 记录，以获取到可能不同的只读副本的连接。

> **Note**
>
> WebSockets 连接通常会长时间保存。要获取不同的只读副本，请执行以下操作：
>
> - 确保您的客户端在每次连接时都会解析 DNS 条目。
> - 关闭连接，然后重新连接。

不同的客户端软件可能在解析 DNS 的方式上各有不同。例如，如果您的客户端解析 DNS，然后对于每个连接使用该 IP，则它会将所有请求定向到单个主机。

客户端或代理的 DNS 缓存将 DNS 名称解析为缓存中的相同终端节点。这对于轮询路由和故障转移场景都是一个问题。

> **Note**
>
> 禁用任何 DNS 缓存设置以每次强制执行 DNS 解析。

数据库集群在可用 Neptune 副本之间分配对读取器端点的连接请求。如果数据库集群只包含主数据库实例，则读取方终端节点从主数据库实例为连接请求提供服务。如果为该数据库集群创建了 Neptune 副本，则读取器端点将从新 Neptune 副本继续为针对读取器端点的连接请求提供服务，对服务造成的中断最少。

以下示例介绍 Neptune 数据库集群的读取器端点。

```
mydbcluster.cluster-ro-123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune 实例端点

实例端点是 Neptune 数据库集群中数据库实例的一个端点，连接到该特定数据库实例。数据库集群中的每个数据库实例具有自己的唯一实例终端节点，而不论具有何种实例类型。因此，数据库集群的当前主数据库实例具有一个实例终端节点。数据库集群中的每个 Neptune 副本也都具有一个实例端点。

对于可能不适合使用集群终端节点或读取器终端节点的场景，实例终端节点提供对与数据库集群连接的直接控制。例如，您的客户端应用程序可能根据工作负载类型需要精细的负载均衡。在这种情况下，您可以配置多个客户端以连接到数据库集群中的不同 Neptune 副本，以便分配读取工作负载。

以下示例介绍 Neptune 数据库集群中数据库实例的实例端点。

```
mydbinstance.123456789012.us-east-1.neptune.amazonaws.com:8182
```

Neptune 自定义端点

Neptune 集群的自定义端点表示一组选定数据库实例。在连接到端点时，Neptune 选择组中的实例之一来处理连接。您可以定义此终端节点引用的实例，并确定此终端节点的用途。

在创建自定义端点之前，Neptune 数据库集群没有自定义端点，您可以为每个预调配的 Neptune 集群创建最多五个自定义端点。

自定义终端节点根据数据库实例的只读或读/写功能以外的条件提供负载均衡的数据库连接。因为连接可以转到与端点关联的任何数据库实例，所以，请确保该组中的所有实例共享相同的性能和内存容量特征。在使用自定义终端节点时，通常不使用该集群的读取器终端节点。

此特征适用于具有特殊类型的工作负载的高级用户，在这些工作负载下，使集群中的所有 Neptune 副本保持相同是不切实际的。利用自定义端点，您可以调整用于每个连接的数据库实例的容量。

例如，如果您定义了多个自定义端点，这些端点连接到具有不同实例类的实例组，则可以将具有不同性能需求的用户定向到最适合其用例的端点。

以下示例介绍 Neptune 数据库集群中数据库实例的自定义端点：

```
myendpoint.cluster-custom-123456789012.us-east-1.neptune.amazonaws.com:8182
```

请参阅使用自定义端点了解更多信息。

Neptune 端点注意事项

使用 Neptune 端点时，请考虑以下问题：

- 使用实例终端节点连接到数据库集群中的特定数据库实例之前，请考虑改为对数据库集群使用集群终端节点或读取方终端节点。

  集群终端节点和读取方终端节点可提供对高可用性场景的支持。如果数据库集群的主数据库实例失败，Neptune 将自动失效转移到新的主数据库实例。它通过将现有 Neptune 副本提升为新的主数据库实例或者创建新的主数据库实例来完成该操作。如果发生了失效转移，您可以使用集群端点重

新连接到新提升或新创建的主数据库实例，或者使用读取器端点重新连接到数据库集群中的其它 Neptune 副本之一。

如果未采用此方法，您仍可以确保连接到数据库集群中的合适数据库实例来执行目标操作。为此，您可以在故障转移之后，以手动或以编程方式先搜索数据库集群中得到的可用数据库实例集，并确认其实例类型，然后再使用特定数据库实例的实例终端节点。

有关故障转移的更多信息，请参阅Neptune 数据库集群的容错能力。

- 读取器端点仅将连接定向到 Neptune 数据库集群中的可用 Neptune 副本。它不会定向特定查询。

  > **Important**
  >
  > Neptune 不会执行负载均衡。

  如果您要实现查询的负载均衡以分配数据库集群的读取工作负载，则必须在应用程序中进行管理。您必须使用实例端点直接连接到 Neptune 副本以进行负载平衡。

- 读取器终端节点轮询路由的运行方式是更改 DNS 条目指向的主机。客户端必须创建新连接并再次解析 DNS 记录，以获取到可能新的只读副本的连接。

- 在失效转移期间，如果将 Neptune 副本提升为新的主数据库实例，则读取器端点可能会在短时间内将连接定向到数据库集群的新的主数据库实例。

在 Neptune 中使用自定义端点

在将数据库实例添加到自定义终端节点或将其从自定义终端节点中删除时，与该数据库实例的任何现有连接都将保持活动状态。

您可以定义要包含在自定义端点中的数据库实例的列表（静态列表），也可以定义要从自定义端点中排除的数据库实例的列表（排除列表）。您可以使用包含/排除机制将数据库实例细分为多个组，并确保自定义端点涵盖集群中的所有数据库实例。每个自定义终端节点只能包含其中一种列表类型。

在中 AWS Management Console，该选项由 “附加 future 实例添加到此集群” 复选框表示。如果清除该复选框，则自定义终端节点将使用仅包含对话框中指定的数据库实例的静态列表。选中此复选框后，自

定义端点将使用排除列表。在这种情况下，自定义端点表示集群中的所有数据库实例（包括您将来添加的任何实例），但在对话框中未选中的实例除外。

当数据库实例由于失效转移或提升而在主实例和 Neptune 副本之间更改角色时，Neptune 不会更改在静态或排除列表中指定的数据库实例。

您可以将一个数据库实例与多个自定义终端节点关联。例如，假设您将新数据库实例添加到集群。在这一情况下，数据库实例将添加到它符合条件的所有自定义端点。为其定义的静态列表或排除列表决定了可以向集群添加哪个数据库实例。

如果端点包含数据库实例的静态列表，则新添加的 Neptune 副本不会添加到该端点。相反，如果端点具有排除列表，则新添加的 Neptune 副本将添加到其中，前提是未在排除列表中指定它们。

如果一个 Neptune 副本变得不可用，该副本仍将与其自定义端点关联。无论该副本处于运行状况不正常、已停止、重启还是由于其它原因不可用，都是如此。但是，只要副本仍然不可用，您就无法通过任何端点连接到它。

由于新创建的 Neptune 集群没有自定义端点，因此，您必须自行创建和管理这些端点。从快照还原的 Neptune 集群也是如此，因为快照中不包含自定义端点。您在还原后再次创建它们，并在还原的集群与原始集群位于同一区域时选择新的端点名称。

创建自定义终端节点

使用 Neptune 控制台管理自定义端点。为此，请导航到 Neptune 集群的详细信息页面，然后使用自定义端点部分中的控件。

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 导航到集群详细信息页面。
3. 在端点部分中选择 `Create custom endpoint` 操作。
4. 为自定义端点选择名称，该名称对于用户 ID 和区域是唯一的。名称长度必须不超过 63 个字符，并采用以下格式：

   endpointName`.cluster-custom-`*customerDnsIdentifier*`.`*dnsSuffix*

   由于自定义终端节点名称不包含集群的名称，因此，如果您重命名集群，则不必更改这些名称。但是，您不能为同一区域中的多个集群重用相同的自定义端点名称。为每个自定义终端节点指定一个名称，该名称在特定区域内的用户 ID 所拥有的集群中是唯一的。

5. 要选择即使在集群扩展时也保持不变的数据库实例列表，请清除 Attach future instances added to this cluster (挂载以后添加到此集群的实例) 复选框。如果选中该复选框，在将任何新实例添加到集群时，自定义端点将动态添加这些实例。

查看自定义终端节点

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 导航到数据库集群的集群详细信息页面。
3. 端点部分仅包含有关自定义端点的信息（有关内置端点的详细信息列在主要详细信息部分中）。要查看特定自定义端点的详细信息，请选择其名称以显示该端点的详细信息页。

编辑自定义终端节点

您可以编辑自定义端点的属性以更改与其关联的数据库实例。您也可以在静态列表和排除列表之间切换。

当编辑操作的更改正在进行中时，您无法连接或使用自定义终端节点。进行更改后，在端点状态返回可用并且您可以再次连接之前，可能需要几分钟时间。

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 导航到集群详细信息页面。
3. 在端点部分，选择要编辑的自定义端点的名称。
4. 在该端点的详细信息页面中，选择编辑操作。

删除自定义终端节点

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 导航到集群详细信息页面。
3. 在端点部分，选择要删除的自定义端点的名称。
4. 在该端点的详细信息页面中，选择删除操作。

将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中

默认情况下，Neptune 为每个查询分配一个唯一的 queryId 值。您可以使用该 ID 获取有关正在运行的查询的信息（请参阅 Gremlin 查询状态 API 或 SPARQL 查询状态 API），也可以取消查询（请参阅 Gremlin 查询取消 或 SPARQL 查询取消）。

Neptune 还允许您在 HTTP 标头中为 Gremlin 或 SPARQL 查询指定自己的 queryId 值，或者通过使用 queryId 查询提示为 SPARQL 查询指定该值。分配自己的 queryID 可轻松跟踪查询来获取状态或取消查询。

> **Note**
>
> 从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始，此特征可用。

使用 HTTP 标头注入自定义 **queryId** 值

对于 Gremlin 和 SPARQL，都可以使用 HTTP 标头将您自己的 queryId 值注入到查询中。

Gremlin 示例

```
curl -XPOST https://your-neptune-endpoint:port \
    -d "{\"gremlin\": \
        \"g.V().limit(1).count()\" , \
        \"queryId\":\"4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47\"  }"
```

SPARQL 示例

```
curl https://your-neptune-endpoint:port/sparql \
    -d "query=SELECT * WHERE { ?s ?p ?o } " \
       --data-urlencode \
       "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

使用 SPARQL 查询提示注入自定义 **queryId** 值

以下是使用 SPARQL queryId 查询提示将自定义 queryId 值注入到 SPARQL 查询中的示例：

```
curl https://your-neptune-endpoint:port/sparql \
    -d "PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#> \
```

```
       SELECT * WHERE { hint:Query hint:queryId \"4d5c4fae-
aa30-41cf-9e1f-91e6b7dd6f47\" \
       {?s ?p ?o}}"
```

使用 **queryId** 值检查查询状态

Gremlin 示例

```
curl https://your-neptune-endpoint:port/gremlin/status \
    -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

SPARQL 示例

```
curl https://your-neptune-endpoint:port/sparql/status \
    -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
```

Neptune 实验室模式

您可以使用 Amazon Neptune 实验室模式启用当前 Neptune 引擎版本中新增的、但尚未准备好用于生产环境因而默认未启用的特征。这让您能够在开发和测试环境中试用这些功能。

> **Note**
>
> 从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始，此特征可用。

使用 Neptune 实验室模式

使用 neptune_lab_mode 数据库集群参数来启用或禁用特征。为此，您可以在数据库集群参数组的 neptune_lab_mode 参数值中包含 *(feature name)*=enabled 或 *(feature name)*=disabled。

例如，在该引擎版本中，您可以将 neptune_lab_mode 参数设置为 Streams=disabled,ReadWriteConflictDetection=enabled。

有关如何编辑数据库的数据库集群参数组的信息，请参阅 编辑参数组。请注意，您无法编辑默认的数据库集群参数组。如果使用默认组，则必须先创建一个新的数据库集群参数组，然后才能设置 neptune_lab_mode 参数。

> **Note**
>
> 当您更改静态数据库集群参数（例如 neptune_lab_mode）时，必须重启集群的主（写入器）实例才能使更改生效。在版本：1.2.0.0（2022 年 7 月 21 日）之前，数据库集群中的所有只读副本将在主实例重启时自动重启。
> 从版本：1.2.0.0（2022 年 7 月 21 日）开始，重启主实例不会导致任何副本重启。这意味着您必须分别重启每个实例，才能获得数据库集群参数的更改（请参阅参数组）。

> **Important**
>
> 目前，如果您提供了错误的实验室模式参数，或者您的请求由于其它原因而失败，则可能不会收到失败的通知。您应始终通过调用状态 API 来验证实验室模式更改请求是否成功，如下所示：

```
curl -G https://your-neptune-endpoint:port/status
```

状态结果包括实验室模式信息，这些信息将显示您请求的更改是否已完成：

```
{
  "status":"healthy",
  "startTime":"Wed Dec 29 02:29:24 UTC 2021",
  "dbEngineVersion":"development",
  "role":"writer",
  "dfeQueryEngine":"viaQueryHint",
  "gremlin":{"version":"tinkerpop-3.5.2"},
  "sparql":{"version":"sparql-1.1"},
  "opencypher":{"version":"Neptune-9.0.20190305-1.0"},
  "labMode":{
    "ObjectIndex":"disabled",
    "ReadWriteConflictDetection":"enabled"
  },
  "features":{
    "LookupCache":{"status":"Available"},
    "ResultCache":{"status":"disabled"},
    "IAMAuthentication":"disabled",
    "Streams":"disabled",
    "AuditLog":"disabled"
  },
  "settings":{"clusterQueryTimeoutInMs":"120000"}
}
```

当前使用实验室模式可以访问以下特征：

OSGP 索引

Neptune 现在可以维护第四个索引，即 OSGP 索引，这对于具有大量谓词的数据集非常有用（请参阅启用 OSGP 索引）。

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.2.1 开始推出。

通过在 neptune_lab_mode 数据库集群参数中设置 ObjectIndex=enabled，可以在新且空的 Neptune 数据库集群中启用 OSGP 索引。只能在新且空的数据库集群中启用 OSGP 索引。

默认情况下，OSGP 索引处于禁用状态。

> **Note**
>
> 设置 neptune_lab_mode 数据库集群参数以启用 OSGP 索引后，必须重启集群的写入器实例才能使更改生效。

> **Warning**
>
> 如果您通过设置 ObjectIndex=disabled 禁用已启用的 OSGP 索引，然后在添加更多数据后将其重新启用，则该索引将无法正确构建。不支持按需重建索引，因此，只有在数据库为空时才应启用 OSGP 索引。

形式化事务语义

Neptune 更新了并发事务的形式语义（请参阅Neptune 中的事务语义）。

在 neptune_lab_mode 参数中，将 ReadWriteConflictDetection 用作启用或禁用形式化事务语义的功能名称。

默认情况下，已启用形式化事务语义。如果要恢复为以前的行为，请在为数据库集群 neptune_lab_mode 参数设置的值中包含 ReadWriteConflictDetection=disabled。

扩展的日期时间支持

Neptune 扩展了对日期时间功能的支持。要启用扩展格式的日期时间，请在为数据库集群neptune_lab_mode参数设置的值DatetimeMillisecond=enabled中包含该值。

Amazon Neptune 替代查询引擎 (DFE)

Amazon Neptune 有一个名为 DFE 的替代查询引擎，它比最初的 Neptune 引擎更高效地使用 CPU 内核、内存和 I/O 等数据库实例资源。

> **Note**
>
> 对于大型数据集，DFE 引擎可能无法在 t3 实例上正常运行。

DFE 引擎运行 SPARQL、Gremlin 和 openCypher 查询，并支持各种计划类型，包括深左深度、密集型和混合型计划类型。计划运算符既可以调用计算操作（在一组预留的计算内核上运行），也可以调用 I/O 操作（每个操作都在 I/O 线程池中各自的线程上运行）。

DFE 使用预生成的有关您的 Neptune 图形数据的统计数据，就如何构建查询做出明智的决定。有关如何生成这些统计数据的信息，请参阅DFE 统计数据。

计划类型和所用计算线程数量的选择是根据预生成的统计数据和 Neptune 头节点中可用的资源自动做出的。对于具有内部计算并行性的计划，结果的顺序不是预先确定的。

控制 Neptune DFE 引擎的使用位置

默认情况下，实例的 neptune_dfe_query_engine 实例参数设置为 `viaQueryHint`，这会导致 DFE 引擎仅用于 openCypher 查询，以及显式包含 `useDFE` 查询提示（设置为 `true`）的 Gremlin 和 SPARQL 查询。

通过将 `neptune_dfe_query_engine` 实例参数设置为 `enabled`，您可以完全启用 DFE 引擎，以便尽可能使用该引擎。

您也可以通过包含特定 Gremlin 查询或 SPARQL 查询的 `useDFE` 查询提示来禁用 DFE。此查询提示允许您阻止 DFE 执行该特定查询。

您可以使用 实例状态 调用来确定是否在实例中启用 DFE，如下所示：

```
curl -G https://your-neptune-endpoint:port/status
```

然后，状态响应会指定 DFE 是否已启用：

```
{
```

```
  "status":"healthy",
  "startTime":"Wed Dec 29 02:29:24 UTC 2021",
  "dbEngineVersion":"development",
  "role":"writer",
  "dfeQueryEngine":"viaQueryHint",
  "gremlin":{"version":"tinkerpop-3.5.2"},
  "sparql":{"version":"sparql-1.1"},
  "opencypher":{"version":"Neptune-9.0.20190305-1.0"},
  "labMode":{
    "ObjectIndex":"disabled",
    "ReadWriteConflictDetection":"enabled"
  },
  "features":{
    "ResultCache":{"status":"disabled"},
    "IAMAuthentication":"disabled",
    "Streams":"disabled",
    "AuditLog":"disabled"
  },
  "settings":{"clusterQueryTimeoutInMs":"120000"}
}
```

Gremlin `explain` 和 `profile` 结果会告诉您 DFE 是否正在执行查询。请参阅Gremlin explain 报告中包含的信息以了解 `explain`，并参阅DFE profile 报告以了解 `profile`。

同样，SPARQL `explain` 会告诉您 DFE 是否正在执行 SPARQL 查询。有关更多详细信息，请参阅启用 DFE 时的 SPARQL explain 输出示例和DFENode 运算符。

Neptune DFE 支持的查询构造

目前，Neptune DFE 支持 SPARQL 和 Gremlin 查询构造的子集。

对于 SPARQL，这是连接性基本图形模式的子集。

对于 Gremlin，它通常是包含遍历链的查询子集，这些遍历链不包含一些更复杂的步骤。

您可以通过以下方式了解您的一个查询是全部还是部分由 DFE 执行：

- 在 Gremlin 中，`explain` 和 `profile` 结果告诉您 DFE 正在执行查询的哪些部分（如果有）。请参阅Gremlin explain 报告中包含的信息以了解 `explain`，并参阅DFE profile 报告以了解 `profile`。另请参阅使用 explain 和 profile 调整 Gremlin 查询。

  有关各个 Gremlin 步骤的 Neptune 引擎支持的详细信息，请参阅Gremlin 步骤支持。

- 同样，SPARQL `explain` 会告诉您 DFE 是否正在执行 SPARQL 查询。有关更多详细信息，请参阅启用 DFE 时的 SPARQL `explain` 输出示例和`DFENode` 运算符。

管理 Neptune DFE 要使用的统计数据

> **Note**
>
> 对 openCypher 的支持取决于 Neptune 中的 DFE 查询引擎。
> DFE 引擎在 Neptune 引擎版本 1.0.3.0 中首次在实验室模式下可用，从 Neptune 引擎版本 1.0.5.0 开始，它默认处于启用状态，但仅用于查询提示和 openCypher 支持。
> 从 Neptune 引擎版本 1.1.1.0 开始，DFE 引擎不再处于实验室模式，现在使用实例的数据库参数组中的 neptune_dfe_query_engine 实例参数进行控制。

在计划查询执行时，DFE 引擎使用有关 Neptune 图形中数据的信息来进行有效的权衡。这些信息采用统计数据的形式，包括可以指导查询计划的所谓特性集和谓词统计数据。

从引擎版本 1.2.1.0 开始，您可以使用摘要 API 或端点从这些统计数据中检索有关图表的GetGraph摘要信息。`summary`

目前，每当图形中超过 10% 的数据发生变化或最新的统计数据超过 10 天时，就会重新生成这些 DFE 统计数据。然而，这些触发因素在未来可能会发生变化。

> **Note**
>
> 在 T3 和 T4g 实例上禁用统计数据生成，因为它可能超过这些实例类型的内存容量。

您可以通过以下端点之一管理 DFE 统计数据的生成：

- `https://`*`your-neptune-host`*`:`*`port`*`/rdf/statistics` （对于 SPARQL）。
- `https://`*`your-neptune-host`*`:`*`port`*`/propertygraph/statistics`（对于 Gremlin 和 openCypher），以及它的替代版本：`https://`*`your-neptune-host`*`:`*`port`*`/pg/statistics`。

> **Note**
>
> 从引擎版本 1.1.1.0 开始，Gremlin 统计数据端点 (`https://`*`your-neptune-host`*`:`*`port`*`/gremlin/statistics`) 已被弃用，转而使用 `propertygraph` 或 `pg` 端点。为了向后兼容，它仍然受支持，但可能会在将来版本中移除。

从引擎版本 1.2.1.0 开始，SPARQL 统计数据端点 (`https://your-neptune-host:port/sparql/statistics`) 已被弃用，转而使用 `rdf` 端点。为了向后兼容，它仍然受支持，但可能会在将来版本中移除。

在下面的示例中，`$STATISTICS_ENDPOINT` 代表这些端点 URL 中的任何一个。

> **Note**
>
> 如果 DFE 统计数据端点位于读取器实例上，则它只能处理状态请求。其它请求将失败，并显示 `ReadOnlyViolationException`。

生成 DFE 统计数据的大小限制

当前，如果达到以下任一大小限制，DFE 统计数据生成就会停止：

- 生成的特性集数量不得超过 5 万个。
- 生成的谓词统计数据数量不得超过一百万个。

这些限制可能会发生变化。

DFE 统计数据的当前状态

您可使用以下 `curl` 请求检查 DFE 统计数据的当前状态：

```
curl -G "$STATISTICS_ENDPOINT"
```

对状态请求的响应包含以下字段：

- `status` – 请求的 HTTP 返回代码。如果请求成功，则代码为 `200`。有关常见错误的列表，请参阅常见错误。
- `payload`:
  - `autoCompute` – (布尔值) 表示是否启用了自动统计数据生成。
  - `active` – (布尔值) 表示是否启用了 DFE 统计数据生成。
  - `statisticsId` – 报告当前统计数据生成运行的 ID。值为 `-1` 表示未生成任何统计数据。
  - `date` – 最近生成 DFE 统计数据的 UTC 时间，采用 ISO 8601 格式。

> **Note**
>
> 在引擎版本 1.2.1.0 之前，这是用分钟精度表示的，但从引擎版本 1.2.1.0 开始，它以毫秒精度表示（例如 `2023-01-24T00:47:43.319Z`）。

- `note` – 有关统计数据无效时出现的问题的注释。
- `signatureInfo` – 包含有关在统计数据中生成的特性集的信息（在引擎版本 1.2.1.0 之前，此字段命名为 `summary`）。这些通常不可直接操作：
  - `signatureCount` – 所有特性集中的签名总数。
  - `instanceCount` – 特性集实例的总数。
  - `predicateCount` – 唯一谓词的总数。

未生成统计数据时对状态请求的响应如下所示：

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : -1
   }
}
```

如果 DFE 统计数据可用，则响应类似于以下内容：

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : true,
    "statisticsId" : 1588893232718,
    "date" : "2020-05-07T23:13Z",
    "summary" : {
      "signatureCount" : 5,
      "instanceCount" : 1000,
      "predicateCount" : 20
    }
  }
```

```
}
```

如果生成 DFE 统计数据失败，例如，因为它超出了统计数据大小限制，则响应如下所示：

```
{
  "status" : "200 OK",
  "payload" : {
    "autoCompute" : true,
    "active" : false,
    "statisticsId" : 1588713528304,
    "date" : "2020-05-05T21:18Z",
    "note" : "Limit reached: Statistics are not available"
  }
}
```

禁用自动生成 DFE 统计数据

默认情况下，启用 DFE 时会启用自动生成 DFE 统计数据。

您可以按如下方式禁用自动生成：

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "disableAutoCompute" }'
```

如果请求成功，则 HTTP 响应代码为 200，响应为：

```
{
  "status" : "200 OK"
}
```

您可以通过发出状态请求并检查响应中的 `autoCompute` 字段是否设置为 `false`，确认自动生成已禁用。

禁用自动生成统计数据不会终止正在进行的统计数据计算。

如果您请求对读取器实例而不是数据库集群的写入器实例禁用自动生成，则请求将失败，HTTP 返回代码为 400，输出如下所示：

```
{
  "detailedMessage" : "Writes are not permitted on a read replica instance",
  "code" : "ReadOnlyViolationException",
  "requestId":"8eb8d3e5-0996-4a1b-616a-74e0ec32d5f7"
```

```
}
```

有关其它常见错误的列表，请参阅常见错误。

重新启用 DFE 统计数据的自动生成

默认情况下，启用 DFE 时已启用自动生成 DFE 统计数据。如果禁用自动生成，则可以稍后按如下方式重新启用：

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "enableAutoCompute" }'
```

如果请求成功，则 HTTP 响应代码为 200，响应为：

```
{
  "status" : "200 OK"
}
```

您可以通过发出状态请求并检查响应中的 autoCompute 字段是否设置为 true，确认自动生成已启用。

手动触发 DFE 统计数据生成

您可以手动启动 DFE 统计数据生成，如下所示：

```
curl -X POST "$STATISTICS_ENDPOINT" -d '{ "mode" : "refresh" }'
```

如果请求成功，则输出如下所示，HTTP 返回代码为 200：

```
{
  "status" : "200 OK",
  "payload" : {
    "statisticsId" : 1588893232718
  }
}
```

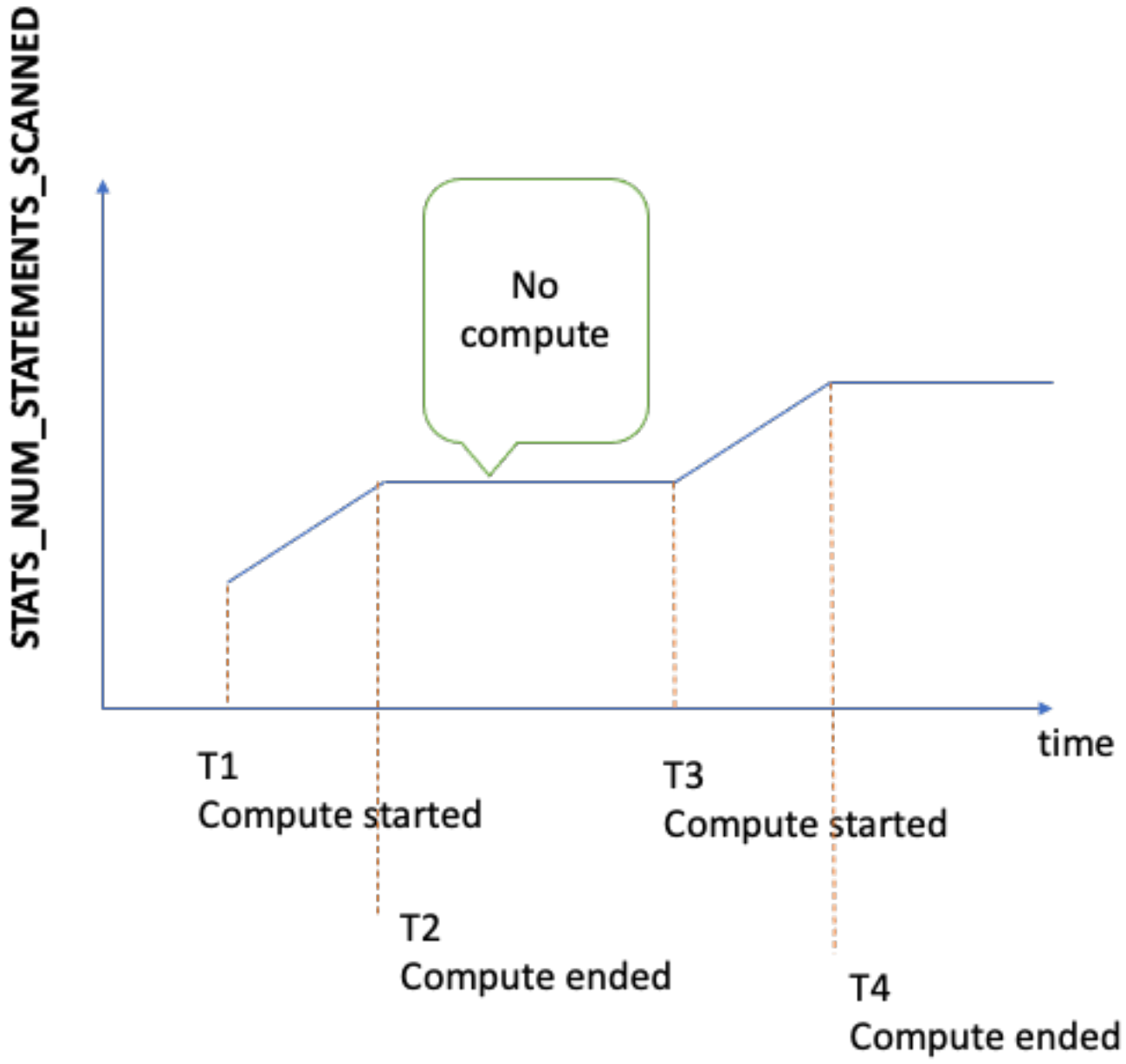
输出中的 statisticsId 是当前正在发生的统计数据生成运行的 ID。如果在请求时运行已在进行中，则该请求将返回该运行的 ID，而不是启动新的运行。一次只能发生一个统计数据生成运行。

如果在生成 DFE 统计数据时发生失效转移，则新的写入器节点将获取上次处理的检查点，并从此处恢复统计数据运行。

使用**StatsNumStatementsScanned** CloudWatch 指标监控统计数据计算

该StatsNumStatementsScanned CloudWatch 指标返回自服务器启动以来为统计计算而扫描的语句总数。它会在每个统计数据计算切片处更新。

每次触发统计数据计算时，这个数字都会增加，当没有进行任何计算时，它会保持不变。因此，查看一段时间内的 StatsNumStatementsScanned 值图，可以非常清楚地了解统计数据计算发生的时间和速度：



发生计算时，图形的斜率显示出速度有多快（斜率越陡，计算统计数据的速度就越快）。

如果图形只是 0 处的一条平线，则表示统计数据特征已启用，但根本没有计算任何统计数据。如果统计数据特征已被禁用，或者您使用的引擎版本不支持统计数据计算，则 `StatsNumStatementsScanned` 不存在。

如前所述，您可以使用统计数据 API 禁用统计数据计算，但将其禁用可能会导致统计数据无法保持最新，这反过来又会导致 DFE 引擎的查询计划生成不佳。

有关如何使用的信息，请参阅使用亚马逊监控 Neptune CloudWatch CloudWatch。

对 DFE 统计终端节点使用 AWS Identity and Access Management (IAM) 身份验证

您可以使用 awscurl 或任何其它支持 HTTPS 和 IAM 的工具，通过 IAM 身份验证安全地访问 DFE 统计数据端点。请参阅将 awscurl 与临时凭证结合使用以安全地连接到启用了 IAM 身份验证的数据库集群，了解如何设置正确的凭证。完成此操作后，您可以发出这样的状态请求：

```
awscurl "$STATISTICS_ENDPOINT" \
    --region (your region) \
    --service neptune-db
```

或者，例如，您可以创建名为 `request.json` 的 JSON 文件，其中包含：

```
{ "mode" : "refresh" }
```

然后，您可以像这样手动启动统计数据生成：

```
awscurl "$STATISTICS_ENDPOINT" \
    --region (your region) \
    --service neptune-db \
    -X POST -d @request.json
```

删除 DFE 统计数据

您可以通过向统计数据端点发出 HTTP DELETE 请求来删除数据库中的所有统计数据：

```
curl -X "DELETE" "$STATISTICS_ENDPOINT"
```

有效的 HTTP 返回代码为：

- 200 – 删除已成功。

  在这种情况下，典型的响应如下所示：

```
{
  "status" : "200 OK",
  "payload" : {
      "active" : false,
      "statisticsId" : -1
  }
}
```

- 204 – 没有要删除的统计数据。

  在这种情况下，响应为空（无响应）。

如果您向读取器节点上的统计数据端点发送删除请求，则引发 ReadOnlyViolationException。

DFE 统计请求的常见错误代码

以下是您向统计数据端点发出请求时可能发生的常见错误列表：

- AccessDeniedException – 返回代码：400。消息：Missing Authentication Token。
- BadRequestException（对于 Gremlin 和 openCypher）– 返回代码：400。消息：Bad route: /pg/statistics。
- BadRequestException（对于 RDF 数据）- 返回代码：400。消息：Bad route: /rdf/statistics。
- InvalidParameterException – 返回代码：400。消息：Statistics command parameter 'mode' has unsupported value '*the invalid value*'。
- MissingParameterException – 返回代码：400。消息：Content-type header not specified.。
- ReadOnlyViolationException – 返回代码：400。消息：Writes are not permitted on a read replica instance。

例如，如果您在未启用 DFE 和统计数据时发出请求，则会得到如下响应：

```
{
  "code" : "BadRequestException",
```

```
  "requestId" : "b2b8f8ee-18f1-e164-49ea-836381a3e174",
  "detailedMessage" : "Bad route: /sparql/statistics"
}
```

获取有关图形的快速摘要报告

Neptune 图形摘要 API 会检索有关您的图形的以下信息：

- 对于属性 (PG) 图，图形摘要 API 会返回节点和边缘标签以及属性键的只读列表，以及节点、边缘和属性的计数。
- 对于资源描述框架 (RDF) 图，图形摘要 API 会返回类和谓词键的只读列表，以及四元组、主题和谓词的计数。

> **Note**
>
> 图形摘要 API 是在 Neptune 引擎版本 1.2.1.0 中引入的。

借助图形摘要 API，您可以快速、全面了解图形数据的大小和内容。还可以使用 `%summary` Neptune Workbench 魔术命令在 Neptune 笔记本中以交互方式使用 API。在图形应用程序中，API 可用于通过在搜索过程中提供已发现的节点或边缘标签来改善搜索结果。

图形摘要数据是从 Neptune DFE 引擎在运行时系统期间计算的 DFE 统计数据中提取的，只要有 DFE 统计数据就可用。当您创建新的 Neptune 数据库集群时，统计数据默认处于启用状态。

> **Note**
>
> 在 `t3` 和 `t4` 实例类型（即在 `db.t3.medium` 和 `db.t4g.medium` 实例类型上）上禁用统计数据生成，以节省内存。因此，这些实例类型上的图形摘要数据也不可用。

您可使用统计数据状态 API 检查 DFE 统计数据的状态。只要未禁用自动生成统计数据，统计数据就会定期自动更新。

如果您想在请求图形摘要时确保统计数据尽可能是最新的，则可以就在检索摘要之前手动触发统计数据更新。如果在计算统计数据时图形发生变化，它们必然会稍微滞后，但程度不会太大。

使用图形摘要 API 检索图形摘要信息

对于您使用 Gremlin 或 openCypher 进行查询的属性图，您可以从属性图摘要端点检索图形摘要。该端点有长的 URI 和短的 URI：

- https://*your-neptune-host*:*port*/propertygraph/statistics/summary
- https://*your-neptune-host*:*port*/pg/statistics/summary

对于使用 SPARQL 查询的 RDF 图形，您可以从 RDF 摘要端点检索图形摘要：

- https://*your-neptune-host*:*port*/rdf/statistics/summary

这些端点是只读的，仅支持 HTTP GET 操作。如果 $GRAPH_SUMMARY_ENDPOINT 设置为要查询的任何端点的地址，则可以使用 curl 和 HTTP GET 检索摘要数据，如下所示：

```
curl -G "$GRAPH_SUMMARY_ENDPOINT"
```

如果您尝试检索图形摘要时没有可用的统计数据，则响应如下所示：

```
{
  "detailedMessage": "Statistics are not available. Summary can only be generated after
 statistics are available.",
  "requestId": "48c1f788-f80b-b69c-d728-3f6df579a5f6",
  "code": "StatisticsNotAvailableException"
}
```

图形摘要 API 的 **mode** URL 查询参数

图形摘要 API 接受名为 mode 的 URL 查询参数，该参数可以采用两个值之一，即 basic（默认）和 detailed。对于 RDF 图形，detailed 模式图形摘要响应包含一个附加 subjectStructures 字段。对于属性图，详细图形摘要响应包含另外两个字段，即 nodeStructures 和 edgeStructures。

要请求 detailed 图形摘要响应，请按以下方式包括 mode 参数：

```
curl -G "$GRAPH_SUMMARY_ENDPOINT?mode=detailed"
```

如果 mode 参数不存在，则默认使用 basic 模式，因此，虽然可以显式指定 ?mode=basic，但这不是必需的。

属性图 (PG) 的图形摘要响应

对于空的属性图，详细的图形摘要响应如下所示：

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-01-10T07:58:47.972Z",
    "graphSummary" : {
      "numNodes" : 0,
      "numEdges" : 0,
      "numNodeLabels" : 0,
      "numEdgeLabels" : 0,
      "nodeLabels" : [ ],
      "edgeLabels" : [ ],
      "numNodeProperties" : 0,
      "numEdgeProperties" : 0,
      "nodeProperties" : [ ],
      "edgeProperties" : [ ],
      "totalNodePropertyValues" : 0,
      "totalEdgePropertyValues" : 0,
      "nodeStructures" : [ ],
      "edgeStructures" : [ ]
    }
  }
}
```

属性图 (PG) 摘要响应包含以下字段：

- **`status`** – 请求的 HTTP 返回代码。如果请求成功，则代码为 200。

  有关常见错误的列表，请参阅常见图形摘要错误。
- **`payload`**
  - **`version`** – 此图形摘要响应的版本。
  - **`lastStatisticsComputationTime`** – Neptune 上次计算统计数据的时间戳，采用 ISO 8601 格式。
  - **`graphSummary`**
    - **`numNodes`** – 图形中的节点数。
    - **`numEdges`** – 图形中的边缘数。
    - **`numNodeLabels`** – 图形中不同节点标签的数量。
    - **`numEdgeLabels`** – 图形中不同边缘标签的数量。
    - **`nodeLabels`** – 图形中不同节点标签的列表。

- **edgeLabels** – 图形中不同边缘标签的列表。
- **numNodeProperties** – 图形中不同节点属性的数量。
- **numEdgeProperties** – 图形中不同边缘属性的数量。
- **nodeProperties** – 图形中不同节点属性的列表，以及使用每个属性的节点计数。
- **edgeProperties** – 图形中不同边缘属性的列表以及使用每个属性的边缘计数。
- **totalNodePropertyValues** – 所有节点属性的总使用次数。
- **totalEdgePropertyValues** – 所有边缘属性的总使用次数。
- **nodeStructures** – 只有在请求中指定 *mode=detailed* 时才会出现此字段。它包含节点结构列表，每个节点结构都包含以下字段：
  - **count** – 具有此特定结构的节点数量。
  - **nodeProperties** – 此特定结构中存在的节点属性列表。
  - **distinctOutgoingEdgeLabels** – 此特定结构中存在的不同传出边缘标签的列表。
- **edgeStructures** – 只有在请求中指定 *mode=detailed* 时才会出现此字段。它包含边缘结构列表，每个边缘结构都包含以下字段：
  - **count** – 具有此特定结构的边缘数量。
  - **edgeProperties** – 此特定结构中存在的边缘属性列表。

RDF 图形的图形摘要响应

对于空的 RDF 图形，详细的图形摘要响应如下所示：

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-01-10T07:58:47.972Z",
    "graphSummary" : {
      "numDistinctSubjects" : 0,
      "numDistinctPredicates" : 0,
      "numQuads" : 0,
      "numClasses" : 0,
      "classes" : [ ],
      "predicates" : [ ],
      "subjectStructures" : [ ]
    }
  }
```

```
}
```

RDF 图形摘要响应具有以下字段：

- **status** – 请求的 HTTP 返回代码。如果请求成功，则代码为 200。

  有关常见错误的列表，请参阅常见图形摘要错误。
- **payload**
  - **version** – 此图形摘要响应的版本。
  - **lastStatisticsComputationTime** – Neptune 上次计算统计数据的时间戳，采用 ISO 8601 格式。
  - **graphSummary**
    - **numDistinctSubjects** – 图形中不同主题的数量。
    - **numDistinctPredicates** – 图形中不同谓词的数量。
    - **numQuads** – 图形中四元组的数量。
    - **numClasses** – 图形中的类数。
    - **classes** – 图形中的类列表。
    - **predicates** – 图形中的谓词列表以及谓词计数。
    - **subjectStructures** – 只有在请求中指定 *mode=detailed* 时才会出现此字段。它包含主题结构列表，每个主题结构都包含以下字段：
      - **count** – 此特定结构的出现次数。
      - **predicates** – 此特定结构中存在的谓词列表。

属性图 (PG) 摘要响应示例

以下是包含示例属性图航线数据集的属性图的详细摘要响应：

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-03-01T14:35:03.804Z",
    "graphSummary" : {
      "numNodes" : 3748,
      "numEdges" : 51300,
      "numNodeLabels" : 4,
```

```
    "numEdgeLabels" : 2,
    "nodeLabels" : [
      "continent",
      "country",
      "version",
      "airport"
    ],
    "edgeLabels" : [
      "contains",
      "route"
    ],
    "numNodeProperties" : 14,
    "numEdgeProperties" : 1,
    "nodeProperties" : [
      {
        "desc" : 3748
      },
      {
        "code" : 3748
      },
      {
        "type" : 3748
      },
      {
        "country" : 3503
      },
      {
        "longest" : 3503
      },
      {
        "city" : 3503
      },
      {
        "lon" : 3503
      },
      {
        "elev" : 3503
      },
      {
        "icao" : 3503
      },
      {
        "region" : 3503
      },
```

```
        {
          "runways" : 3503
        },
        {
          "lat" : 3503
        },
        {
          "date" : 1
        },
        {
          "author" : 1
        }
      ],
      "edgeProperties" : [
        {
          "dist" : 50532
        }
      ],
      "totalNodePropertyValues" : 42773,
      "totalEdgePropertyValues" : 50532,
      "nodeStructures" : [
        {
          "count" : 3471,
          "nodeProperties" : [
            "city",
            "code",
            "country",
            "desc",
            "elev",
            "icao",
            "lat",
            "lon",
            "longest",
            "region",
            "runways",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [
            "route"
          ]
        },
        {
          "count" : 161,
          "nodeProperties" : [
```

```
            "code",
            "desc",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [
            "contains"
          ]
        },
        {
          "count" : 83,
          "nodeProperties" : [
            "code",
            "desc",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [ ]
        },
        {
          "count" : 32,
          "nodeProperties" : [
            "city",
            "code",
            "country",
            "desc",
            "elev",
            "icao",
            "lat",
            "lon",
            "longest",
            "region",
            "runways",
            "type"
          ],
          "distinctOutgoingEdgeLabels" : [ ]
        },
        {
          "count" : 1,
          "nodeProperties" : [
            "author",
            "code",
            "date",
            "desc",
            "type"
          ],
```

```
          "distinctOutgoingEdgeLabels" : [ ]
        }
      ],
      "edgeStructures" : [
        {
          "count" : 50532,
          "edgeProperties" : [
            "dist"
          ]
        }
      ]
    }
  }
}
```

示例 RDF 图形摘要响应

以下是包含示例 RDF 航线数据集的 RDF 图形的详细摘要响应：

```
{
  "status" : "200 OK",
  "payload" : {
    "version" : "v1",
    "lastStatisticsComputationTime" : "2023-03-01T14:54:13.903Z",
    "graphSummary" : {
      "numDistinctSubjects" : 54403,
      "numDistinctPredicates" : 19,
      "numQuads" : 158571,
      "numClasses" : 4,
      "classes" : [
        "http://kelvinlawrence.net/air-routes/class/Version",
        "http://kelvinlawrence.net/air-routes/class/Airport",
        "http://kelvinlawrence.net/air-routes/class/Continent",
        "http://kelvinlawrence.net/air-routes/class/Country"
      ],
      "predicates" : [
        {
          "http://kelvinlawrence.net/air-routes/objectProperty/route" : 50656
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/dist" : 50656
        },
        {
```

```
        "http://kelvinlawrence.net/air-routes/objectProperty/contains" : 7004
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/code" : 3747
      },
      {
        "http://www.w3.org/2000/01/rdf-schema#label" : 3747
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/type" : 3747
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/desc" : 3747
      },
      {
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#type" : 3747
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/icao" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/lat" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/region" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/runways" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/longest" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/elev" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/lon" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/country" : 3502
      },
      {
        "http://kelvinlawrence.net/air-routes/datatypeProperty/city" : 3502
      },
```

```
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/author" : 1
        },
        {
          "http://kelvinlawrence.net/air-routes/datatypeProperty/date" : 1
        }
      ],
      "subjectStructures" : [
        {
          "count" : 50656,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/dist"
          ]
        },
        {
          "count" : 3471,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/city",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/country",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/elev",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/icao",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lat",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lon",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/longest",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/region",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/runways",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://kelvinlawrence.net/air-routes/objectProperty/route",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 238,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://kelvinlawrence.net/air-routes/objectProperty/contains",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
```

```
        },
        {
          "count" : 31,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/city",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/country",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/elev",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/icao",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lat",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/lon",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/longest",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/region",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/runways",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 6,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        },
        {
          "count" : 1,
          "predicates" : [
            "http://kelvinlawrence.net/air-routes/datatypeProperty/author",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/code",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/date",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/desc",
            "http://kelvinlawrence.net/air-routes/datatypeProperty/type",
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
            "http://www.w3.org/2000/01/rdf-schema#label"
          ]
        }
      ]
    }
```

```
  }
}
```

对图表摘要终端节点使用 AWS Identity and Access Management (IAM) 身份验证

您可以使用 awscurl 或任何其它支持 HTTPS 和 IAM 的工具，通过 IAM 身份验证安全地访问图形摘要端点。请参阅将 awscurl 与临时凭证结合使用以安全地连接到启用了 IAM 身份验证的数据库集群，了解如何设置正确的凭证。完成此操作后，您可以提出这样的请求：

```
awscurl "$GRAPH_SUMMARY_ENDPOINT" \
    --region (your region) \
    --service neptune-db
```

> **Important**
>
> 创建临时证书的 IAM 身份或角色必须附加允许GetGraph摘要 IAM 操作的 IAM 策略。

有关您可能遇到的常见 IAM 错误的列表，请参阅IAM 身份验证错误。

图形摘要请求可能返回的常见错误代码

Neptune 服务错误代码	HTTP 状态	消息	错误情形	缓解措施
AccessDeniedException	403	身份验证令牌缺失。	未签名或签名错误的请求已发送到启用 IAM 的 Neptune 数据库。	在发送请求之前，使用 SigV4 对请求进行签名（请参阅IAM 和图形摘要）。
	403	*##:### ARN#####:### ARN#GetGraphSummary ######neptune-db##*	当图表GetGraph摘要请求发送到启用 IAM 的 Neptune 数据库时，IAM 策略不允许使用操作摘要。	确保附加到发出请求的用户或角色的 IAM policy 允许执行 `GetGraphSummary` 操作。

Neptune 服务错误代码	HTTP 状态	消息	错误情形	缓解措施
BadRequestException	400	统计数据被禁用，因此图形摘要也被禁用。	正在尝试提取禁用统计数据的可突增实例类型（t3 或 t4g）的摘要。	使用启用了统计数据生成的实例类型（除 t3 和 t4g 之外所有支持的实例）。
	400	错误路由：*/rdf/statistics/summarypathapi*	请求已发送到无效路径。	为图形摘要端点使用正确的路由。
InvalidParameterException	400	请求包含未知参数：“*##########*”。	在请求中指定无效的参数时。	仅在请求中使用有效的参数（例如 mode）。
InvalidParameterException	400	URI 查询参数“mode”具有不支持的值“*#####*”。	当请求中的 URL 参数“mode”后跟一个无效值时。	指定 URL 参数“mode”时，请使用有效值（例如 basic 或 detailed）。
MethodNotAllowedException	405	不允许此方法。	使用除 GET 之外的任何 HTTP 方法（例如 POST 或 DELETE）调用摘要端点。	调用摘要端点时使用 HTTP GET 方法。
StatisticsNotAvailableException	400	统计数据尚未计算，统计数据计算完成后将提供图形摘要。	当请求发送到摘要端点时，没有可用的统计数据。	等到统计数据生成完成。您可使用统计数据状态 API 检查统计数据生成的状态。
	400	已达到统计限制，因此图形摘要不可用。	统计数据生成已停止，因为它已达到统计数据大小限制。	此图形上没有图形摘要。

例如，如果您请求在启用了 IAM 身份验证的 Neptune 数据库中绘制摘要端点的图形，而请求者的 IAM policy 中没有必要的权限，您将收到如下响应：

```
{
  "detailedMessage": "User: arn:aws:iam::(account ID):(user or user name) is not
 authorized to perform: neptune-db:GetGraphSummary on resource: arn:aws:neptune-
db:(region):(account ID):(cluster resource ID)/*",
  "requestId": "7ac2b98e-b626-d239-1d05-74b4c88fce82",
  "code": "AccessDeniedException"
}
```

Amazon Neptune JDBC 连接

Amazon Neptune 发布了一款开源 JDBC 驱动程序，它支持 openCypher、Gremlin、SQL-Gremlin 和 SPARQL 查询。JDBC 连接使您可以使用 Tableau 等商业智能 (BI) 工具轻松连接到 Neptune。在 Neptune 上使用 JDBC 驱动程序不会产生额外费用，您仍然只需为消耗的 Neptune 资源付费。

该驱动程序与 JDBC 4.2 兼容，并且至少需要 Java 8。有关如何使用 JDBC 驱动程序的信息，请参阅 JDBC API 文档。

该 GitHub 项目包含驱动程序的详细文档，您可以在其中提交问题和打开功能请求：

适用于 Amazon Neptune 的 JDBC 驱动程序

- 将 SQL 与 JDBC 驱动程序结合使用
- 将 Gremlin 与 JDBC 驱动程序结合使用
- 将 openCypher 与 JDBC 驱动程序结合使用
- 将 SPARQL 与 JDBC 驱动程序结合使用

Neptune JDBC 驱动程序入门

要使用 Neptune JDBC 驱动程序连接到 Neptune 实例，要么必须将 JDBC 驱动程序部署在与您的 Neptune 数据库集群在同一 VPC 中的 Amazon EC2 实例上，要么该实例必须通过 SSH 隧道或负载均衡器可用。SSH 隧道可以在驱动程序内部设置，也可以在外部设置。

您可从此处下载驱动程序。该驱动程序打包为单个 JAR 文件，名称类似于 `neptune-jdbc-1.0.0-all.jar`。要使用它，请将 JAR 文件放在应用程序的 `classpath` 中。或者，如果您的应用程序使用 Maven 或 Gradle，则可以使用相应的 Maven 或 Gradle 命令从 JAR 安装驱动程序。

驱动程序需要 JDBC 连接 URL 才能连接 Neptune，格式如下：

```
jdbc:neptune:(connection
 type)://(host);property=value;property=value;...;property=value
```

GitHub 项目中每种查询语言的章节描述了您可以在 JDBC 连接 URL 中为该查询语言设置的属性。

如果 JAR 文件位于应用程序的 `classpath` 中，则无需进行其它配置。您可以使用 JDBC `DriverManager` 接口和 Neptune 连接字符串连接驱动程序。例如，如果您可以通过端口 8182 上的 `neptune-example.com` 端点访问您的 Neptune 数据库集群，则可以像这样连接 openCypher：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

void example() {
    String url = "jdbc:neptune:opencypher://bolt://neptune-example:8182";

    Connection connection = DriverManager.getConnection(url);
    Statement statement = connection.createStatement();

    connection.close();
}
```

GitHub 项目中每种查询语言的文档部分描述了在使用该查询语言时如何构造连接字符串。

将 Tableau 与 Neptune JDBC 驱动程序结合使用

要将 Tableau 与 Neptune JDBC 驱动程序结合使用，请先下载并安装最新版本的 Tableau Desktop。下载 Neptune JDBC 驱动程序的 JAR 文件，以及 Neptune Tableau 连接器文件（`.taco` 文件）。

在 Mac 上连接适用于 Neptune 的 Tableau

1. 将 Neptune JDBC 驱动程序 JAR 文件放入 `/Users/`*`(your user name)`*`/Library/Tableau/Drivers` 文件夹中。
2. 将 Neptune Tableau 连接器 `.taco` 文件放在 `/Users/`*`(your user name)`*`/Documents/My Tableau Repository/Connectors` 文件夹中。
3. 如果您启用了 IAM 身份验证，请为其设置环境。请注意，在 `.zprofile/`、`.zshenv/`、`.bash_profile` 等中设置的环境变量将不起作用。必须设置环境变量，这样它们才能由 GUI 应用程序加载。

   设置凭证的一种方法是将访问密钥和私密密钥放在 `/Users/`*`(your user name)`*`/.aws/credentials` 文件中。

   设置服务区域的一种简单方法是打开终端，然后使用应用程序的区域（例如，`us-east-1`）输入以下命令：

   ```
   launchctl setenv SERVICE_REGION region name
   ```

   还有其它方法可以设置在重启后仍然存在的环境变量，但是无论使用哪种技术，都必须设置 GUI 应用程序可以访问的变量。

4. 要将环境变量加载到 Mac 上的 GUI 中，请在终端上输入以下命令：

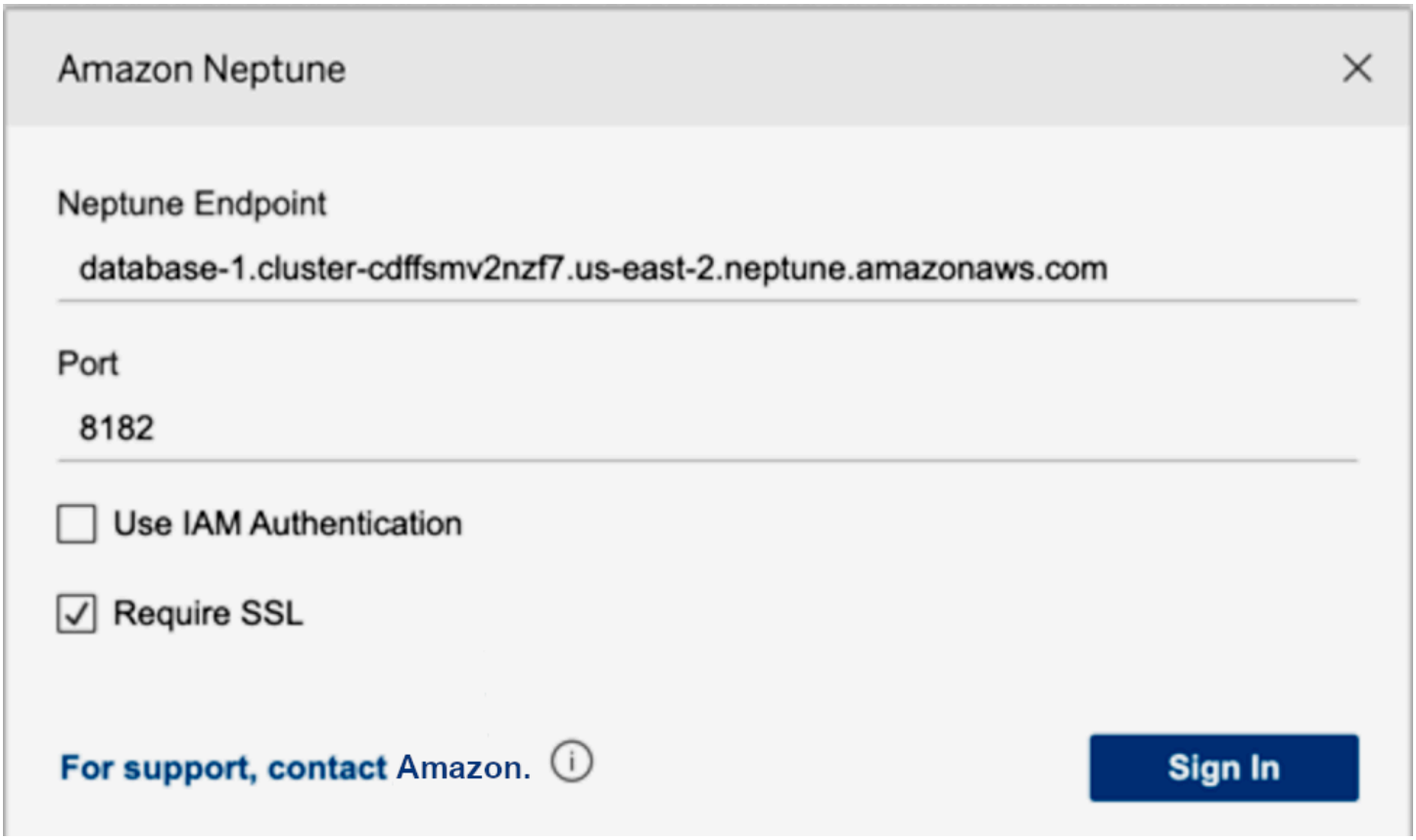
```
/Applications/Tableau/Desktop/2021.1.app/Contents/MacOS/Tableau
```

在 Windows 计算机上连接到适用于 Neptune 的 Tableau

1. 将 Neptune JDBC 驱动程序 JAR 文件放入 `C:\Program Files\Tableau\Drivers` 文件夹中。
2. 将 Neptune Tableau 连接器 `.taco` 文件放在 `C:\Users\`*`(your user name)`*`\Documents\My Tableau Repository\Connectors` 文件夹中。
3. 如果您启用了 IAM 身份验证，请为其设置环境。

   这可以像设置用户 `ACCESS_KEY`、`SECRET_KEY` 和 `SERVICE_REGION` 环境变量一样简单。

Tableau 处于打开状态后，选择窗口左侧的更多。如果 Tableau 连接器文件位置正确，则可以在显示的列表中选择 AWS的 Amazon Neptune：



您不必编辑端口，也不必添加任何连接选项。输入您的 Neptune 端点并设置您的 IAM 和 SSL 配置（如果您使用 IAM，则必须启用 SSL）。

当您选择登录时，如果您的图形很大，则连接时间可能超过 30 秒。Tableau 正在收集顶点和边缘表并联接边缘上的顶点，并创建可视化效果。

对 JDBC 驱动程序连接进行故障排除

如果驱动程序无法连接到服务器，请使用 JDBC `Connection` 对象的 `isValid` 函数来检查连接是否有效。如果函数返回 `false`（表示连接无效），请检查所连接的端点是否正确，以及您是否在 Neptune 数据库集群的 VPC 中，或者您是否有通往该集群的有效 SSH 隧道。

如果您从 `DriverManager.getConnection` 调用中获得 `No suitable driver found for` *`(connection string)`* 响应，则连接字符串的开头可能存在问题。请确保您的连接字符串以如下方式开头：

```
jdbc:neptune:opencypher://...
```

要收集有关连接的更多信息，可以在连接字符串中添加 `LogLevel`，如下所示：

```
jdbc:neptune:opencypher://(JDBC URL):(port);logLevel=trace
```

或者，您可以在输入属性中添加 `properties.put("logLevel", "trace")` 来记录跟踪信息。

Amazon Neptune 引擎更新

Amazon Neptune 定期发布引擎更新。您可以使用实例状态 API 确定当前安装的引擎发行版本。

引擎版本在 Amazon Neptune 的引擎版本 中列出，补丁在 最新更新 中列出。

您可以在集群维护中找到有关如何在数据库中发布更新以及如何升级 Neptune 引擎的更多信息。例如，引擎版本号中对版本编号进行了说明。

Amazon Neptune 中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon Neptune 的合规性计划，请参阅合规性计划范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Neptune 时应用责任共担模式。以下主题说明如何配置 Neptune 以实现您的安全性和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 Neptune 资源。

主题

Amazon Neptune 中的数据保护

AWS 分担责任模型分担责任模型适用于 Amazon Neptune 中的数据保护。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。

您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准 (FIPS) 第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括当你 AWS 服务 使用控制台、API 或 SDK 与 Neptune 或其他人合作时 AWS CLI。 AWS 在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

> **Important**
>
> 只有 Neptune 引擎版本 1.3.2.0 及更高版本支持 TLS 1.3。

您可以使用 AWS 已发布的 API 调用通过网络管理 Neptune。客户端必须使用强密码套件支持传输层安全性协议 (TLS) 1.2 或更高版本，如传输中加密中所述。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

接下来的部分将进一步介绍如何保护 Neptune 数据。

主题

- 每个 Amazon Neptune 数据库集群都位于 Amazon VPC 中
- 传输中加密：使用 SSL/HTTPS 连接到 Neptune

- 静态加密 Neptune 资源

每个 Amazon Neptune 数据库集群都位于 Amazon VPC 中

Amazon Neptune 数据库集群只能在 Amazon Virtual Private Cloud (Amazon VPC) 中创建，并且其端点只能在该 VPC 内访问，通常是从在该 VPC 中运行的 Amazon Elastic Compute Cloud (Amazon EC2) 实例进行访问。

您可以通过限制对 Neptune 数据库集群所在的 VPC 的访问权限来保护您的 Neptune 数据，如连接到您的 Amazon Neptune 图形中所述。

传输中加密：使用 SSL/HTTPS 连接到 Neptune

从引擎版本 1.0.4.0 开始，Amazon Neptune 仅允许使用安全套接字层 (SSL) 连接通过 HTTPS 连接到任何实例或集群端点。

Neptune 至少需要 TLS 版本 1.2，使用以下强密码套件：

- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`

从 Neptune 引擎版本 1.3.2.0 开始，Neptune 使用以下密码套件支持 TLS 版本 1.3：

- TLS_AES_128_GCM_SHA256
- TLS_AES_256_GCM_SHA384

即使在早期引擎版本中允许 HTTP 连接的地方，默认情况下，任何使用新数据库集群参数组的数据库集群都需要使用 SSL。为了保护您的数据，引擎版本 *`1.0.4.0`* 及以上版本的 Neptune 端点仅支持 HTTPS 请求。请参阅使用 HTTP REST 端点连接到 Neptune 数据库实例了解更多信息。

Neptune 自动为 Neptune 数据库实例提供 SSL 证书。您无需请求任何证书。证书在您创建新实例时提供。

Neptune 为您账户中的每个区域的实例分配一个通配符 SSL 证书。 AWS 证书提供集群终端节点、集群只读终端节点和实例终端节点的条目。

证书详细信息

以下条目包含在提供的证书中：

- 集群端点 — `*.cluster-a1b2c3d4wxyz.region.neptune.amazonaws.com`
- 只读端点 — `*.cluster-ro-a1b2c3d4wxyz.region.neptune.amazonaws.com`
- 实例端点 — `*.a1b2c3d4wxyz.region.neptune.amazonaws.com`

仅支持此处列出的条目。

代理连接

证书仅支持上一节中列出的主机名。

如果您使用的是负载均衡器或代理服务器（如 HAProxy），则必须使用 SSL 终止并在代理服务器上拥有您自己的 SSL 证书。

SSL 传递不起作用，因为提供的 SSL 证书与代理服务器主机名不匹配。

根 CA 证书

Neptune 实例的证书一般将通过操作系统的本地信任存储或 SDK（如 Java SDK）来进行验证。

如果您需要手动提供根证书，可以从 Amazon 信任服务策略存储库中下载 PEM 格式的 Amazon 根 CA 证书。

更多信息

有关使用 SSL 连接到 Neptune 端点的更多信息，请参阅the section called “安装 Gremlin 控制台”和the section called “HTTP REST”。

静态加密 Neptune 资源

Neptune 加密的实例通过防止对基础存储进行未经授权的访问来帮助保护您的数据，提供了额外的一层数据保护。您可以使用 Neptune 加密来增强部署在云中的应用程序的数据保护。您还可以使用它来满足 data-at-rest 加密的合规性要求。

要管理用于加密和解密 Neptune 资源的密钥，可以使用 ()。AWS Key Management ServiceAWS KMS AWS KMS 将安全、高度可用的硬件和软件相结合，提供可扩展到云端的密钥管理系统。使用 AWS KMS，您可以创建加密密钥并定义控制如何使用这些密钥的策略。 AWS KMS 支持 AWS CloudTrail，因此您可以审核密钥使用情况，以验证密钥的使用是否正确。您可以将 AWS KMS 密钥与

Neptune 和支持的 AWS 服务结合使用，例如亚马逊简单存储服务 (Amazon S3)、亚马逊弹性区块存储 (Amazon EBS) Block Store 和 Amazon Redshift。有关支持的服务列表 AWS KMS，请参阅《AWS Key Management Service 开发人员指南》 AWS KMS中的AWS 服务使用方式。

可为 Neptune 加密的实例加密所有日志、备份和快照。

为 Neptune 数据库实例启用加密

要为新的 Neptune 数据库实例启用加密，请在 Neptune 控制台的启用加密部分选择是。有关创建 Neptune 数据库实例的信息，请参阅创建新的 Neptune 数据库集群。

在创建加密的 Neptune 数据库实例时，您还可以提供加密 AWS KMS 密钥的密钥标识符。如果您未指定 AWS KMS 密钥标识符，Neptune 会将您的默认 Amazon RDS 加密密钥 (`aws/rds`) 用于您的新 Neptune 数据库实例。 AWS KMS 为您的账户创建 Neptune 的默认加密密钥。 AWS 您的 AWS 账户在每个 AWS 区域都有不同的默认加密密钥。

创建加密的 Neptune 数据库实例后，您无法更改该实例的加密密钥。因此，请确保先确定您的加密密钥要求，然后再创建加密的 Neptune 数据库实例。

可使用另一账户中的密钥的 Amazon 资源名称 (ARN) 来加密 Neptune 数据库实例。如果您使用拥有用于加密新 Neptune 数据库实例的 AWS KMS 加密密钥的相同 AWS 账户创建 Neptune 数据库实例，则您传递的 AWS KMS 密钥 ID 可以是密钥别名，而不是 AWS KMS 密钥的 ARN。

> **Important**
>
> 如果 Neptune 失去对 Neptune 数据库实例的加密密钥的访问权限（例如，在撤销对密钥的 Neptune 访问权限时），该加密的数据库实例将被置于最终状态且只能从备份进行还原。我们强烈建议您始终对加密的 Neptune 数据库实例启用备份，以防止数据库中的加密数据丢失。

启用加密时所需的密钥权限

创建加密 Neptune 数据库实例的 IAM 用户或角色必须至少拥有 KMS 密钥的以下权限：

- `"kms:Encrypt"`
- `"kms:Decrypt"`
- `"kms:GenerateDataKey"`
- `"kms:ReEncryptTo"`
- `"kms:GenerateDataKeyWithoutPlaintext"`

- "kms:CreateGrant"
- "kms:ReEncryptFrom"
- "kms:DescribeKey"

以下是包含必要权限的密钥策略示例：

```
{
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-3",
  "Statement": [
    {
      "Sid": "Enable Permissions for root principal",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key for Neptune",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:GenerateDataKey",
        "kms:ReEncryptTo",
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:CreateGrant",
        "kms:ReEncryptFrom",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "rds.us-east-1.amazonaws.com"
        }
      }
    },
```

```
    {
      "Sid": "Deny use of the key for non Neptune",
      "Effect": "Deny",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/NeptuneFullAccess"
      },
      "Action": [
        "kms:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "kms:ViaService": "rds.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

- 本策略中的第一条语句是可选的。它允许访问用户的根主体。
- 第二条语句提供对该角色所有必需 AWS KMS 的 API 的访问权限，其范围仅限于 RDS 服务主体。
- 第三个语句强制该角色不能将此密钥用于任何其他 AWS 服务，从而进一步加强了安全性。

也可以通过添加以下内容来进一步缩小 `createGrant` 权限范围：

```
"Condition": {
  "Bool": {
    "kms:GrantIsForAWSResource": true
  }
}
```

Neptune 加密的限制

加密 Neptune 集群存在以下限制：

- 您无法将未加密的数据库集群转换为加密的数据库集群。

  但是，您可以将未加密的 数据库集群快照还原为加密的 数据库集群。为此，请在从未加密的数据库集群快照还原时指定 KMS 加密密钥。
- 您无法将未加密的数据库实例转换为加密的数据库实例。您只能在创建数据库实例时为其启用加密。

- 此外，不能将已加密的数据库实例修改为禁用加密。
- 您无法拥有未加密数据库实例的加密只读副本或加密数据库实例的未加密只读副本。
- 加密的只读副本的加密密钥必须用与源数据库实例的相同。

亚马逊 Neptune 中的 AWS Identity and Access Management (IAM) 概述

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制可以通过身份验证（登录）和授权（具有权限）使用 Neptune 资源的人员。您可以使用 IAM AWS 服务 ，无需支付额外费用。

您可以使用 AWS Identity and Access Management (IAM) 对您的 Neptune 数据库实例或数据库集群进行身份验证。启用 IAM 数据库身份验证后，必须使用签 AWS 名版本 4 对每个请求进行签名。

AWS 签名版本 4 将身份验证信息添加到 AWS 请求中。出于安全考虑，对启用了 IAM 身份验证的 Neptune 数据库集群的所有请求必须使用访问密钥签名。此密钥由访问密钥 ID 和私有访问密钥组成。身份验证在外部是使用 IAM policy 进行管理的。

Neptune 在连接时进行身份验证，对于 WebSockets 连接，它会定期验证权限以确保用户仍然可以访问。

> **Note**
>
> - 建议不要撤消、删除或轮换与 IAM 用户关联的凭证，因为它不会终止任何已打开的连接。
> - 每个数据库实例的并发 WebSocket 连接数以及连接可以保持打开状态的时间都有限制。有关更多信息，请参阅 WebSockets 限制。

IAM 的用法取决于您的角色

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 Neptune 中所做的工作。

服务用户 – 如果使用 Neptune 服务来完成任务，则管理员会为您提供使用 Neptune 数据面板所需的凭证和权限。由于您需要更多的访问权限来完成您的工作，因此了解如何管理数据访问权限可帮助您从管理员请求适合的权限。

服务管理员 – 如果您在公司负责管理 Neptune 资源，则您可能可以访问 Neptune 管理操作，这些操作与 Neptune 管理 API 相对应。确定服务用户需要哪些 Neptune 数据访问操作和资源才能完成其工作也可能是您的任务。然后，IAM 管理员可以应用 IAM policy 来更改您的服务用户的权限。

IAM 管理员 – 如果您是 IAM 管理员，您需要编写 IAM policy 来管理 Neptune 的管理和数据访问。要查看您可以使用的基于 Neptune 身份的示例策略，请参阅使用不同类型的 IAM policy 控制对 Neptune 的访问权限。

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的 多重身份验证和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定

的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅 IA M 用户指南中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Simple Storage Service (Amazon S3) 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他

AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。

- 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
- 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

在 Neptune 中启用 IAM 数据库身份验证

默认情况下，创建 Amazon Neptune 数据库集群时禁用了 IAM 数据库身份验证。您可以使用 AWS Management Console启用 IAM 数据库身份验证（或再次禁用它）。

要通过控制台创建使用 IAM 身份验证的新 Neptune 数据库集群，请按照使用 AWS Management Console启动 Neptune 数据库集群中的说明创建 Neptune 数据库集群。

在创建过程的第二页上，对于启用 IAM 数据库身份验证，选择是。

为现有数据库实例或集群启用或禁用 IAM 身份验证

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择集群。
3. 选择要修改的 Neptune 数据库集群，然后选择集群操作。然后选择 Modify cluster (修改集群)。
4. 在 Database options (数据库选项) 部分中，对于 IAM DB Authentication (IAM 数据库身份验证)，选择 Enable IAM DB authorization (启用 IAM 数据库授权) 或 No (否)（禁用）。然后选择 Continue (继续)。
5. 要立即应用更改，请选择立即应用。

6. 选择修改集群。

使用签名版本 4 进行连接和 AWS 签名

启用了 IAM 数据库身份验证的 Amazon Neptune 资源要求使用签 AWS 名版本 4 对所有 HTTP 请求进行签名。有关使用签 AWS 名版本 4 对请求进行签名的一般信息，请参阅签署 AWS API 请求。

AWS 签名版本 4 是向 AWS 请求添加身份验证信息的过程。为了安全起见，对的大多数请求都 AWS 必须使用访问密钥进行签名，访问密钥由访问密钥 ID 和私有访问密钥组成。

> **Note**
>
> 如果您使用的是临时凭证，这些凭证（包括会话令牌）将在指定时间间隔后到期。
> 在请求新凭证时，您必须更新您的会话令牌。有关更多信息，请参阅使用临时安全证书请求对 AWS 资源的访问权限。

> **Important**
>
> 使用基于 IAM 的身份验证访问 Neptune 需要创建 HTTP 请求并自己对这些请求签名。

签名版本 4 的工作原理

1. 您创建一个规范请求。
2. 您可以使用规范请求和其他一些信息来创建. string-to-sign
3. 您可以使用私有访问 AWS 密钥派生签名密钥，然后使用该签名密钥和创建签名。 string-to-sign
4. 您将生成的签名添加到 HTTP 请求的标头中或者作为查询字符串参数添加。

Neptune 收到请求后，将执行完成的相同步骤来计算签名。之后，Neptune 会将计算得到的签名与在请求中发送的签名进行比较。如果签名匹配，则处理请求。如果签名不匹配，则拒绝请求。

有关使用签 AWS 名版本 4 对请求进行签名的一般信息，请参阅中的签名版本 4 签名流程AWS 一般参考。

以下部分包含示例，演示如何将签名请求发送到启用了 IAM 身份验证的 Neptune 数据库实例的 Gremlin 和 SPARQL 端点。

主题

Amazon Linux EC2 上的先决条件

以下示例说明如何在 Amazon EC2 实例上安装 Apache Maven 和 Java 8。这些是 Amazon Neptune 签名版本 4 身份验证示例所必需的。

在 EC2 实例上安装 Apache Maven 和 Java 8

1. 使用 SSH 客户端连接到您的 Amazon EC2 实例。
2. 在您的 EC2 实例上安装 Apache Maven。首先，输入以下命令以添加具有 Maven 程序包的存储库。

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   输入以下命令以设置该程序包的版本号。

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

   然后，可以使用 yum 安装 Maven。

   ```
   sudo yum install -y apache-maven
   ```
3. Gremlin 库需要 Java 8。输入以下命令以在 EC2 实例上安装 Java 8。

   ```
   sudo yum install java-1.8.0-devel
   ```
4. 输入以下命令以在 EC2 实例上将 Java 8 设置为默认运行时系统。

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

在系统提示时，输入 Java 8 的版本号。

5. 输入以下命令以在 EC2 实例上将 Java 8 设置为默认编译器。

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

在系统提示时，输入 Java 8 的版本号。

使用命令行工具向您的 Neptune 数据库集群提交查询

如本文档中的许多示例所示，拥有用于向 Neptune 数据库集群提交查询的命令行工具非常方便。如果未启用 IAM 身份验证，则 curl 工具是与 Neptune 端点通信的上佳选择。

但是，为了确保您的数据安全，最好启用 IAM 身份验证。

启用 IAM 身份验证后，必须使用签名版本 4 (Sig4) 对每个请求进行签名。第三方 awscurl 命令行工具使用与 `curl` 相同的语法，并且可以使用 Sig4 签名对查询进行签名。使用 `awscurl`部分解释了如何将 `awscurl` 安全地与临时凭证结合使用。

设置命令行工具以使用 HTTPS

Neptune 要求所有连接都使用 HTTPS。任何像 `curl` 或 `awscurl` 这样的命令行工具都需要访问相应的证书才能使用 HTTPS。只要 `curl` 或 `awscurl` 可以找到相应的证书，它们即可像处理 HTTP 连接一样处理 HTTPS 连接，而无需额外的参数。本文档中的示例基于该场景。

要了解如何获取此类证书以及如何将其正确格式化为 `curl` 可以使用的证书颁发机构 (CA) 证书存储，请参阅 `curl` 文档中的 SSL 证书验证。

然后，您使用 `CURL_CA_BUNDLE` 环境变量指定此 CA 证书存储的位置。在 Windows 上，`curl` 自动在名为 `curl-ca-bundle.crt` 的文件中查找它。首先在与 `curl.exe` 相同的目录中查找，然后在路径的其他位置查找。有关更多信息，请参阅 SSL 证书验证。

将 **`awscurl`** 与临时凭证结合使用以安全地连接到启用了 IAM 身份验证的数据库集群

awscurl 工具使用与 `curl` 相同的语法，但还需要其它信息：

- **`--access_key`** – 有效的访问密钥。如果未使用此参数提供，则必须在 `AWS_ACCESS_KEY_ID` 环境变量或配置文件中提供它。

- **`--secret_key`** – 对应于访问密钥的有效秘密密钥。如果未使用此参数提供，则必须在 `AWS_SECRET_ACCESS_KEY` 环境变量或配置文件中提供它。
- **`--security_token`** – 有效的会话令牌。如果未使用此参数提供，则必须在 `AWS_SECURITY_TOKEN` 环境变量或配置文件中提供它。

过去，通常的做法是将永久凭证与 `awscurl` 结合使用，例如 IAM 用户凭证甚至根凭证，但不建议这样做。而是使用 AWS 安全令牌服务 (STS) API 之一或其 AWS CLI 包装器之一生成临时凭证。

最好将 STS 调用返回的 `AccessKeyId`、`SecretAccessKey` 和 `SessionToken` 值放入 Shell 会话的相应环境变量中，而不是放在配置文件中。然后，当 Shell 终止时，凭证会被自动丢弃，而配置文件则不是这样。同样，请求的临时凭证期限也不要超过您可能需要的期限。

以下示例显示了您可以在 Linux Shell 中使用 sts assume-role 获取有效期为半小时的临时凭证，然后将它们放在 `awscurl` 可以找到它们的环境变量中的步骤：

```
aws sts assume-role \
    --duration-seconds 1800 \
    --role-arn "arn:aws:iam::(account-id):role/(rolename)" \
    --role-session-name AWSCLI-Session > $output
AccessKeyId=$(cat $output | jq '.Credentials''.AccessKeyId')
SecretAccessKey=$(cat $output | jq '.Credentials''.SecretAccessKey')
SessionToken=$(cat $output | jq '.Credentials''.SessionToken')

export AWS_ACCESS_KEY_ID=$AccessKeyId
export AWS_SECRET_ACCESS_KEY=$SecretAccessKey
export AWS_SESSION_TOKEN=$SessionToken
```

然后，您可以使用 `awscurl` 向数据库集群发出签名请求，如下所示：

```
awscurl (your cluster endpoint):8182/status \
    --region us-east-1 \
    --service neptune-db
```

使用 Gremlin 控制台和签名版本 4 签名连接到 Neptune

使用带有签名版本 4 身份验证的 Gremlin 控制台连接到 Amazon Neptune 的方式取决于您使用的是版本还是更高 TinkerPop 版本`3.4.11`，还是更早的版本。无论哪种情况，都必须满足以下先决条件：

- 您必须具有对请求进行签名所需的 IAM 凭证。请参阅《AWS SDK for Java 开发者指南》中的使用默认凭证提供商链。

- 您已安装的 Gremlin 控制台版本必须与数据库集群正在使用的 Neptune 引擎版本兼容。

如果您使用的是临时凭证，则它们将在指定的间隔后过期，会话令牌也是如此，因此在请求新凭证时必须更新会话令牌。请参阅 IAM 用户指南中的使用临时安全证书请求 AWS 资源访问权限。

有关使用 SSL/TLS 帮助进行连接的帮助，请参阅SSL/TLS 配置。

使用 TinkerPop 3.4.11 或更高版本通过 Sig4 签名连接到 Neptune

在 TinkerPop 3.4.11 或更高版本中，您将使用`handshakeInterceptor()`，它提供了一种将 Sigv4 签名者插入命令建立的连接的方法。`:remote`与用于 Java 的方法一样，它要求您手动配置 `Cluster` 对象，然后将其传递给 `:remote` 命令。

请注意，这与 `:remote` 命令使用配置文件来形成连接的典型情况大不相同。配置文件方法不起作用，因为必须以编程方式设置 `handshakeInterceptor()`，并且无法从文件加载其配置。

使用 Sig4 签名连接 Gremlin 主机（TinkerPop 3.4.11 及更高版本）

1. 启动 Gremlin 控制台：

   ```
   $ bin/gremlin.sh
   ```

2. 在 `gremlin>` 提示符下，安装 `amazon-neptune-sigv4-signer` 库（对于控制台，此操作只需执行一次）：

   ```
   :install com.amazonaws amazon-neptune-sigv4-signer 2.4.0
   ```

   如果您在此步骤中遇到问题，请参考有关 Gr ap e 配置的TinkerPop 文档可能会有所帮助。

   > **Note**
   >
   > 如果您使用的是 HTTP 代理，则在此步骤中可能会遇到 `:install` 命令未完成的错误。要解决此问题，请运行以下命令来告知控制台有关代理的信息：
   >
   > ```
   > System.setProperty("https.proxyHost", "(the proxy IP address)")
   > System.setProperty("https.proxyPort", "(the proxy port)")
   > ```

3. 将处理签名所需的类导入到 `handshakeInterceptor()` 中：

```
:import com.amazonaws.auth.DefaultAWSCredentialsProviderChain
:import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer
```

4. 如果您使用的是临时凭证，则还需要按以下方式提供会话令牌：

```
System.setProperty("aws.sessionToken","(your session token)")
```

5. 如果您尚未以其它方式建立账户凭证，则可以按以下方式分配凭证：

```
System.setProperty("aws.accessKeyId","(your access key)")
System.setProperty("aws.secretKey","(your secret key)")
```

6. 手动构造要连接到 Neptune 的 `Cluster` 对象：

```
cluster = Cluster.build("(host name)")  \
                 .enableSsl(true) \
                 .handshakeInterceptor { r ->  \
                   def sigV4Signer = new NeptuneNettyHttpSigV4Signer("(Amazon
 region)", \
                                       new DefaultAWSCredentialsProviderChain()); \
                   sigV4Signer.signRequest(r); \
                   return r; } \
                 .create()
```

有关查找 Neptune 数据库实例的主机名的帮助，请参阅连接到 Amazon Neptune 端点。

7. 使用上一步中 `Cluster` 对象的变量名建立 `:remote` 连接：

```
:remote connect tinkerpop.server cluster
```

8. 输入以下命令以切换到远程模式。这会将所有 Gremlin 查询发送到远程连接：

```
:remote console
```

使用 3.4.11 TinkerPop 之前的版本通过 Sig4 签名连接到 Neptune

在 TinkerPop 3.4.10 或更低版本中，使用 Neptune 提供的`amazon-neptune-gremlin-java-sigv4`库通过 Sigv4 签名将主机连接到 Neptune，如下所述：

使用 Sig4 签名连接 Gremlin 主机（3.4.11 之前的TinkerPop 版本）

1. 启动 Gremlin 控制台：

```
$ bin/gremlin.sh
```

2. 在 `gremlin>` 提示符下，安装 `amazon-neptune-sigv4-signer` 库（对于控制台，此操作只需执行一次）：

```
:install com.amazonaws amazon-neptune-sigv4-signer 2.4.0
```

> **Note**
>
> 如果您使用的是 HTTP 代理，则在此步骤中可能会遇到 `:install` 命令未完成的错误。要解决此问题，请运行以下命令来告知控制台有关代理的信息：
>
> ```
> System.setProperty("https.proxyHost", "(the proxy IP address)")
> System.setProperty("https.proxyPort", "(the proxy port)")
> ```
>
> 查阅有关 Gr ap e 配置的TinkerPop 文档也可能有所帮助。

3. 在提取的目录的 `conf` 子目录中，创建名为 `neptune-remote.yaml` 的文件。

   如果您使用该 AWS CloudFormation 模板创建 Neptune 数据库集群，则`neptune-remote.yaml`文件将已经存在。在这种情况下，您所要做的就是编辑现有文件，以包含如下所示的频道选择器设置。

   否则，请将以下文本复制到文件中，同时将 *(host name)* 替换为 Neptune 数据库实例的主机名或 IP 地址。请注意，必须使用方括号 ([ ]) 包含主机名。

```
hosts: [(host name)]
port: 8182
connectionPool: {
  channelizer: org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer,
  enableSsl: true
}
serializer: { className:
 org.apache.tinkerpop.gremlin.driver.ser.GraphBinaryMessageSerializerV1,
              config: { serializeResultToString: true }}
```

4.

> **Important**
>
> 您必须提供 IAM 凭证才能对请求进行签名。输入以下命令以将您的凭证设置为环境变量，替换与您的凭证相关的项目。
>
> ```
> export AWS_ACCESS_KEY_ID=access_key_id
> export AWS_SECRET_ACCESS_KEY=secret_access_key
> export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or
>  ca-central-1 or
>                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or
>  eu-west-3 or eu-central-1 or me-south-1 or
>                      me-central-1 or il-central-1 or af-south-1 or
>  ap-east-1 or ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-
> southeast-2 or ap-south-1 or
>                      cn-north-1 or cn-northwest-1 or
>                      us-gov-east-1 or us-gov-west-1
> ```
>
> Neptune 版本 4 签名程序使用默认的凭证提供程序链。有关提供凭证的其它方法，请参阅《AWS SDK for Java 开发人员指南》中的使用默认凭证提供程序链。
> SERVICE_REGION 变量是必需的，即使在使用凭证文件时也是如此。

5. 使用 `.yaml` 文件建立 `:remote` 连接：

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

6. 输入以下命令切换到远程模式，该模式会将所有 Gremlin 查询发送到远程连接：

```
:remote console
```

使用 Java 和 Gremlin 及签名版本 4 签名连接到 Neptune

使用 TinkerPop 3.4.11 或更高版本通过 Sig4 签名连接到 Neptune

以下是使用 TinkerPop 3.4.11 或更高版本时如何使用带有 Sig4 签名的 Gremlin Java API 连接到 Neptune 的示例（它假设对使用 Maven 有常识）。首先，将依赖关系定义为 `pom.xml` 文件的一部分：

```
<dependency>
  <groupId>com.amazonaws</groupId>
```

```
  <artifactId>amazon-neptune-sigv4-signer</artifactId>
  <version>2.4.0</version>
</dependency>
```

然后，使用如下代码：

```
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer;
import com.amazonaws.neptune.auth.NeptuneSigV4SignerException;

 ...

System.setProperty("aws.accessKeyId","your-access-key");
System.setProperty("aws.secretKey","your-secret-key");

 ...

Cluster cluster = Cluster.build((your cluster))
                 .enableSsl(true)
                 .handshakeInterceptor( r ->
                  {
                    try {
                      NeptuneNettyHttpSigV4Signer sigV4Signer =
                        new NeptuneNettyHttpSigV4Signer("(your region)", new
 DefaultAWSCredentialsProviderChain());
                      sigV4Signer.signRequest(r);
                    } catch (NeptuneSigV4SignerException e) {
                      throw new RuntimeException("Exception occurred while signing the
 request", e);
                    }
                    return r;
                  }
                 ).create();
try {
  Client client = cluster.connect();
  client.submit("g.V().has('code','IAD')").all().get();
} catch (Exception e) {
  throw new RuntimeException("Exception occurred while connecting to cluster", e);
}
```

> **Note**
>
> 如果要从 3.4.11 升级，请移除对 amazon-neptune-gremlin-java-sigv4 库的引用。如上例所示，使用 handshakeInterceptor() 时不再需要它。不要尝试将 handshakeInterceptor() 与通道选择器 (SigV4WebSocketChannelizer.class) 结合使用，因为它会产生错误。

使用 3.4.11 TinkerPop 之前的版本通过 Sig4 签名连接到 Neptune

TinkerPop 之前的版本3.4.11不支持上一节中显示的handshakeInterceptor()配置，因此必须依赖该amazon-neptune-gremlin-java-sigv4软件包。这是一个包含该类的 Neptune 库，它用可以自动注入 SigV4 签名的SigV4WebSocketChannelizer类取代了标准的 TinkerPop Channelizer。如果可能，请升级到 TinkerPop 3.4.11 或更高版本，因为该amazon-neptune-gremlin-java-sigv4库已被弃用。

以下是使用 3.4.11 之前的 TinkerPop 版本时如何使用带有 Sig4 签名的 Gremlin Java API 连接到 Neptune 的示例（它假设对如何使用 Maven 有常识）。

首先，将依赖关系定义为 pom.xml 文件的一部分：

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>amazon-neptune-gremlin-java-sigv4</artifactId>
  <version>2.4.0</version>
</dependency>
```

上面的依赖关系将包括 Gremlin 驱动程序版本 3.4.10。尽管可以使用较新的 Gremlin 驱动程序版本（直到 3.4.13），但 3.4.10 版本之后的驱动程序升级应包括使用上述 handshakeInterceptor() 模型的更改。

然后，应在 Java 代码中按如下方式配置 gremlin-driver 集群对象：

```
import org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer;

 ...

Cluster cluster = Cluster.build(your cluster)
                        .enableSsl(true)
                        .channelizer(SigV4WebSocketChannelizer.class)
```

```
                        .create();
Client client = cluster.connect();
client.submit("g.V().has('code','IAD')").all().get();
```

使用 Java 和 SPARQL 以及签名版本 4 签名（RDF4J 和 Jena）连接到 Neptune

本节介绍如何使用具有签名版本 4 身份验证的 RDF4J 或 Apache Jena 连接到 Neptune。

先决条件

- Java 8 或更高版本。
- Apache Maven 3.3 或更高版本。

  有关在运行 Amazon Linux 的 EC2 实例上安装这些先决条件的信息，请参阅 Amazon Linux EC2 上的先决条件。
- 用于对请求进行签名的 IAM 凭证。有关更多信息，请参阅《AWS SDK for Java 开发人员指南》中的使用默认凭证提供程序链。

  > **Note**
  >
  > 如果您使用的是临时凭证，这些凭证（包括会话令牌）将在指定时间间隔后到期。
  > 在请求新凭证时，您必须更新您的会话令牌。有关更多信息，请参阅 IAM 用户指南中的使用临时安全证书请求 AWS 资源访问权限。
- 将 SERVICE_REGION 变量设置为以下内容之一，以指示 Neptune 数据库实例的区域：
  - 美国东部（弗吉尼亚州北部）：us-east-1
  - 美国东部（俄亥俄州）：us-east-2
  - 美国西部（北加利福尼亚）：us-west-1
  - 美国西部（俄勒冈州）：us-west-2
  - 加拿大（中部）：ca-central-1
  - 南美洲（圣保罗）：sa-east-1
  - 欧洲地区（斯德哥尔摩）：eu-north-1
  - 欧洲地区（爱尔兰）：eu-west-1
  - 欧洲地区（伦敦）：eu-west-2
  - 欧洲地区（巴黎）：eu-west-3

- 欧洲地区（法兰克福）：eu-central-1
- 中东（巴林）：me-south-1
- 中东（阿联酋）：me-central-1
- 以色列（特拉维夫）：il-central-1
- 非洲（开普敦）：af-south-1
- 亚太地区（香港）：ap-east-1
- 亚太地区（东京）：ap-northeast-1
- 亚太地区（首尔）：ap-northeast-2
- 亚太地区（大阪）：ap-northeast-3
- 亚太地区（新加坡）：ap-southeast-1
- 亚太地区（悉尼）：ap-southeast-2
- 亚太地区（孟买）：ap-south-1
- 中国（北京）：cn-north-1
- 中国（宁夏）：cn-northwest-1
- AWS GovCloud （美国西部）：us-gov-west-1
- AWS GovCloud （美国东部）：us-gov-east-1

使用 RDF4J 或 Apache Jena 以及签名版本 4 签名连接到 Neptune

1. 从中克隆示例存储库 GitHub。

```
git clone https://github.com/aws/amazon-neptune-sparql-java-sigv4.git
```

2. 更改到克隆的目录中。

```
cd amazon-neptune-sparql-java-sigv4
```

3. 通过检出具有最新标签的分支获取项目的最新版本。

```
git checkout $(git describe --tags `git rev-list --tags --max-count=1`)
```

4. 输入以下命令之一以编译和运行示例代码。

   将 *your-neptune-endpoint* 替换为 Neptune 数据库实例的主机名或 IP 地址。默认端口为 8182。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点部分。

Eclipse RDF4J

输入以下内容以运行 RDF4J 示例。

```
mvn compile exec:java \
    -Dexec.mainClass="com.amazonaws.neptune.client.rdf4j.NeptuneRdf4JSigV4Example"
 \
    -Dexec.args="https://your-neptune-endpoint:port"
```

Apache Jena

输入以下内容以运行 Apache Jena 示例。

```
mvn compile exec:java \
    -Dexec.mainClass="com.amazonaws.neptune.client.jena.NeptuneJenaSigV4Example" \
    -Dexec.args="https://your-neptune-endpoint:port"
```

5. 要查看示例的源代码，请参阅 `src/main/java/com/amazonaws/neptune/client/` 目录中的示例。

要在您自己的 Java 应用程序中使用 SigV4 签名驱动程序，请将 `amazon-neptune-sigv4-signer` Maven 程序包添加到 `pom.xml` 的 `<dependencies>` 部分。我们建议您从示例开始着手。

使用 SPARQL 和 Node.js 以及签名版本 4 签名连接到 Neptune

使用签名 V4 签名和 Javascript V3 AWS 开发工具包进行查询

以下是如何使用带有签名版本 4 身份验证的 Node.js 和适用于 Javascript V3 的 S AWS DK 连接到 Neptune SPARQL 的示例：

```
const { HttpRequest }  = require('@smithy/protocol-http');
const { fromNodeProviderChain } = require('@aws-sdk/credential-providers');
const { SignatureV4 } = require('@smithy/signature-v4');
```

```
const { Sha256 } = require('@aws-crypto/sha256-universal');
const https = require('https');

var region = 'us-west-2'; // e.g. us-west-1
var neptune_endpoint = 'your-Neptune-cluster-endpoint';  // like: 'cluster-
id.region.neptune.amazonaws.com'
var query = `query=PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX class: <http://aws.amazon.com/neptune/csv2rdf/class/>
PREFIX resource: <http://aws.amazon.com/neptune/csv2rdf/resource/>
PREFIX prop: <http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/>
PREFIX objprop: <http://aws.amazon.com/neptune/csv2rdf/objectProperty/>

SELECT ?movies ?title WHERE {
  ?jel prop:name "James Earl Jones" .
  ?movies ?p2 ?jel .
  ?movies prop:title ?title
} LIMIT 10`;

runQuery(query);

function runQuery(q) {
  var request = new HttpRequest({
    hostname: neptune_endpoint,
    port: 8182,
    path: 'sparql',
    body: encodeURI(query),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'host': neptune_endpoint + ':8182',
    },
    method: 'POST',
  });

  const credentialProvider = fromNodeProviderChain();
  let credentials = credentialProvider();
  credentials.then(
    (cred)=>{
      var signer = new SignatureV4({credentials: cred, region: region, sha256: Sha256,
 service: 'neptune-db'});
      signer.sign(request).then(
        (req)=>{
          var responseBody = '';
          var sendreq = https.request(
            {
```

```
              host: req.hostname,
              port: req.port,
              path: req.path,
              method: req.method,
              headers: req.headers,
            },
          (res) => {
            res.on('data', (chunk) => { responseBody += chunk; });
            res.on('end', () => {
                console.log(JSON.parse(responseBody));
            });
          });
          sendreq.write(req.body);
          sendreq.end();
        }
      );
    },
    (err)=>{
      console.error(err);
    }
  );
}
```

使用签名 V4 签名和适用于 Javascript V2 的 AWS SDK 进行查询

以下是如何使用带签名版本 4 身份验证的 Node.js 和适用于 Javascript V2 的 S AWS DK 连接到 Neptune SPARQL 的示例：

```
var AWS = require('aws-sdk');

var region = 'us-west-2'; // e.g. us-west-1
var neptune_endpoint = 'your-Neptune-cluster-endpoint';  // like: 'cluster-
id.region.neptune.amazonaws.com'
var query = `query=PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX class: <http://aws.amazon.com/neptune/csv2rdf/class/>
PREFIX resource: <http://aws.amazon.com/neptune/csv2rdf/resource/>
PREFIX prop: <http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/>
PREFIX objprop: <http://aws.amazon.com/neptune/csv2rdf/objectProperty/>

SELECT ?movies ?title WHERE {
    ?jel prop:name "James Earl Jones" .
    ?movies ?p2 ?jel .
    ?movies prop:title ?title
```

```
} LIMIT 10`;

runQuery(query);

function runQuery(q) {

    var endpoint = new AWS.Endpoint(neptune_endpoint);
    endpoint.port = 8182;
    var request = new AWS.HttpRequest(endpoint, region);
    request.path += 'sparql';
    request.body = encodeURI(query);
    request.headers['Content-Type'] = 'application/x-www-form-urlencoded';
    request.headers['host'] = neptune_endpoint;
    request.method = 'POST';

    var credentials = new AWS.CredentialProviderChain();
    credentials.resolve((err, cred)=>{
        var signer = new AWS.Signers.V4(request, 'neptune-db');
        signer.addAuthorization(cred, new Date());
    });

    var client = new AWS.HttpClient();
    client.handleRequest(request, null, function(response) {
        console.log(response.statusCode + ' ' + response.statusMessage);
        var responseBody = '';
        response.on('data', function (chunk) {
            responseBody += chunk;
        });
        response.on('end', function (chunk) {
            console.log('Response body: ' + responseBody);
        });
    }, function(error) {
        console.log('Error: ' + error);
    });
}
```

示例：使用 Python 及签名版本 4 签名连接到 Neptune

本部分演示使用 Python 编写的程序示例，此示例阐释如何对 Amazon Neptune 使用签名版本 4。此示例基于 https://docs.aws.amazon.com/general/latest/gr/sigv4-signed-request-examples.html 中 Amazon Web Services 一般参考签名版本 4 签名过程部分中的示例。

要使用此示例程序，您需要：

- 计算机上安装有 Python 3.x，您可以从 Python 站点获取。这些程序已使用 Python 3.6 测试过。
- Python 请求库，示例脚本使用此库发出 Web 请求。一种方便的 Python 程序包安装方法是使用 `pip`，它可从 Python 程序包索引站点获取程序包。然后，您可以在命令行上运行 `requests` 来安装 `pip install requests`。
- 环境变量中名为 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 的访问密钥（访问密钥 ID 和私有访问密钥）。作为最佳实践，我们建议您不要在代码中嵌入凭证。有关更多信息，请参阅《AWS Account Management 参考指南》中的 AWS 账户最佳实践。

  名为 `SERVICE_REGION` 的环境变量中的 Neptune 数据库集群的区域。

  如果您使用的是临时凭证，则除了 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `SERVICE_REGION` 之外，您还必须指定 `AWS_SESSION_TOKEN`。

> **Note**
>
> 如果您使用的是临时凭证，这些凭证（包括会话令牌）将在指定时间间隔后到期。
> 在请求新凭证时，您必须更新您的会话令牌。有关更多信息，请参阅使用临时安全凭证以请求对 AWS 资源的访问权限。

以下示例演示如何使用 Python 对 Neptune 发出签名请求。此请求发出一个 GET 或 POST 请求。身份验证信息是通过 `Authorization` 请求标头传递的。

这个例子也可以作为一个 AWS Lambda 函数使用。有关更多信息，请参阅 the section called “设置 Lambda”。

对 Gremlin 和 SPARQL Neptune 端点发出签名请求

1. 创建一个名为 `neptunesigv4.py` 的新文件并在文本编辑器中打开它。
2. 复制以下代码并将其粘贴到 `neptunesigv4.py` 文件中：

```
# Amazon Neptune version 4 signing example (version v3)

# The following script requires python 3.6+
#    (sudo yum install python36 python36-virtualenv python36-pip)
# => the reason is that we're using urllib.parse() to manually encode URL
#    parameters: the problem here is that SIGV4 encoding requires whitespaces
#    to be encoded as %20 rather than not or using '+', as done by previous/
#    default versions of the library.
```

```
# See: https://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
import sys, datetime, hashlib, hmac
import requests  # pip3 install requests
import urllib
import os
import json
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import ReadOnlyCredentials
from types import SimpleNamespace
from argparse import RawTextHelpFormatter
from argparse import ArgumentParser

# Configuration. https is required.
protocol = 'https'

# The following lines enable debugging at httplib level (requests->urllib3-
>http.client)
# You will see the REQUEST, including HEADERS and DATA, and RESPONSE with HEADERS
 but without DATA.
#
# The only thing missing will be the response.body which is not logged.
#
# import logging
# from http.client import HTTPConnection
# HTTPConnection.debuglevel = 1
# logging.basicConfig()
# logging.getLogger().setLevel(logging.DEBUG)
# requests_log = logging.getLogger("requests.packages.urllib3")
# requests_log.setLevel(logging.DEBUG)
# requests_log.propagate = True


# Read AWS access key from env. variables. Best practice is NOT
# to embed credentials in code.
access_key = os.getenv('AWS_ACCESS_KEY_ID', '')
secret_key = os.getenv('AWS_SECRET_ACCESS_KEY', '')
region = os.getenv('SERVICE_REGION', '')

# AWS_SESSION_TOKEN is optional environment variable. Specify a session token only
 if you are using temporary
# security credentials.
```

```
session_token = os.getenv('AWS_SESSION_TOKEN', '')

### Note same script can be used for AWS Lambda (runtime = python3.6).
## Steps to use this python script for AWS Lambda
# 1. AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY and AWS_SESSION_TOKEN and AWS_REGION
 variables are already part of Lambda's Execution environment
#    No need to set them up explicitly.
# 3. Create Lambda deployment package https://docs.aws.amazon.com/lambda/latest/dg/
lambda-python-how-to-create-deployment-package.html
# 4. Create a Lambda function in the same VPC and assign an IAM role with neptune
 access

def lambda_handler(event, context):
    # sample_test_input = {
    #     "host": "END_POINT:8182",
    #     "method": "GET",
    #     "query_type": "gremlin",
    #     "query": "g.V().count()"
    # }

    # Lambda uses AWS_REGION instead of SERVICE_REGION
    global region
    region = os.getenv('AWS_REGION', '')

    host = event['host']
    method = event['method']
    query_type = event['query_type']
    query =  event['query']

    return make_signed_request(host, method, query_type, query)

def validate_input(method, query_type):
    # Supporting GET and POST for now:
    if (method != 'GET' and method != 'POST'):
        print('First parameter must be "GET" or "POST", but is "' + method + '".')
        sys.exit()

    # SPARQL UPDATE requires POST
    if (method == 'GET' and query_type == 'sparqlupdate'):
        print('SPARQL UPDATE is not supported in GET mode. Please choose POST.')
        sys.exit()

def get_canonical_uri_and_payload(query_type, query, method):
    # Set the stack and payload depending on query_type.
```

```
    if (query_type == 'sparql'):
        canonical_uri = '/sparql/'
        payload = {'query': query}

    elif (query_type == 'sparqlupdate'):
        canonical_uri = '/sparql/'
        payload = {'update': query}

    elif (query_type == 'gremlin'):
        canonical_uri = '/gremlin/'
        payload = {'gremlin': query}
        if (method == 'POST'):
            payload = json.dumps(payload)

    elif (query_type == 'openCypher'):
        canonical_uri = '/openCypher/'
        payload = {'query': query}

    elif (query_type == "loader"):
        canonical_uri = "/loader/"
        payload = query

    elif (query_type == "status"):
        canonical_uri = "/status/"
        payload = {}

    elif (query_type == "gremlin/status"):
        canonical_uri = "/gremlin/status/"
        payload = {}

    elif (query_type == "openCypher/status"):
        canonical_uri = "/openCypher/status/"
        payload = {}

    elif (query_type == "sparql/status"):
        canonical_uri = "/sparql/status/"
        payload = {}

    else:
        print(
            'Third parameter should be from ["gremlin", "sparql", "sparqlupdate",
"loader", "status] but is "' + query_type + '".')
        sys.exit()
    ## return output as tuple
```

```
    return canonical_uri, payload

def make_signed_request(host, method, query_type, query):
    service = 'neptune-db'
    endpoint = protocol + '://' + host

    print()
    print('+++++ USER INPUT +++++')
    print('host = ' + host)
    print('method = ' + method)
    print('query_type = ' + query_type)
    print('query = ' + query)

    # validate input
    validate_input(method, query_type)

    # get canonical_uri and payload
    canonical_uri, payload = get_canonical_uri_and_payload(query_type, query,
 method)

    # assign payload to data or params
    data = payload if method == 'POST' else None
    params = payload if method == 'GET' else None

    # create request URL
    request_url = endpoint + canonical_uri

    # create and sign request
    creds = SimpleNamespace(
        access_key=access_key, secret_key=secret_key, token=session_token,
 region=region,
    )

    request = AWSRequest(method=method, url=request_url, data=data, params=params)
    SigV4Auth(creds, service, region).add_auth(request)

    r = None

    # ************* SEND THE REQUEST *************
    if (method == 'GET'):

        print('++++ BEGIN GET REQUEST +++++')
        print('Request URL = ' + request_url)
```

```
        r = requests.get(request_url, headers=request.headers, verify=False,
 params=params)

    elif (method == 'POST'):

        print('\n+++++ BEGIN POST REQUEST +++++')
        print('Request URL = ' + request_url)
        if (query_type == "loader"):
            request.headers['Content-type'] = 'application/json'
        r = requests.post(request_url, headers=request.headers, verify=False,
 data=data)

    else:
        print('Request method is neither "GET" nor "POST", something is wrong
 here.')

    if r is not None:
        print()
        print('+++++ RESPONSE +++++')
        print('Response code: %d\n' % r.status_code)
        response = r.text
        r.close()
        print(response)

        return response

help_msg = '''
    export AWS_ACCESS_KEY_ID=[MY_ACCESS_KEY_ID]
    export AWS_SECRET_ACCESS_KEY=[MY_SECRET_ACCESS_KEY]
    export AWS_SESSION_TOKEN=[MY_AWS_SESSION_TOKEN]
    export SERVICE_REGION=[us-east-1|us-east-2|us-west-2|eu-west-1]

    python version >=3.6 is required.

    Examples: For help
    python3 program_name.py -h

    Examples: Queries
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
```

```
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q
 sparqlupdate -d "INSERT DATA { <https://s> <https://p> <https://o> }"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d
 "g.V().count()"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d
 "g.V().count()"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/
status
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher
 -d "MATCH (n1) RETURN n1 LIMIT 1;"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher
 -d "MATCH (n1) RETURN n1 LIMIT 1;"
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{}'
    python3 program_name.py -ho your-neptune-endpoint -p 8182 -a POST -q loader
 -d '{"source": "source", "format" : "csv", "failOnError": "fail_on_error",
 "iamRoleArn": "iam_role_arn", "region": "region"}'

    Environment variables must be defined as AWS_ACCESS_KEY_ID,
 AWS_SECRET_ACCESS_KEY and SERVICE_REGION.
    You should also set AWS_SESSION_TOKEN environment variable if you are using
 temporary credentials (ex. IAM Role or EC2 Instance profile).

    Current Limitations:
    - Query mode "sparqlupdate" requires POST (as per the SPARQL 1.1 protocol)
            '''

def exit_and_print_help():
    print(help_msg)
    exit()

def parse_input_and_query_neptune():


    parser = ArgumentParser(description=help_msg,
 formatter_class=RawTextHelpFormatter)
    group_host = parser.add_mutually_exclusive_group()
    group_host.add_argument("-ho", "--host", type=str)
```

```
    group_port = parser.add_mutually_exclusive_group()
    group_port.add_argument("-p", "--port", type=int, help="port ex. 8182,
default=8182", default=8182)
    group_action = parser.add_mutually_exclusive_group()
    group_action.add_argument("-a", "--action", type=str, help="http action,
default = GET", default="GET")
    group_endpoint = parser.add_mutually_exclusive_group()
    group_endpoint.add_argument("-q", "--query_type", type=str, help="query_type,
default = status ", default="status")
    group_data = parser.add_mutually_exclusive_group()
    group_data.add_argument("-d", "--data", type=str, help="data required for the
http action", default="")

    args = parser.parse_args()
    print(args)

    # Read command line parameters
    host = args.host
    port = args.port
    method = args.action
    query_type = args.query_type
    query = args.data

    if (access_key == ''):
        print('!!! ERROR: Your AWS_ACCESS_KEY_ID environment variable is
undefined.')
        exit_and_print_help()

    if (secret_key == ''):
        print('!!! ERROR: Your AWS_SECRET_ACCESS_KEY environment variable is
undefined.')
        exit_and_print_help()

    if (region == ''):
        print('!!! ERROR: Your SERVICE_REGION environment variable is undefined.')
        exit_and_print_help()

    if host is None:
        print('!!! ERROR: Neptune DNS is missing')
        exit_and_print_help()

    host = host + ":" + str(port)
    make_signed_request(host, method, query_type, query)
```

```
if __name__ == "__main__":
    parse_input_and_query_neptune()
```

3. 在终端中，导航到 neptunesigv4.py 文件的位置。
4. 输入以下命令，使用正确的值替换访问密钥、私有密钥和区域。

```
export AWS_ACCESS_KEY_ID=MY_ACCESS_KEY_ID
export AWS_SECRET_ACCESS_KEY=MY_SECRET_ACCESS_KEY
export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

如果您使用的是临时凭证，则除了 AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY 和 SERVICE_REGION 之外，您还必须指定 AWS_SESSION_TOKEN。

```
export AWS_SESSION_TOKEN=MY_AWS_SESSION_TOKEN
```

> **Note**
>
> 如果您使用的是临时凭证，这些凭证（包括会话令牌）将在指定时间间隔后到期。在请求新凭证时，您必须更新您的会话令牌。有关更多信息，请参阅使用临时安全凭证以请求对 AWS 资源的访问权限。

5. 输入以下命令之一将签名请求发送给 Neptune 数据库实例。这些示例使用 Python 版本 3.6。

终端节点状态

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q status
```

Gremlin

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin -d
 "g.V().count()"
```

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q gremlin -d
 "g.V().count()"
```

Gremlin 状态

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q gremlin/
status
```

SPARQL

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql -d
 "SELECT ?s WHERE { ?s ?p ?o }"
```

SPARQL UPDATE

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q sparqlupdate
 -d "INSERT DATA { <https://s> <https://p> <https://o> }"
```

SPARQL 状态

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q sparql/status
```

openCypher

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher -d
 "MATCH (n1) RETURN n1 LIMIT 1;"

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q openCypher -
d "MATCH (n1) RETURN n1 LIMIT 1;"
```

openCypher 状态

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q openCypher/
status
```

加载程序

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{"loadId": "68b28dcc-8e15-02b1-133d-9bd0557607e6"}'

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a GET -q loader -d
 '{}'

python3.6 neptunesigv4.py -ho your-neptune-endpoint -p 8182 -a POST -q loader
 -d '{"source": "source", "format" : "csv", "failOnError": "fail_on_error",
 "iamRoleArn": "iam_role_arn", "region": "region"}'
```

6. 运行 Python 脚本的语法如下所示：

```
python3.6 neptunesigv4.py -ho your-neptune-endpoint -p port -a GET|POST -q gremlin|
sparql|sparqlupdate|loader|status -d "string0data"
```

SPARQL UPDATE 需要 POST。

使用 IAM policy 管理访问权限

IAM policy 是定义操作和资源使用权限的 JSON 对象。

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

对组织使用服务控制策略 (SCP) AWS

服务控制策略 (SCP) 是 JSON 策略，用于指定中AWS Organizations组织或组织单位 (OU) 的最大权限。 AWS Organizations 是一项用于对您的企业拥有的多个 AWS 账户进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体的权限，包括每个 AWS 账户的根用户。有关 Organizations 和 SCP 的更多信息，请参阅《 AWS Organizations 用户指南》中的 SCP 的工作原理。

在 AWS 组织内的 AWS 账户中部署 Amazon Neptune 的客户可以利用 SCP 来控制哪些账户可以使用 Neptune。为确保通过成员账户访问 Neptune，请务必分别使用 `neptune:*` 和 `neptune-db:*` 允许访问控制面板和数据面板 IAM 操作。

使用 Amazon Neptune 控制台所需的权限

对于要使用 Amazon Neptune 控制台的用户，用户必须拥有一组最低权限。这些权限允许用户描述其 AWS 账户的 Neptune 资源并提供其它相关信息（包括 Amazon EC2 安全和网络信息）。

如果创建比必需的最低权限更为严格的 IAM policy，对于附加了该 IAM policy 的用户，控制台将无法按预期正常运行。要确保这些用户仍可使用 Neptune 控制台，同时向用户附加 `NeptuneReadOnlyAccess` 托管式策略，请参阅AWS 亚马逊 Neptune 的托管（预定义）策略。

对于仅调用 AWS CLI 或 Amazon Neptune API 的用户，您无需为其设置最低控制台权限。

将 IAM policy 附加到 IAM 用户

要应用托管式策略或自定义策略，您需要将其附加到 IAM 用户。有关此主题的教程，请参阅《IAM 用户指南》中的创建和附加您的第一个客户管理型策略。

在演练此教程时，您可以使用此部分中显示的策略示例之一作为起点并根据您的需求进行定制。在教程结束时，您将有一个 IAM 用户，该用户具有可使用 `neptune-db:*` 操作的附加策略。

> **Important**
>
> - 对 IAM policy 的更改将需要长达 10 分钟才能应用于指定的 Neptune 资源。
> - 应用于 Neptune 数据库集群的 IAM policy 将应用于集群中的所有实例。

使用不同类型的 IAM policy 控制对 Neptune 的访问权限

要提供对 Neptune 管理操作或 Neptune 数据库集群中数据的访问权限，您需要向 IAM 用户或角色附加策略。有关如何将 IAM policy 附加到用户的信息，请参阅将 IAM policy 附加到 IAM 用户。有关将策略附加到角色的信息，请参阅《IAM 用户指南》中的添加和删除 IAM policy。

要获得对 Neptune 的常规访问权限，您可以使用 Neptune 的托管式策略之一。要获得更受限的访问权限，您可以使用 Neptune 支持的管理操作和资源创建自己的自定义策略。

在自定义 IAM policy 中，您可以使用两种不同的策略语句来控制 Neptune 数据库集群的不同访问模式：

- 管理策略语句 – 管理策略语句提供对用于创建、配置和管理数据库集群及其实例的 Neptune 管理 API 的访问权限。

  由于 Neptune 与 Amazon RDS 共享功能，因此 Neptune 策略中的管理操作、资源和条件键在设计上使用 `rds:` 前缀。
- 数据访问策略语句 – 数据访问策略语句使用数据访问操作、资源和条件键来控制对数据库集群包含的数据的访问。

  Neptune 数据访问操作、资源和条件键使用 `neptune-db:` 前缀。

在 Amazon Neptune 中使用 IAM 条件上下文键

您可以在 IAM policy 语句中指定用于控制 Neptune 访问权限的条件。仅当条件为 true 时，策略语句才有效。

例如，您可能希望策略语句仅在特定日期之后生效，或者仅在请求中存在特定值时才允许访问。

要表达条件，请在策略语句的 `Condition` 元素中使用预定义的条件键，以及 IAM 条件策略运算符（如等于或小于）。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

条件键的数据类型确定您可以使用哪些条件运算符以将请求中的值与策略语句中的值进行比较。如果您使用的条件运算符与该数据类型不兼容，则匹配始终失败，并且策略语句从不会适用。

Neptune 对于管理策略语句和数据访问策略语句支持不同的条件键集合：

- 管理策略语句的条件键
- 数据访问策略语句的条件键

Amazon Neptune 中对 IAM policy 和访问控制特征的支持

下表显示了 Neptune 对于管理策略语句和数据访问策略语句支持的 IAM 特征：

您可以与 Neptune 结合使用的 IAM 特征

IAM 功能	管理	数据访问
基于身份的策略	是	是
基于资源的策略	否	否
策略操作	是	是
策略资源	是	是
全局条件键	是	（子集）
基于标签的条件键	是	不支持
访问控制列表 (ACL)	否	否
服务控制策略 (SCP)	是	是
服务相关角色	是	不支持

IAM 策略限制

对 IAM policy 的更改将需要长达 10 分钟才能应用于指定的 Neptune 资源。

应用于 Neptune 数据库集群的 IAM policy 将应用于集群中的所有实例。

Neptune 目前不支持跨账户访问控制。

AWS 亚马逊 Neptune 的托管（预定义）策略

AWS 通过提供由创建和管理的独立 IAM 策略来解决许多常见用例 AWS。托管式策略可授予常用案例的必要权限，因此，您可以免去调查都需要哪些权限的工作。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略。

以下 AWS 托管策略适用于使用 Amazon Neptune 管理 API，您可以将其附加到账户中的用户：

- NeptuneReadOnlyAccess— 授予根账户中所有 Neptune 资源的只读访问权限，用于管理和数据访问目的。 AWS
- NeptuneFull访问权限-授予根账户中所有 Neptune 资源的完全访问权限，用于管理和数据访问目的。 AWS 如果您需要从 AWS CLI 或 SDK 获得完整的 Neptune 访问权限，但不要求 AWS Management Console 访问，则建议使用此方法。
- NeptuneConsoleFullAccess— 在 root AWS 帐户中授予对所有 Neptune 管理操作和资源的完全访问权限，但不授予对任何数据访问操作或资源的完全访问权限。它还包括用于简化从控制台访问 Neptune 的额外权限，包括有限的 IAM 和 Amazon EC2 (VPC) 权限。
- NeptuneGraphReadOnlyAccess — 提供对所有 Amazon Neptune Analytics 资源的只读访问权限以及依赖服务的只读权限
- AWSServiceRoleForNeptuneGraphPolicy— 让 Neptune Analytics 图表发布 CloudWatch 操作和使用指标以及日志。

Neptune IAM 角色和策略授予对 Amazon RDS 资源的部分访问权限，因为对于特定的管理特征，Neptune 采用了与 Amazon RDS 相同的操作技术。这包括管理 API 权限，这就是 Neptune 管理操作具有 `rds:` 前缀的原因。

Neptune AWS 托管策略的更新

下表跟踪从 Neptune 开始跟踪相应更改之时起对 Neptune 托管式策略的更新：

策略	描述	日期
AWS Amazon Neptune 的托管政策-现有政策的更新	`NeptuneReadOnlyAccess` 和`NeptuneFullAccess` 托管策略现在将`Sid`（声明 ID）作为标识符包含在策略声明中。	2024-01-22
NeptuneGraphReadOnly访问权限（已发布）	已发布，可提供对 Neptune Analytics 图和资源的只读访问权限。	2023-11-29
AWSServiceRoleForNeptuneGraphPolicy（已发布）	发布的目的是允许 Neptune Analytics 图表访问 CloudWatch 以发布操作和使用指标以及日志。请参阅 Using service-linked roles (SLRs) in Neptune Analytics。	2023-11-29
NeptuneConsoleFullAccess（已添加权限）	新增的权限提供了与 Neptune Analytics 图交互所需的所有访问权限。	2023-11/29
NeptuneFull访问权限（已添加权限）	添加了数据访问权限和对于新的全球数据库 API 的权限。	2022-07-28
NeptuneConsoleFullAccess（已添加权限）	添加了对于新的全球数据库 API 的权限。	2022-07-21
Neptune 开始了跟踪更改	Neptune 开始跟踪其 AWS 托管策略的变更。	2022-07-21

`NeptuneReadOnlyAccess`AWS 托管策略

以下NeptuneReadOnlyAccess托管策略授予对所有 Neptune 操作和资源的只读访问权限，用于管理和数据访问目的。

> **Note**
>
> 本策略已于 2022 年 7 月 21 日更新，以包括只读数据访问权限和只读管理权限，并包括全球数据库操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowReadOnlyPermissionsForRDS",
            "Effect": "Allow",
            "Action": [
                "rds:DescribeAccountAttributes",
                "rds:DescribeCertificates",
                "rds:DescribeDBClusterParameterGroups",
                "rds:DescribeDBClusterParameters",
                "rds:DescribeDBClusterSnapshotAttributes",
                "rds:DescribeDBClusterSnapshots",
                "rds:DescribeDBClusters",
                "rds:DescribeDBEngineVersions",
                "rds:DescribeDBInstances",
                "rds:DescribeDBLogFiles",
                "rds:DescribeDBParameterGroups",
                "rds:DescribeDBParameters",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DownloadDBLogFilePortion",
                "rds:ListTagsForResource"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForCloudwatch",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
```

```
                "cloudwatch:ListMetrics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForEC2",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForKMS",
            "Effect": "Allow",
            "Action": [
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "kms:ListAliases",
                "kms:ListKeyPolicies"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowReadOnlyPermissionsForLogs",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams",
                "logs:GetLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*",
                "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
            ]
        },
        {
            "Sid": "AllowReadOnlyPermissionsForNeptuneDB",
            "Effect": "Allow",
```

```
            "Action": [
                "neptune-db:Read*",
                "neptune-db:Get*",
                "neptune-db:List*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

NeptuneFullAccessAWS 托管策略

以下NeptuneFull访问管理策略授予出于管理和数据访问目的对所有 Neptune 操作和资源的完全访问权限。如果您需要从 AWS CLI 或 SDK 获得完全访问权限，但不需要从 SDK 获得完全访问权限，则建议您这样做 AWS Management Console。

> **Note**
>
> 本策略已于 2022 年 7 月 21 日更新，以包括完全数据访问权限和完全管理权限，并包括全球数据库操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowNeptuneCreate",
            "Effect": "Allow",
            "Action": [
                "rds:CreateDBCluster",
                "rds:CreateDBInstance"
            ],
            "Resource": [
                "arn:aws:rds:*:*:*"
            ],
            "Condition": {
                "StringEquals": {
                    "rds:DatabaseEngine": [
                        "graphdb",
                        "neptune"
```

```
                ]
            }
        }
    },
    {
        "Sid": "AllowManagementPermissionsForRDS",
        "Effect": "Allow",
        "Action": [
            "rds:AddRoleToDBCluster",
            "rds:AddSourceIdentifierToSubscription",
            "rds:AddTagsToResource",
            "rds:ApplyPendingMaintenanceAction",
            "rds:CopyDBClusterParameterGroup",
            "rds:CopyDBClusterSnapshot",
            "rds:CopyDBParameterGroup",
            "rds:CreateDBClusterEndpoint",
            "rds:CreateDBClusterParameterGroup",
            "rds:CreateDBClusterSnapshot",
            "rds:CreateDBParameterGroup",
            "rds:CreateDBSubnetGroup",
            "rds:CreateEventSubscription",
            "rds:CreateGlobalCluster",
            "rds:DeleteDBCluster",
            "rds:DeleteDBClusterEndpoint",
            "rds:DeleteDBClusterParameterGroup",
            "rds:DeleteDBClusterSnapshot",
            "rds:DeleteDBInstance",
            "rds:DeleteDBParameterGroup",
            "rds:DeleteDBSubnetGroup",
            "rds:DeleteEventSubscription",
            "rds:DeleteGlobalCluster",
            "rds:DescribeDBClusterEndpoints",
            "rds:DescribeAccountAttributes",
            "rds:DescribeCertificates",
            "rds:DescribeDBClusterParameterGroups",
            "rds:DescribeDBClusterParameters",
            "rds:DescribeDBClusterSnapshotAttributes",
            "rds:DescribeDBClusterSnapshots",
            "rds:DescribeDBClusters",
            "rds:DescribeDBEngineVersions",
            "rds:DescribeDBInstances",
            "rds:DescribeDBLogFiles",
            "rds:DescribeDBParameterGroups",
            "rds:DescribeDBParameters",
```

```
                "rds:DescribeDBSecurityGroups",
                "rds:DescribeDBSubnetGroups",
                "rds:DescribeEngineDefaultClusterParameters",
                "rds:DescribeEngineDefaultParameters",
                "rds:DescribeEventCategories",
                "rds:DescribeEventSubscriptions",
                "rds:DescribeEvents",
                "rds:DescribeGlobalClusters",
                "rds:DescribeOptionGroups",
                "rds:DescribeOrderableDBInstanceOptions",
                "rds:DescribePendingMaintenanceActions",
                "rds:DescribeValidDBInstanceModifications",
                "rds:DownloadDBLogFilePortion",
                "rds:FailoverDBCluster",
                "rds:FailoverGlobalCluster",
                "rds:ListTagsForResource",
                "rds:ModifyDBCluster",
                "rds:ModifyDBClusterEndpoint",
                "rds:ModifyDBClusterParameterGroup",
                "rds:ModifyDBClusterSnapshotAttribute",
                "rds:ModifyDBInstance",
                "rds:ModifyDBParameterGroup",
                "rds:ModifyDBSubnetGroup",
                "rds:ModifyEventSubscription",
                "rds:ModifyGlobalCluster",
                "rds:PromoteReadReplicaDBCluster",
                "rds:RebootDBInstance",
                "rds:RemoveFromGlobalCluster",
                "rds:RemoveRoleFromDBCluster",
                "rds:RemoveSourceIdentifierFromSubscription",
                "rds:RemoveTagsFromResource",
                "rds:ResetDBClusterParameterGroup",
                "rds:ResetDBParameterGroup",
                "rds:RestoreDBClusterFromSnapshot",
                "rds:RestoreDBClusterToPointInTime",
                "rds:StartDBCluster",
                "rds:StopDBCluster"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowOtherDepedentPermissions",
```

```
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:ListMetrics",
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeVpcs",
                "kms:ListAliases",
                "kms:ListKeyPolicies",
                "kms:ListKeys",
                "kms:ListRetirableGrants",
                "logs:DescribeLogStreams",
                "logs:GetLogEvents",
                "sns:ListSubscriptions",
                "sns:ListTopics",
                "sns:Publish"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "AllowPassRoleForNeptune",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:passedToService": "rds.amazonaws.com"
                }
            }
        },
        {
            "Sid": "AllowCreateSLRForNeptune",
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "rds.amazonaws.com"
```

```
                }
            }
        },
        {
            "Sid": "AllowDataAccessForNeptune",
            "Effect": "Allow",
            "Action": [
                "neptune-db:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

NeptuneConsoleFullAccessAWS 托管策略

以下NeptuneConsoleFullAccess托管策略授予出于管理目的对所有 Neptune 操作和资源的完全访问权限，但不允许出于数据访问目的。它还包括用于简化从控制台访问 Neptune 的额外权限，包括有限的 IAM 和 Amazon EC2 (VPC) 权限。

> **Note**
>
> 本策略已于 2023 年 11 月 29 日更新，增加了与 Neptune Analytics 图交互所需的权限。
> 它已于 2022 年 7 月 21 日更新，以包括对全球数据库操作的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowNeptuneCreate",
      "Effect": "Allow",
      "Action": [
        "rds:CreateDBCluster",
        "rds:CreateDBInstance"
      ],
      "Resource": [
        "arn:aws:rds:*:*:*"
      ],
      "Condition": {
```

```
        "StringEquals": {
          "rds:DatabaseEngine": [
            "graphdb",
            "neptune"
          ]
        }
      }
    },
    {
      "Sid": "AllowManagementPermissionsForRDS",
      "Action": [
        "rds:AddRoleToDBCluster",
        "rds:AddSourceIdentifierToSubscription",
        "rds:AddTagsToResource",
        "rds:ApplyPendingMaintenanceAction",
        "rds:CopyDBClusterParameterGroup",
        "rds:CopyDBClusterSnapshot",
        "rds:CopyDBParameterGroup",
        "rds:CreateDBClusterParameterGroup",
        "rds:CreateDBClusterSnapshot",
        "rds:CreateDBParameterGroup",
        "rds:CreateDBSubnetGroup",
        "rds:CreateEventSubscription",
        "rds:DeleteDBCluster",
        "rds:DeleteDBClusterParameterGroup",
        "rds:DeleteDBClusterSnapshot",
        "rds:DeleteDBInstance",
        "rds:DeleteDBParameterGroup",
        "rds:DeleteDBSubnetGroup",
        "rds:DeleteEventSubscription",
        "rds:DescribeAccountAttributes",
        "rds:DescribeCertificates",
        "rds:DescribeDBClusterParameterGroups",
        "rds:DescribeDBClusterParameters",
        "rds:DescribeDBClusterSnapshotAttributes",
        "rds:DescribeDBClusterSnapshots",
        "rds:DescribeDBClusters",
        "rds:DescribeDBEngineVersions",
        "rds:DescribeDBInstances",
        "rds:DescribeDBLogFiles",
        "rds:DescribeDBParameterGroups",
        "rds:DescribeDBParameters",
        "rds:DescribeDBSecurityGroups",
        "rds:DescribeDBSubnetGroups",
```

```
        "rds:DescribeEngineDefaultClusterParameters",
        "rds:DescribeEngineDefaultParameters",
        "rds:DescribeEventCategories",
        "rds:DescribeEventSubscriptions",
        "rds:DescribeEvents",
        "rds:DescribeOptionGroups",
        "rds:DescribeOrderableDBInstanceOptions",
        "rds:DescribePendingMaintenanceActions",
        "rds:DescribeValidDBInstanceModifications",
        "rds:DownloadDBLogFilePortion",
        "rds:FailoverDBCluster",
        "rds:ListTagsForResource",
        "rds:ModifyDBCluster",
        "rds:ModifyDBClusterParameterGroup",
        "rds:ModifyDBClusterSnapshotAttribute",
        "rds:ModifyDBInstance",
        "rds:ModifyDBParameterGroup",
        "rds:ModifyDBSubnetGroup",
        "rds:ModifyEventSubscription",
        "rds:PromoteReadReplicaDBCluster",
        "rds:RebootDBInstance",
        "rds:RemoveRoleFromDBCluster",
        "rds:RemoveSourceIdentifierFromSubscription",
        "rds:RemoveTagsFromResource",
        "rds:ResetDBClusterParameterGroup",
        "rds:ResetDBParameterGroup",
        "rds:RestoreDBClusterFromSnapshot",
        "rds:RestoreDBClusterToPointInTime"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowOtherDepedentPermissions",
      "Action": [
        "cloudwatch:GetMetricStatistics",
        "cloudwatch:ListMetrics",
        "ec2:AllocateAddress",
        "ec2:AssignIpv6Addresses",
        "ec2:AssignPrivateIpAddresses",
        "ec2:AssociateAddress",
        "ec2:AssociateRouteTable",
```

```
        "ec2:AssociateSubnetCidrBlock",
        "ec2:AssociateVpcCidrBlock",
        "ec2:AttachInternetGateway",
        "ec2:AttachNetworkInterface",
        "ec2:CreateCustomerGateway",
        "ec2:CreateDefaultSubnet",
        "ec2:CreateDefaultVpc",
        "ec2:CreateInternetGateway",
        "ec2:CreateNatGateway",
        "ec2:CreateNetworkInterface",
        "ec2:CreateRoute",
        "ec2:CreateRouteTable",
        "ec2:CreateSecurityGroup",
        "ec2:CreateSubnet",
        "ec2:CreateVpc",
        "ec2:CreateVpcEndpoint",
        "ec2:CreateVpcEndpoint",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAccountAttributes",
        "ec2:DescribeAddresses",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeCustomerGateways",
        "ec2:DescribeInstances",
        "ec2:DescribeNatGateways",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribePrefixLists",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroupReferences",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs",
        "ec2:DescribeVpcs",
        "ec2:ModifyNetworkInterfaceAttribute",
        "ec2:ModifySubnetAttribute",
        "ec2:ModifyVpcAttribute",
        "ec2:ModifyVpcEndpoint",
        "iam:ListRoles",
        "kms:ListAliases",
```

```
        "kms:ListKeyPolicies",
        "kms:ListKeys",
        "kms:ListRetirableGrants",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "sns:ListSubscriptions",
        "sns:ListTopics",
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "AllowPassRoleForNeptune",
      "Action": "iam:PassRole",
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:passedToService": "rds.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowCreateSLRForNeptune",
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "rds.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowManagementPermissionsForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": [
        "neptune-graph:CreateGraph",
        "neptune-graph:DeleteGraph",
        "neptune-graph:GetGraph",
```

```
        "neptune-graph:ListGraphs",
        "neptune-graph:UpdateGraph",
        "neptune-graph:ResetGraph",
        "neptune-graph:CreateGraphSnapshot",
        "neptune-graph:DeleteGraphSnapshot",
        "neptune-graph:GetGraphSnapshot",
        "neptune-graph:ListGraphSnapshots",
        "neptune-graph:RestoreGraphFromSnapshot",
        "neptune-graph:CreatePrivateGraphEndpoint",
        "neptune-graph:GetPrivateGraphEndpoint",
        "neptune-graph:ListPrivateGraphEndpoints",
        "neptune-graph:DeletePrivateGraphEndpoint",
        "neptune-graph:CreateGraphUsingImportTask",
        "neptune-graph:GetImportTask",
        "neptune-graph:ListImportTasks",
        "neptune-graph:CancelImportTask"
      ],
      "Resource": [
        "arn:aws:neptune-graph:*:*:*"
      ]
    },
    {
      "Sid": "AllowPassRoleForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:passedToService": "neptune-graph.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AllowCreateSLRForNeptuneAnalytics",
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/neptune-graph.amazonaws.com/
AWSServiceRoleForNeptuneGraph",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "neptune-graph.amazonaws.com"
        }
      }
    }
```

```
  ]
}
```

NeptuneGraphReadOnlyAccessAWS 托管策略

以下NeptuneGraphReadOnly访问托管策略提供对所有 Amazon Neptune Analytics 资源的只读访问权限以及依赖服务的只读权限。

此策略包括以下权限：

- 对于 Amazon EC2 - 检索有关 VPC、子网、安全组和可用区的信息。
- 用于 AWS KMS-检索有关 KMS 密钥和别名的信息。
- 对于 CloudWatch-检索有关 CloudWatch 指标的信息。
- 对于 CloudWatch 日志-检索有关 CloudWatch 日志流和事件的信息。

> **Note**
>
> 该策略已于 2023 年 11 月 29 日发布。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReadOnlyPermissionsForNeptuneGraph",
      "Effect": "Allow",
      "Action": [
        "neptune-graph:Get*",
        "neptune-graph:List*",
        "neptune-graph:Read*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForEC2",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcAttribute",
        "ec2:DescribeSecurityGroups",
```

```
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeAvailabilityZones"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForKMS",
      "Effect": "Allow",
      "Action": [
        "kms:ListKeys",
        "kms:ListAliases"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForCloudwatch",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:GetMetricData",
        "cloudwatch:ListMetrics",
        "cloudwatch:GetMetricStatistics"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowReadOnlyPermissionsForLogs",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
      ]
    }
  ]
}
```

AWSServiceRoleForNeptuneGraphPolicyAWS 托管策略

下面的AWSServiceRoleForNeptuneGraphPolicy托管策略允许图表发布操作和使用情况指标和日志。CloudWatch 请参阅 nan-service-linked-roles。

> **Note**
>
> 该策略已于 2023 年 11 月 29 日发布。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GraphMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/Neptune",
            "AWS/Usage"
          ]
        }
      }
    },
    {
      "Sid": "GraphLogGroup",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    },
    {
      "Sid": "GraphLogEvents",
      "Effect": "Allow",
```

```
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/neptune/*:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "${aws:PrincipalAccount}"
        }
      }
    }
  ]
}
```

Amazon Neptune 支持的 IAM 条件上下文键

您可以在 IAM policy 中指定条件，以控制对 Neptune 管理操作和资源的访问权限。仅当条件为 true 时，策略语句才有效。

例如，您可能希望策略语句仅在特定日期之后生效，或者仅在 API 请求中存在特定值时才允许访问。

要表达条件，请在策略语句的 `Condition` 元素中使用预定义的条件键，以及 IAM 条件策略运算符（如等于或小于）。

如果您在一个语句中指定多个 `Condition` 元素，或在单个 `Condition` 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

条件键的数据类型确定您可以使用哪些条件运算符以将请求中的值与策略语句中的值进行比较。如果您使用的条件运算符与该数据类型不兼容，则匹配始终失败，并且策略语句从不会适用。

Neptune 管理策略语句的 IAM 条件键

- 全局条件键-您可以在 Neptune 管理策略声明中使用大多数 AWS 全局条件键。

- 特定于服务的条件密钥-这些密钥是为特定 AWS 服务定义的密钥。Neptune IAM 管理策略语句中提供的条件键中列出了 Neptune 对于管理策略语句支持的条件键。

Neptune 数据访问策略语句的 IAM 条件键

- 全局条件键 – AWS Neptune 在数据访问策略声明中支持的全局条件上下文密钥中列出了 Neptune 在数据访问策略语句中支持的这些键的子集。
- 条件键中列出了 Neptune 为数据访问策略语句定义的特定于服务的条件键。

Amazon Neptune 的自定义 IAM 管理策略语句

管理策略语句允许您控制 IAM 用户可以执行哪些操作来管理 Neptune 数据库。

Neptune 管理策略语句授予对 Neptune 支持的一项或多项管理操作和管理资源的访问权限。您还可以使用条件键以使管理权限更具体。

> **Note**
>
> 由于 Neptune 与 Amazon RDS 共享功能，因此管理策略语句中的管理操作、资源和特定于服务的条件键在设计上使用 `rds:` 前缀。

主题

- Neptune IAM 管理策略语句中提供的操作
- IAM Neptune 管理策略语句中提供的资源类型
- Neptune IAM 管理策略语句中提供的条件键
- Neptune 的 IAM 管理策略语句示例

Neptune IAM 管理策略语句中提供的操作

您可以使用下面在 IAM policy 语句的 `Action` 元素中列出的管理操作来控制对 Neptune 管理 API 的访问权限。您在策略中使用一项操作时，通常使用相同的名称允许或拒绝对 API 操作或 CLI 命令的访问。但在某些情况下，单一动作可控制对多项操作的访问。还有某些操作需要多种不同的动作。

下表中的 `Resource type` 字段指示每项操作是否支持资源级权限。如果此字段中没有任何值，您必须在策略语句的 `Resource` 元素中指定所有资源 ("*")。如果该列包含一种资源类型，则可以在含有该操作的语句中指定该类型的资源 ARN。本页列出了 Neptune 管理资源类型。

必需资源在下面的列表中以星号 (*) 指示。如果在使用该操作的语句中指定资源级权限 ARN，则它必须属于该类型。某些操作支持多种资源类型。如果资源类型是可选的（换句话说，没有用星号标记），则不必将其包括在内。

有关此处列出的字段的更多信息，请参阅 IAM 用户指南中的操作表。

rds：AddRoletodbClust

AddRoleToDBCluster 将 IAM 角色与 Neptune 数据库集群关联。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：cluster（必需）。

rds：AddSourceIdentifierTo订阅

AddSourceIdentifierToSubscription 将源标识符添加到现有 Neptune 事件通知订阅。

访问级别：Write。

资源类型：es（必需）。

rds: AddTags ToResource

AddTagsToResource 将 IAM 角色与 Neptune 数据库集群关联。

访问级别：Write。

资源类型：

- db
- es
- pg
- cluster-snapshot
- subgrp

条件键：

- aws:RequestTag/*tag*-key

- aws：TagKeys

rds: ApplyPending MaintenanceAction

ApplyPendingMaintenanceAction 将待处理的维护操作应用于资源。

访问级别：Write。

资源类型：db（必需）。

RDS: CopyDB 组 ClusterParameter

CopyDBClusterParameterGroup 复制指定的数据库集群参数组。

访问级别：Write。

资源类型：cluster-pg（必需）。

rds: copyDB ClusterSnapshot

CopyDBClusterSnapshot 复制数据库集群的快照。

访问级别：Write。

资源类型：cluster-snapshot（必需）。

rds: copyDB ParameterGroup

CopyDBParameterGroup 复制指定的数据库参数组。

访问级别：Write。

资源类型：pg（必需）。

rds:CreateDBCluster

CreateDBCluster 创建新的 Neptune 数据库集群。

访问级别：Tagging。

相关操作：iam:PassRole。

资源类型：

- cluster（必需）。
- cluster-pg（必需）。
- subgrp（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys
- neptune-rds_ DatabaseEngine

RDS: CreateDB 组 ClusterParameter

`CreateDBClusterParameterGroup` 创建新的数据库集群参数组。

访问级别：`Tagging`。

资源类型：cluster-pg（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

RDS: createDB ClusterSnapshot

`CreateDBClusterSnapshot` 创建数据库集群的快照。

访问级别：`Tagging`。

资源类型：

- cluster（必需）。
- cluster-snapshot（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

rds:CreateDBInstance

CreateDBInstance 创建新的数据库实例。

访问级别：Tagging。

相关操作：iam:PassRole。

资源类型：

- db（必需）。
- pg（必需）。
- subgrp（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

RDS: createDB ParameterGroup

CreateDBParameterGroup 创建一个新的数据库参数组。

访问级别：Tagging。

资源类型：pg（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

RDS: createDB SubnetGroup

CreateDBSubnetGroup 创建新的数据库子网组。

访问级别：Tagging。

资源类型：subgrp（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

rds：CreateEvent订阅

CreateEventSubscription 创建 Neptune 事件通知订阅。

访问级别：Tagging。

资源类型：es（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

rds:DeleteDBCluster

DeleteDBCluster 删除现有的 Neptune 数据库集群。

访问级别：Write。

资源类型：

- cluster（必需）。
- cluster-snapshot（必需）。

RDS: deletedB 群组 ClusterParameter

DeleteDBClusterParameterGroup 删除指定的数据库集群参数组。

访问级别：Write。

资源类型：cluster-pg（必需）。

RDS: deletedB ClusterSnapshot

DeleteDBClusterSnapshot 删除数据库集群快照。

访问级别：Write。

资源类型：cluster-snapshot（必需）。

rds:DeleteDBInstance

DeleteDBInstance 删除指定的数据库实例。

访问级别：Write。

资源类型：db（必需）。

RDS: deletedB ParameterGroup

DeleteDBParameterGroup删除指定的数据库ParameterGroup。

访问级别：Write。

资源类型：pg（必需）。

RDS: deletedB SubnetGroup

DeleteDBSubnetGroup 删除数据库子网组。

访问级别：Write。

资源类型：subgrp（必需）。

rds：DeleteEvent订阅

DeleteEventSubscription 删除事件通知订阅。

访问级别：Write。

资源类型：es（必需）。

RDSClusterParameter: describedB 群组

DescribeDBClusterParameterGroups返回数据库ClusterParameterGroup 描述列表。

访问级别：List。

资源类型：cluster-pg（必需）。

RDS: describedB ClusterParameters

DescribeDBClusterParameters 返回特定数据库集群参数组的详细参数列表。

访问级别：List。

资源类型：cluster-pg（必需）。

rds: describedB 属性 ClusterSnapshot

DescribeDBClusterSnapshotAttributes 返回手动数据库集群快照的数据库集群快照属性名称和值的列表。

访问级别：List。

资源类型：cluster-snapshot（必需）。

RDS: describedB ClusterSnapshots

DescribeDBClusterSnapshots 返回有关数据库集群快照的信息。

访问级别：Read。

rds:DescribeDBClusters

DescribeDBClusters 返回有关预调配的 Neptune 数据库集群的信息。

访问级别：List。

资源类型：cluster（必需）。

RDS: describedB EngineVersions

DescribeDBEngineVersions 返回可用数据库引擎的列表。

访问级别：List。

资源类型：pg（必需）。

rds:DescribeDBInstances

DescribeDBInstances 返回有关数据库实例的信息。

访问级别：List。

资源类型：es（必需）。

RDS: describedB ParameterGroups

DescribeDBParameterGroups返回数据库ParameterGroup 描述列表。

访问级别：List。

资源类型：pg（必需）。

rds:DescribeDBParameters

DescribeDBParameters 返回特定数据库参数组的详细参数列表。

访问级别：List。

资源类型：pg（必需）。

RDS: describedB SubnetGroups

DescribeDBSubnetGroups返回数据库SubnetGroup 描述列表。

访问级别：List。

资源类型：subgrp（必需）。

rds: DescribeEvent 类别

DescribeEventCategories 返回所有事件源类型的类别列表；或如果指定，则显示指定源类型的类别列表。

访问级别：List。

rds: DescribeEvent 订阅

DescribeEventSubscriptions 列出客户账户的所有订阅描述。

访问级别：List。

资源类型：es（必需）。

rds: DescribeEvents

DescribeEvents 返回过去 14 天与数据库实例、数据库安全组和数据库参数组相关的事件。

访问级别：List。

资源类型：es（必需）。

r DescribeOrderable ds: 数据库 InstanceOptions

DescribeOrderableDBInstanceOptions 返回指定引擎的可订购数据库实例选项的列表。

访问级别：List。

rds: DescribePending MaintenanceActions

DescribePendingMaintenanceActions 返回至少具有一个待处理的维护操作的资源（例如，数据库实例）的列表。

访问级别：List。

资源类型：db（必需）。

r DescribeValid ds: 数据库 InstanceModifications

DescribeValidDBInstanceModifications 列出可以对数据库实例进行的修改。

访问级别：List。

资源类型：db（必需）。

rds:FailoverDBCluster

FailoverDBCluster 强制数据库集群失效转移。

访问级别：Write。

资源类型：cluster（必需）。

rds: ListTags ForResource

ListTagsForResource 列出 Neptune 资源上的所有标签。

访问级别：Read。

资源类型：

- cluster-snapshot
- db
- es
- pg
- subgrp

rds:ModifyDBCluster

ModifyDBCluster

修改 Neptune 数据库集群的设置。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：

- cluster（必需）。
- cluster-pg（必需）。

RDS: modifyDB 组 ClusterParameter

ModifyDBClusterParameterGroup 修改数据库集群参数组的参数。

访问级别：Write。

资源类型：cluster-pg（必需）。

rds: modifyDB 属性 ClusterSnapshot

ModifyDBClusterSnapshotAttribute 向手动数据库集群快照添加属性和值，或者从中删除属性和值。

访问级别：Write。

资源类型：cluster-snapshot（必需）。

rds:ModifyDBInstance

ModifyDBInstance 修改数据库实例的设置。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：

- db（必需）。

- pg（必需）。

rds: modifyDB ParameterGroup

ModifyDBParameterGroup 修改数据库参数组的参数。

访问级别：Write。

资源类型：pg（必需）。

rds: modifyDB SubnetGroup

ModifyDBSubnetGroup 修改现有的数据库子网组。

访问级别：Write。

资源类型：subgrp（必需）。

rds：ModifyEvent订阅

ModifyEventSubscription 修改现有 Neptune 事件通知订阅。

访问级别：Write。

资源类型：es（必需）。

rds:RebootDBInstance

RebootDBInstance 重启实例的数据库引擎服务。

访问级别：Write。

资源类型：db（必需）。

rds：RemoveRole来自dbCluster

RemoveRoleFromDBCluster取消 AWS 身份和访问管理 (IAM) Access Management 角色与亚马逊 Neptune 数据库集群的关联。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：cluster（必需）。

rds：RemoveSourceIdentifierFrom订阅

RemoveSourceIdentifierFromSubscription 从现有 Neptune 事件通知订阅中移除源标识符。

访问级别：Write。

资源类型：es（必需）。

rds: RemoveTags FromResource

RemoveTagsFromResource从 Neptune 资源中移除元数据标签。

访问级别：Tagging。

资源类型：

- cluster-snapshot
- db
- es
- pg
- subgrp

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

rds: resetDB 组 ClusterParameter

ResetDBClusterParameterGroup 将数据库集群参数组的参数修改为默认值。

访问级别：Write。

资源类型：cluster-pg（必需）。

rds: resetDB ParameterGroup

ResetDBParameterGroup 将数据库参数组的参数修改为引擎/系统默认值。

访问级别：Write。

资源类型：pg（必需）。

RDSClusterFrom: restoredB 快照

RestoreDBClusterFromSnapshot 从数据库集群快照中创建新的数据库集群。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：

- cluster（必需）。
- cluster-snapshot（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

RDS ClusterToPointIn: restoredB 时间

RestoreDBClusterToPointInTime 将数据库集群还原到任意时间点。

访问级别：Write。

相关操作：iam:PassRole。

资源类型：

- cluster（必需）。
- subgrp（必需）。

条件键：

- aws:RequestTag/*tag*-key
- aws：TagKeys

rds:StartDBCluster

StartDBCluster 启动指定的数据库集群。

访问级别：Write。

资源类型：cluster（必需）。

rds:StopDBCluster

StopDBCluster 停止指定的数据库集群。

访问级别：Write。

资源类型：cluster（必需）。

IAM Neptune 管理策略语句中提供的资源类型

Neptune 支持下表中的资源类型，以用于 IAM 管理策略语句的 Resource 元素。有关 Resource 元素的更多信息，请参阅 IAM JSON 策略元素：Resource。

Neptune 管理操作列表确定了可以使用每个操作指定的资源类型。资源类型还会决定您可以在策略中包含哪些条件键，如下表的最后一列所指定。

下表中的 ARN 列指定在引用该类型的资源时必须使用的 Amazon 资源名称 (ARN) 格式。前缀为 $ 的部分必须替换为您的方案的实际值。例如，如果在 ARN 中看到 $user-name，您必须将该字符串替换为实际 IAM 用户的名称或包含 IAM 用户名的策略变量。有关 ARN 的更多信息，请参阅 IAM ARN 和在 Amazon Neptune 中使用管理 ARN。

Condition Keys 列指定条件上下文键，只有在 IAM policy 语句中同时包含该资源和兼容的支持操作时，才能在该语句中包含这些键。

资源类型	ARN	条件键
cluster（数据库集群）	arn:*partition* :rds:*region*:*account-id* :cluster: *instance-name*	aws:ResourceTag/*tag*-key rds:cluster-tag/*tag-key*
cluster-pg（数据库集群参数组）	arn:*partition* :rds:*region*:*account-id* :cluster-pg: *neptune-DBClusterParameterGroupName*	aws:ResourceTag/*tag*-key

资源类型	ARN	条件键
cluster-snapshot（数据库集群快照）	arn:*partition* :rds:*region*:*account-id* :cluster-snapshot: *neptune-DBClusterSnapshotName*	aws:ResourceTag/*tag*-key rds:cluster-snapshot-tag/*tag-key*
db（数据库实例）	arn:*partition* :rds:*region*:*account-id* :db:*neptune-DbInstanceName*	aws:ResourceTag/*tag*-key rds: DatabaseClass rds: DatabaseEngine rds:db-tag/*tag-key*
es（事件订阅）	arn:*partition* :rds:*region*:*account-id* :es:*neptune-CustSubscriptionId*	aws:ResourceTag/*tag*-key rds:es-tag/*tag-key*
pg（数据库参数组）	arn:*partition* :rds:*region*:*account-id* :pg:*neptune-ParameterGroupName*	aws:ResourceTag/*tag*-key rds:pg-tag/*tag-key*
subgrp（数据库子网组）	arn:*partition* :rds:*region*:*account-id* :subgrp:*neptune-DBSubnetGroupName* }	aws:ResourceTag/*tag*-key rds:subgrp-tag/*tag-key*

Neptune IAM 管理策略语句中提供的条件键

使用条件键，您可以在 IAM policy 语句中指定条件，这样该语句仅在条件为 true 时才生效。您可以在 Neptune 管理策略语句中使用的条件键分为以下几类：

- 全局条件键 — 这些条件键由定义 AWS ，供一般 AWS 服务使用。大多数可用于 Neptune 管理策略语句中。

- 管理资源属性条件键 - 下面列出的这些键基于管理资源的属性。
- 基于标签的访问条件键 - 下面列出的这些键基于附加到管理资源的 AWS 标签。

Neptune 管理资源属性条件键

条件键	描述	类型
`rds:DatabaseClass`	按数据库实例类的类型筛选访问。	String
`rds:DatabaseEngine`	按数据库引擎筛选访问。有关可能的值，请参阅 CreateDBInstance API 中的引擎参数	字符串
`rds:DatabaseName`	按数据库实例上的数据库的用户定义名称筛选访问	字符串
`rds:EndpointType`	按终端节点类型筛选访问。它是以下内容之一：READER、WRITER、CUSTOM	String
`rds:Vpc`	指定数据库实例是否在 Amazon Virtual Private Cloud (Amazon VPC) 中运行的值筛选访问。要指示数据库实例在 Amazon VPC 中运行，请指定 `true`。	布尔值

基于管理标签的条件键

Amazon Neptune 支持在 IAM policy 中使用自定义标签指定条件，以通过 管理 API 参考 控制对 Neptune 的访问。

例如，如果您向数据库实例添加一个名为 `environment` 的标签，其值为 `beta`、`staging`、和 `production` 等，则可以创建一个策略，以根据该标签的值限制对实例的访问。

> **Important**
>
> 如果您使用标签管理对 Neptune 资源的访问，请保护对于标签的访问。您可通过为 `AddTagsToResource` 和 `RemoveTagsFromResource` 操作创建策略来限制对标签的访问。例如，您可以使用以下策略拒绝用户为所有资源添加或删除标签。然后，您可以创建策略来允许特定用户添加或删除标签。

```
{ "Version": "2012-10-17",
  "Statement":[
    { "Sid": "DenyTagUpdates",
      "Effect": "Deny",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource":"*"
    }
  ]
}
```

以下基于标签的条件键仅适用于管理策略语句中的管理资源。

基于标签的管理条件键

条件键	描述	类型
aws:RequestTag/${TagKey}	根据在请求中是否具有标签键/值对来筛选访问。	String
aws:ResourceTag/${TagKey}	根据附加到资源的标签键/值对筛选访问。	String
aws:TagKeys	根据在请求中是否具有标签键来筛选访问。	String
rds:cluster-pg-tag/${TagKey}	按附加到数据库集群参数组的标签筛选访问。	String
rds:cluster-snapshot-tag/${TagKey}	按附加到数据库集群快照的标签筛选访问。	String

条件键	描述	类型
rds:cluster-tag/${TagKey}	按附加到数据库集群的标签筛选访问。	String
rds:db-tag/${TagKey}	按附加到数据库实例的标签筛选访问。	String
rds:es-tag/${TagKey}	按附加到事件订阅的标签筛选访问。	String
rds:pg-tag/${TagKey}	按附加到数据库参数组的标签筛选访问。	String
rds:req-tag/${TagKey}	按可用于对资源进行标记的一组标签键和值筛选访问。	String
rds:secgrp-tag/${TagKey}	按附加到数据库安全组的标签筛选访问。	String
rds:snapshot-tag/${TagKey}	按附加到数据库快照的标签筛选访问。	String
rds:subgrp-tag/${TagKey}	按附加到数据库子网组的标签筛选访问	String

Neptune 的 IAM 管理策略语句示例

一般管理策略示例

以下示例说明如何创建 Neptune 管理策略，以授予对数据库集群执行各种管理操作的权限。

防止 IAM 用户删除指定数据库实例的策略

以下是防止 IAM 用户删除指定的 Neptune 数据库实例的策略示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyDeleteOneInstance",
      "Effect": "Deny",
      "Action": "rds:DeleteDBInstance",
      "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-instance-name"
    }
  ]
}
```

授予创建新数据库实例的权限的策略

以下是允许 IAM 用户在指定的 Neptune 数据库集群中创建数据库实例的示例策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateInstance",
      "Effect": "Allow",
      "Action": "rds:CreateDBInstance",
      "Resource": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster"
    }
  ]
}
```

授予创建新数据库实例（使用特定数据库参数组）的权限的策略

以下是一个策略示例，它允许 IAM 用户仅使用指定的数据库参数组，在指定的 Neptune 数据库集群的指定数据库集群（此处为 us-west-2）中创建数据库实例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateInstanceWithPG",
      "Effect": "Allow",
```

```
      "Action": "rds:CreateDBInstance",
      "Resource": [
        "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
        "arn:aws:rds:us-west-2:123456789012:pg:my-instance-pg"
      ]
    }
  ]
}
```

授予描述任何资源的权限的策略

以下是允许 IAM 用户描述任何 Neptune 资源的策略示例。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribe",
      "Effect": "Allow",
      "Action": "rds:Describe*",
      "Resource": *
    }
  ]
}
```

基于标签的管理策略示例

以下示例说明如何创建 Neptune 管理策略，这些策略将进行标记，以筛选在数据库集群上执行各种管理操作的权限。

示例 1：使用可以取多个值的自定义标签，授予对资源执行操作的权限

以下策略允许在 env 标签设置为 dev 或 test 的任何数据库实例上使用 ModifyDBInstance、CreateDBInstance 或 DeleteDBInstance API：

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
        "rds:ModifyDBInstance",
        "rds:CreateDBInstance",
```

```
        "rds:DeleteDBInstance"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "rds:db-tag/env": [
              "dev",
              "test"
          ],
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

示例 2：限制可用于对资源进行标记的一组标签键和值

此策略使用 Condition 键来允许将键 env 且值为 test、qa 或 dev 的标签添加到资源中：

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "rds:AddTagsToResource",
        "rds:RemoveTagsFromResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "rds:req-tag/env": [
            "test",
            "qa",
            "dev"
          ],
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

示例 3：允许基于 **aws:ResourceTag** 对 Neptune 资源进行完全访问

以下策略与上面的第一个示例类似，但改为使用 aws:ResourceTag：

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowFullAccessToDev",
      "Effect": "Allow",
      "Action": [
        "rds:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "dev",
          "rds:DatabaseEngine": "neptune"
        }
      }
    }
  ]
}
```

Amazon Neptune 的自定义 IAM 数据访问策略语句

Neptune 数据访问策略语句使用数据访问操作、资源和条件键，所有这些都以 neptune-db: 前缀开头。

主题

- 在 Neptune 数据访问策略语句中使用查询操作
- Neptune IAM 数据访问策略语句中提供的操作
- 在 Neptune IAM 数据访问策略语句中指定资源
- Neptune IAM 数据访问策略语句中提供的条件键
- Neptune IAM 数据访问策略示例

在 Neptune 数据访问策略语句中使用查询操作

在数据访问策略语句中可以使用三个 Neptune 查询操作，即 `ReadDataViaQuery`、`WriteDataViaQuery` 和 `DeleteDataViaQuery`。特定的查询可能需要权限才能执行其中多个操作，而且可能并不总是很清楚必须允许这些操作的哪种组合才能运行查询。

在运行查询之前，Neptune 会确定运行每个查询步骤所需的权限，并将这些权限组合成查询所需的完整权限集。请注意，这组完整的权限包括查询可能执行的所有操作，这些操作不一定是查询在数据上运行时实际执行的一组操作。

这意味着，要允许给定的查询运行，必须针对该查询可能执行的每个操作提供权限，无论它是否实际执行这些操作。

以下是一些示例 Gremlin 查询，其中对此进行了更详细的解释：

- ```
  g.V().count()
  ```

  `g.V()` 和 `count()` 只需要读取权限，因此整个查询只需要 `ReadDataViaQuery` 访问权限。

- ```
  g.addV()
  ```

  在插入新顶点之前，`addV()` 需要检查具有给定 ID 的顶点是否存在。这意味着它同时需要 `ReadDataViaQuery` 和 `WriteDataViaQuery` 访问权限。

- ```
  g.V('1').as('a').out('created').addE('createdBy').to('a')
  ```

  `g.V('1').as('a')` 和 `out('created')` 只需要读取权限，但 `addE().from('a')` 同时需要读取和写入权限，因为在添加新边缘之前，`addE()` 需要读取 `from` 和 `to` 顶点并检查是否已经存在具有相同 ID 的边缘。因此，整个查询同时需要 `ReadDataViaQuery` 和 `WriteDataViaQuery` 访问权限。

- ```
  g.V().drop()
  ```

  `g.V()` 只需要读取访问权限。`drop()` 同时需要读取和删除访问权限，因为它需要在删除顶点或边缘之前读取顶点或边缘，因此查询总体同时需要 `ReadDataViaQuery` 和 `DeleteDataViaQuery` 访问权限。

- ```
  g.V('1').property(single, 'key1', 'value1')
  ```
  ```

`g.V('1')` 只需要读取访问权限，但 `property(single, 'key1', 'value1')` 需要读取、写入和删除访问权限。在这里，如果键和值尚不存在于顶点中，则 `property()` 步骤会插入它们，但如果它们已经存在，则该步骤会删除现有的属性值并在其位置插入一个新值。因此，整个查询需要 `ReadDataViaQuery`、`WriteDataViaQuery` 和 `DeleteDataViaQuery` 访问权限。

任何包含 `property()` 步骤的查询都需要 `ReadDataViaQuery`、`WriteDataViaQuery` 和 `DeleteDataViaQuery` 权限。

下面是一些 openCypher 示例：

- ```
  MATCH (n)
  RETURN n
  ```

  此查询读取数据库中的所有节点并返回它们，这只需要 `ReadDataViaQuery` 访问权限即可。

- ```
  MATCH (n:Person)
  SET n.dept = 'AWS'
  ```

  此查询需要 `ReadDataViaQuery`、`WriteDataViaQuery` 和 `DeleteDataViaQuery` 访问权限。它会读取所有标签为“Person”的节点，然后向它们添加一个带有键 `dept` 和值 `AWS` 的新属性，或者如果该 `dept` 属性已经存在，则删除旧值并改为插入 `AWS`。此外，如果要设置的值为 `null`，`SET` 会完全删除该属性。

  由于 `SET` 子句在某些情况下可能需要删除现有值，因此它始终需要 `DeleteDataViaQuery` 权限以及 `ReadDataViaQuery` 和 `WriteDataViaQuery` 权限。

- ```
  MATCH (n:Person)
  DETACH DELETE n
  ```

  此查询需要 `ReadDataViaQuery` 和 `DeleteDataViaQuery` 权限。它会找到所有带有标签 `Person` 的节点，并将它们以及连接到这些节点的边缘和任何关联的标签和属性一起删除。

- ```
  MERGE (n:Person {name: 'John'})-[:knows]->(:Person {name: 'Peter'})
  RETURN n
  ```

  此查询需要 `ReadDataViaQuery` 和 `WriteDataViaQuery` 权限。`MERGE` 子句要么匹配指定的模式，要么创建该模式。因为当模式不匹配时，可能会发生写入操作，因此需要写入权限以及读取权限。

Neptune IAM 数据访问策略语句中提供的操作

请注意，Neptune 数据访问操作具有前缀 `neptune-db:`，而 Neptune 中的管理操作具有前缀 `rds:`。

IAM 中的数据资源的 Amazon 资源名称 (ARN) 与创建时分配给集群的 ARN 不同。您必须按照指定数据资源中所示构造 ARN。此类数据资源 ARN 可使用通配符以包含多个资源。

数据访问策略声明还可以包含 ne ptune-db: QueryLanguage 条件键，以限制按查询语言进行访问。

从版本：1.2.0.0（2022 年 7 月 21 日）开始，Neptune 支持将权限限制为一个或多个特定的 Neptune 操作。这提供了比以前更精细的访问控制。

> **Important**
>
> - 对 IAM policy 的更改将需要长达 10 分钟才能应用于指定的 Neptune 资源。
> - 应用于 Neptune 数据库集群的 IAM policy 将应用于集群中的所有实例。

基于查询的数据访问操作

> **Note**
>
> 运行给定查询需要什么权限并不总是显而易见的，因为根据查询处理的数据，查询可能会采取多个操作。请参阅使用查询操作了解更多信息。

`neptune-db:ReadDataViaQuery`

`ReadDataViaQuery` 允许用户通过提交查询从 Neptune 数据库中读取数据。

操作组：只读、读写。

操作上下文键：`neptune-db:QueryLanguage`。

所需资源：数据库。

`neptune-db:WriteDataViaQuery`

`WriteDataViaQuery` 允许用户通过提交查询将数据写入 Neptune 数据库。

操作组：读写。

操作上下文键：neptune-db:QueryLanguage。

所需资源：数据库。

neptune-db:DeleteDataViaQuery

DeleteDataViaQuery 允许用户通过提交查询从 Neptune 数据库中删除数据。

操作组：读写。

操作上下文键：neptune-db:QueryLanguage。

所需资源：数据库。

neptune-db:GetQueryStatus

GetQueryStatus 允许用户检查所有活动查询的状态。

操作组：只读、读写。

操作上下文键：neptune-db:QueryLanguage。

所需资源：数据库。

neptune-db:GetStreamRecords

GetStreamRecords 允许用户从 Neptune 提取流记录。

操作组：读写。

操作上下文键：neptune-db:QueryLanguage。

所需资源：数据库。

neptune-db:CancelQuery

CancelQuery 允许用户取消查询。

操作组：读写。

所需资源：数据库。

常规数据访问操作

neptune-db:GetEngineStatus

GetEngineStatus 允许用户检查 Neptune 引擎的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:GetStatisticsStatus

GetStatisticsStatus 允许用户检查为数据库收集的统计数据的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:GetGraphSummary

GetGraphSummary 图形摘要 API 允许您检索图形的只读摘要。

操作组：只读、读写。

所需资源：数据库。

neptune-db:ManageStatistics

ManageStatistics 允许用户管理数据库的统计数据收集。

操作组：读写。

所需资源：数据库。

neptune-db:DeleteStatistics

DeleteStatistics 允许用户删除数据库中的所有统计数据。

操作组：读写。

所需资源：数据库。

neptune-db:ResetDatabase

ResetDatabase 允许用户获取重置所需的令牌，并重置 Neptune 数据库。

操作组：读写。

所需资源：数据库。

批量加载程序数据访问操作

neptune-db:StartLoaderJob

StartLoaderJob 允许用户启动批量加载程序任务。

操作组：读写。

所需资源：数据库。

neptune-db:GetLoaderJobStatus

GetLoaderJobStatus 允许用户检查批量加载程序任务的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:ListLoaderJobs

ListLoaderJobs 允许用户列出所有批量加载程序任务。

操作组：仅列出、只读、读写。

所需资源：数据库。

neptune-db:CancelLoaderJob

CancelLoaderJob 允许用户取消加载程序任务。

操作组：读写。

所需资源：数据库。

机器学习数据访问操作

neptune-db:StartMLDataProcessingJob

StartMLDataProcessingJob 允许用户启动 Neptune ML 数据处理任务。

操作组：读写。

所需资源：数据库。

neptune-db:StartMLModelTrainingJob

StartMLModelTrainingJob 允许用户启动 ML 模型训练任务。

操作组：读写。

所需资源：数据库。

neptune-db:StartMLModelTransformJob

StartMLModelTransformJob 允许用户启动 ML 模型转换任务。

操作组：读写。

所需资源：数据库。

neptune-db:CreateMLEndpoint

CreateMLEndpoint 允许用户创建 Neptune 机器学习端点。

操作组：读写。

所需资源：数据库。

neptune-db:GetMLDataProcessingJobStatus

GetMLDataProcessingJobStatus 允许用户检查 Neptune ML 数据处理任务的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:GetMLModelTrainingJobStatus

GetMLModelTrainingJobStatus 允许用户检查 Neptune ML 模型训练任务的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:GetMLModelTransformJobStatus

GetMLModelTransformJobStatus 允许用户检查 Neptune ML 模型转换任务的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:GetMLEndpointStatus

GetMLEndpointStatus 允许用户检查 Neptune ML 端点的状态。

操作组：只读、读写。

所需资源：数据库。

neptune-db:ListMLDataProcessingJobs

ListMLDataProcessingJobs 允许用户列出所有 Neptune ML 数据处理任务。

操作组：仅列出、只读、读写。

所需资源：数据库。

neptune-db:ListMLModelTrainingJobs

ListMLModelTrainingJobs 允许用户列出所有 Neptune ML 模型训练任务。

操作组：仅列出、只读、读写。

所需资源：数据库。

neptune-db:ListMLModelTransformJobs

ListMLModelTransformJobs 允许用户列出所有 ML 模型转换任务。

操作组：仅列出、只读、读写。

所需资源：数据库。

neptune-db:ListMLEndpoints

ListMLEndpoints 允许用户列出所有 Neptune ML 端点。

操作组：仅列出、只读、读写。

所需资源：数据库。

neptune-db:CancelMLDataProcessingJob

CancelMLDataProcessingJob 允许用户取消 Neptune ML 数据处理任务。

操作组：读写。

所需资源：数据库。

neptune-db:CancelMLModelTrainingJob

CancelMLModelTrainingJob 允许用户取消 Neptune ML 模型训练任务。

操作组：读写。

所需资源：数据库。

neptune-db:CancelMLModelTransformJob

CancelMLModelTransformJob 允许用户取消 Neptune ML 模型转换任务。

操作组：读写。

所需资源：数据库。

neptune-db:DeleteMLEndpoint

DeleteMLEndpoint 允许用户删除 Neptune 机器学习端点。

操作组：读写。

所需资源：数据库。

在 Neptune IAM 数据访问策略语句中指定资源

数据资源（如数据操作）具有 neptune-db: 前缀。

在 Neptune 数据访问策略中，您可以按以下格式在 ARN 中指定要授予访问权限的数据库集群：

```
arn:aws:neptune-db:region:account-id:cluster-resource-id/*
```

这样的资源 ARN 包含以下部分：

- *region*是 Amazon Neptune 数据库集群的 AWS 区域。
- *account-id* 是数据库集群的 AWS 账号。
- *cluster-resource-id* 是数据库集群的资源 ID。

> **Important**
>
> cluster-resource-id 不同于集群标识符。要在 Neptune 中查找群集资源 ID AWS Management Console，请在 “配置” 部分中查找相关数据库集群。

Neptune IAM 数据访问策略语句中提供的条件键

使用条件键，您可以在 IAM policy 语句中指定条件，这样该语句仅在条件为 true 时才生效。

您可以在 Neptune 数据访问策略语句中使用的条件键分为以下几类：

- 全局条件键 -下面列出了 Neptune 在数据访问策略声明中支持的 AWS 全局条件键的子集。
- 特定于服务的条件键 – 这些键由 Neptune 定义，专门用于数据访问策略语句中。目前只有一个 neptune-db: QueryLanguage，它只有在使用特定的查询语言时才授予访问权限。

AWS Neptune 在数据访问策略声明中支持的全局条件上下文密钥

下表列出了 Amazon Neptune 支持在数据访问策略语句中使用的 AWS 全局条件上下文键的子集：

可在数据访问策略语句中使用的全局条件键

条件键	描述	类型
aws:CurrentTime	按请求的当前日期和时间筛选访问。	String
aws:EpochTime	按请求的日期和时间（以 UNIX 纪元值表示）筛选访问。	Numeric
aws:PrincipalAccount	按发出请求的主体所属的账户筛选访问。	String
aws:PrincipalArn	按发出请求的主体的 ARN 筛选访问。	String
aws:PrincipalIsAWSService	仅当呼叫由 AWS 服务主体直接拨打时才允许访问。	Boolean
aws:PrincipalOrgID	根据请求委托人所属的 Organizations 中的 AWS 组织标识符筛选访问权限。	String
aws:PrincipalOrgPaths	按 AWS Organizations 路径筛选提出请求的委托人的访问权限。	String
aws:PrincipalTag	按附加到发出请求的主体的标签筛选访问。	String

条件键	描述	类型
aws:PrincipalType	按发出请求的主体的类型筛选访问。	String
aws:RequestedRegion	按请求中调用的 AWS 区域筛选访问权限。	String
aws:SecureTransport	仅当使用 SSL 发送请求时才允许访问。	Boolean
aws:SourceIp	按请求者的 IP 地址筛选访问。	String
aws:TokenIssueTime	按颁发临时安全凭证的日期和时间筛选访问。	String
aws:UserAgent	按请求者的客户端应用程序筛选访问。	String
aws:userid	按请求者的主体标识符筛选访问。	String
aws:ViaAWSService	仅当 AWS 服务代表您提出请求时才允许访问。	Boolean

Neptune 特定于服务的条件键

Neptune 针对 IAM policy 支持以下特定于服务的条件键：

Neptune 特定于服务的条件键

条件键	描述	类型
neptune-db:QueryLanguage	按所使用的查询语言筛选数据访问权限。 有效值包括 Gremlin、OpenCypher 和 Sparql。 支持的操作为 ReadDataViaQuery 、WriteDataViaQuery 、DeleteDataViaQuery 、GetQueryStatus 和 CancelQuery 。	String

Neptune IAM 数据访问策略示例

以下示例展示了如何创建自定义 IAM policy，这些策略使用 Neptune 引擎发行版 1.2.0.0 中引入的数据面板 API 和操作的精细访问控制。

策略示例：允许不受限制地访问 Neptune 数据库集群中的数据

以下示例策略允许 IAM 用户使用 IAM 数据库身份验证连接到 Neptune 数据库集群，并使用“*”字符匹配所有可用的操作。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

以上示例包含采用特定于 Neptune IAM 身份验证的格式的资源 ARN。要构造 ARN，请参阅指定数据资源。请注意，用于 IAM 授权 Resource 的 ARN 与创建时分配给集群的 ARN 不同。

允许对 Neptune 数据库集群进行只读访问的策略示例

以下策略授予对 Neptune 数据库集群中数据的完全只读访问权限：

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:Read*",
        "neptune-db:Get*",
        "neptune-db:List*"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
```

```
}
```

策略示例：拒绝对 Neptune 数据库集群的所有访问权限

默认 IAM 操作是拒绝对数据库集群的访问，除非授予 `Allow` 效果。但是，以下策略拒绝特定 AWS 账户和区域对数据库集群的所有访问权限，然后优先于任何`Allow`效果。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

策略示例：通过查询授予读取访问权限

以下策略仅授予使用查询从 Neptune 数据库集群读取数据的权限：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:ReadDataViaQuery",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

仅允许 Gremlin 查询的策略示例

以下策略使用 `neptune-db:QueryLanguage` 条件键授予仅使用 Gremlin 查询语言查询 Neptune 的权限：

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "neptune-db:QueryLanguage": "Gremlin"
        }
      }
    }
  ]
}
```

策略示例：允许除 Neptune ML 模型管理之外的所有访问权限

以下策略授予对 Neptune 图形操作的完全访问权限，但 Neptune ML 模型管理特征除外：

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:CancelLoaderJob",
        "neptune-db:CancelQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:DeleteStatistics",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetLoaderJobStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:GetStatisticsStatus",
        "neptune-db:GetStreamRecords",
        "neptune-db:ListLoaderJobs",
        "neptune-db:ManageStatistics",
        "neptune-db:ReadDataViaQuery",
        "neptune-db:ResetDatabase",
        "neptune-db:StartLoaderJob",
        "neptune-db:WriteDataViaQuery"
      ],
```

```
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

允许访问 Neptune ML 模型管理的策略示例

此策略授予对 Neptune ML 模型管理特征的访问权限：

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:CancelMLDataProcessingJob",
        "neptune-db:CancelMLModelTrainingJob",
        "neptune-db:CancelMLModelTransformJob",
        "neptune-db:CreateMLEndpoint",
        "neptune-db:DeleteMLEndpoint",
        "neptune-db:GetMLDataProcessingJobStatus",
        "neptune-db:GetMLEndpointStatus",
        "neptune-db:GetMLModelTrainingJobStatus",
        "neptune-db:GetMLModelTransformJobStatus",
        "neptune-db:ListMLDataProcessingJobs",
        "neptune-db:ListMLEndpoints",
        "neptune-db:ListMLModelTrainingJobs",
        "neptune-db:ListMLModelTransformJobs",
        "neptune-db:StartMLDataProcessingJob",
        "neptune-db:StartMLModelTrainingJob",
        "neptune-db:StartMLModelTransformJob"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

策略示例：授予完全查询访问权限

以下策略授予对 Neptune 图形查询操作的完全访问权限，但不授予对快速重置、流、批量加载程序、Neptune ML 模型管理等特征的完全访问权限：

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:CancelQuery"
      ],
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

策略示例：仅授予对 Gremlin 查询的完全访问权限

以下策略授予使用 Gremlin 查询语言对 Neptune 图形查询操作的完全访问权限，但不授予对其它语言的查询的完全访问权限，也不授予对快速重置、流、批量加载程序、Neptune ML 模型管理等特征的完全访问权限：

```
{
  "Version":"2012-10-17",
  "Statement":[
    {
      "Effect": "Allow",
      "Action": [
        "neptune-db:ReadDataViaQuery",
        "neptune-db:WriteDataViaQuery",
        "neptune-db:DeleteDataViaQuery",
        "neptune-db:GetEngineStatus",
        "neptune-db:GetQueryStatus",
        "neptune-db:CancelQuery"
      ],
      "Resource": [
        "arn:aws:neptune-db:us-east-1:123456789012:cluster-ABCD1234EFGH5678IJKL90MNOP/
*"
      ],
      "Condition": {
```

```
        "StringEquals": {
           "neptune-db:QueryLanguage":"Gremlin"
        }
      }
    }
  ]
}
```

策略示例：授予完全访问权限，但快速重置除外

以下策略授予对 Neptune 数据库集群的完全访问权限，但使用快速重置除外：

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    },
    {
      "Effect": "Deny",
      "Action": "neptune-db:ResetDatabase",
      "Resource": "arn:aws:neptune-db:us-east-1:123456789012:cluster-
ABCD1234EFGH5678IJKL90MNOP/*"
    }
  ]
}
```

对 Neptune 使用服务相关角色

亚马逊 Neptune 使用 AWS Identity and Access Management (IAM) 服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 Neptune 直接相关。服务相关角色由 Neptune 预定义，包括该服务代表您调用 AWS 其他服务所需的所有权限。

> **Important**
>
> 对于某些管理特征，Amazon Neptune 使用与 Amazon RDS 共享的操作技术。这包括服务相关角色 和管理 API 权限。

服务相关角色可让您更轻松地使用 Neptune，因为您不必手动添加必要的权限。Neptune 定义其服务相关角色的权限，除非另外定义，否则只有 Neptune 可以代入该角色。定义的权限包括信任策略和权限策略，而且权限策略不能附加到任何其它 IAM 实体。

只有在首先删除角色的相关资源后，才能删除角色。这将保护您的 Neptune 资源，因为您不会无意中删除对资源的访问权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的AWS 服务并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

Neptune 的服务相关角色权限

Neptune 使用AWSServiceRoleForRDS服务相关角色允许 Neptune 和 Amazon RDS 代表您的数据库实例调用 AWS 服务。AWSServiceRoleForRDS 服务相关角色信任 rds.amazonaws.com 服务来代入角色。

角色权限策略允许 Neptune 对指定的资源完成以下操作：

- 对 ec2 的操作：
  - AssignPrivateIpAddresses
  - AuthorizeSecurityGroupIngress
  - CreateNetworkInterface
  - CreateSecurityGroup
  - DeleteNetworkInterface
  - DeleteSecurityGroup
  - DescribeAvailabilityZones
  - DescribeInternetGateways
  - DescribeSecurityGroups
  - DescribeSubnets
  - DescribeVpcAttribute
  - DescribeVpcs
  - ModifyNetworkInterfaceAttribute
  - RevokeSecurityGroupIngress
  - UnassignPrivateIpAddresses
- 对 sns 的操作：

  - ListTopic
  - Publish
- 对 cloudwatch 的操作：
  - PutMetricData
  - GetMetricData
  - CreateLogStream
  - PullLogEvents
  - DescribeLogStreams
  - CreateLogGroup

> **Note**
>
> 您必须配置权限，允许 IAM 实体（如用户、组或角色）创建、编辑或删除服务相关角色。您可能会遇到以下错误消息：
> Unable to create the resource. Verify that you have permission to create service linked role。Otherwise wait and try again later.
> 如果您看到此消息，请确保您已启用以下权限：
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> 有关更多信息，请参阅《IAM 用户指南》中的服务相关角色权限。

为 Neptune 创建服务相关角色

您无需手动创建服务相关角色。当您创建实例或集群时，Neptune 将为您创建服务相关角色。

> **Important**
>
> 要了解更多信息，请参阅《IAM 用户指南》中的我的 IAM 账户中出现新角色。

如果删除此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您创建实例或集群时，Neptune 将再次为您创建服务相关角色。

编辑 Neptune 的服务相关角色

Neptune 不允许您编辑 AWSServiceRoleForRDS 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。但是可以使用 IAM 编辑角色描述。有关更多信息，请参阅《IAM 用户指南》中的编辑服务相关角色。

删除 Neptune 的服务相关角色

如果不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样就没有未被主动监控或维护的未使用实体。但是，您必须先删除所有实例和群集才能删除关联的服务相关角色。

在删除之前清除服务相关角色

必须先确认服务相关角色没有活动会话并删除该角色使用的任何资源，然后才能使用 IAM 删除服务相关角色。

在 IAM 控制台中检查服务相关角色是否具有活动会话

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择角色。然后选择 AWSServiceRoleForRDS 角色的名称（不是复选框）。
3. 在所选角色的 Summary 页面上，选择 Access Advisor 选项卡。
4. 在访问顾问选项卡查看服务相关角色的近期活动。

> **Note**
>
> 如果您不确定 Neptune 是否正在使用 AWSServiceRoleForRDS 角色，可以尝试删除该角色。如果服务正在使用该角色，则删除操作会失败，并且您可以查看正在使用该角色的

区域。如果该角色已被使用，则您必须等待会话结束，然后才能删除该角色。您无法撤销服务相关角色对会话的权限。

如果您要删除 AWSServiceRoleForRDS 角色，必须首先删除您的所有实例和集群。

删除所有实例

使用以下过程之一删除每个实例。

删除一个实例 (控制台)

1. 通过以下网址打开 Amazon RDS 控制台：https://console.aws.amazon.com/rds/。
2. 在导航窗格中，选择实例。
3. 在实例列表中，选择要删除的实例。
4. 选择实例操作，然后选择删除。
5. 如果系统提示您 Create final Snapshot? (是否创建最终快照?)，请选择 Yes (是) 或 No (否)。
6. 如果您在上一步中选择了 Yes (是)，请为 Final snapshot name (最终快照名称) 输入最终快照的名称。
7. 选择删除。

删除实例 (AWS CLI)

请参阅 AWS CLI Command Reference 中的 `delete-db-instance`。

删除一个实例 (API)

请参阅 `DeleteDBInstance`。

删除所有集群

使用以下过程之一删除单个集群，然后对您的每个集群重复该过程。

删除一个集群 (控制台)

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在 Clusters 列表中，选择要删除的群集。
3. 选择 Cluster Actions，然后选择 Delete。

4. 选择删除。

删除一个集群 (CLI)

请参阅 AWS CLI Command Reference 中的 delete-db-cluster。

删除一个集群 (API)

请参阅 DeleteDBCluster。

可以使用 IAM 控制台、IAM CLI 或 IAM API 删除 AWSServiceRoleForRDS 服务相关角色。有关更多信息，请参见《IAM 用户指南》中的删除服务相关角色。

使用临时凭证进行的 IAM 身份验证

Amazon Neptune 支持使用临时凭证进行 IAM 身份验证。

您可以使用代入的角色通过 IAM 身份验证策略（如上面各节中的示例策略之一）进行身份验证。

如果您使用的是临时凭证，则除了 AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY 和 SERVICE_REGION 之外，您还必须指定 AWS_SESSION_TOKEN。

> **Note**
>
> 临时凭证（包括会话令牌）将在指定时间间隔后到期。
> 在请求新凭证时，您必须更新您的会话令牌。有关更多信息，请参阅使用临时安全证书请求对 AWS 资源的访问权限。

以下各节介绍如何允许访问和检索临时凭证。

使用临时凭证进行身份验证

1. 创建有权访问 Neptune 集群的 IAM 角色。有关创建此角色的信息，请参阅 the section called “IAM policy 的类型”。
2. 向角色添加允许访问凭证的信任关系。

   检索临时凭证（包括 AWS_ACCESS_KEY_ID、AWS_SECRET_ACCESS_KEY 和 AWS_SESSION_TOKEN）。

3. 使用临时凭证连接到 Neptune 集群并签署请求。有关连接和签名请求的更多信息，请参阅the section called “连接和签名”。

可通过多种方法检索临时凭证，具体取决于环境。

主题

- 使用 AWS CLI获取临时凭证
- 为 Neptune 的 IAM 身份验证设置 AWS Lambda
- 为 Neptune IAM 身份验证设置 Amazon EC2

使用 AWS CLI获取临时凭证

要使用 AWS Command Line Interface (AWS CLI) 获取证书，首先需要添加信任关系，向将运行该 AWS CLI 命令的 AWS 用户授予代入该角色的权限。

向 Neptune IAM 身份验证角色添加以下信任关系。如果您没有 Neptune IAM 身份验证角色，请参阅the section called “IAM policy 的类型”。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:user/test"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

有关向角色添加信任关系的信息，请参阅《AWS Directory Service 管理指南》中的编辑现有角色的信任关系。

如果 Neptune 策略尚未附加到角色，请创建一个新的角色。附加 Neptune IAM 身份验证策略，然后添加信任策略。有关创建新角色的信息，请参阅创建新角色。

> **Note**
>
> 以下各节假设您已经 AWS CLI 安装了。

要 AWS CLI 手动运行

1. 使用 AWS CLI输入以下命令以请求凭证。将角色 ARN、会话名称和配置文件替换为您自己的值。

```
aws sts assume-role  --role-arn  arn:aws:iam::123456789012:role/NeptuneIAMAuthRole
 --role-session-name test --profile testprofile
```

2. 下面是此命令中的示例输出。Credentials 部分包含您需要的值。

> **Note**
>
> 记录 Expiration 值，因为在此次之后您将需要获取新凭证。

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA3XFRBF535PLBIFPI4:s3-access-example",
        "Arn": "arn:aws:sts::123456789012:assumed-role/xaccounts3access/s3-access-
example"
    },
    "Credentials": {
        "SecretAccessKey": "9drTJvcXLB89EXAMPLELB8923FB892xMFI",
        "SessionToken": "AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/qwjzP2iEXAMPLEbw/
m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8BRi
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=",
        "Expiration": "2016-03-15T00:05:07Z",
        "AccessKeyId": "ASIAJEXAMPLEXEG2JICEA"
    }
}
```

3. 使用返回的凭证设置环境变量。

```
export AWS_ACCESS_KEY_ID=ASIAJEXAMPLEXEG2JICEA
export AWS_SECRET_ACCESS_KEY=9drTJvcXLB89EXAMPLELB8923FB892xMFI
export AWS_SESSION_TOKEN=AQoXdzELDDY//////////
wEaoAK1wvxJY12r2IrDFT2IvAzTCn3zHoZ7YNtpiQLF0MqZye/qwjzP2iEXAMPLEbw/
m3hsj8VBTkPORGvr9jM5sgP+w9IZWZnU+LWhmg
+a5fDi2oTGUYcdg9uexQ4mtCHIHfi4citgqZTgco40Yqr4lIlo4V2b2Dyauk0eYFNebHtYlFVgAUj
+7Indz3LU0aTWk1WKIjHmmMCIoTkyYp/k7kUG7moeEYKSitwQIi6Gjn+nyzM
+PtoA3685ixzv0R7i5rjQi0YE0lf1oeie3bDiNHncmzosRM6SFiPzSvp6h/32xQuZsjcypmwsPSDtTPYcs0+YN/8BRi
IcrxSpnWEXAMPLEXSDFTAQAM6Dl9zR0tXoybnlrZIwMLlMi1Kcgo5OytwU=

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

4. 使用下列方法之一进行连接。

   - the section called “Gremlin 控制台”
   - the section called “Gremlin Java”
   - the section called “SPARQL Java (RDF4J 和 Jena) ”
   - the section called “Python 示例”

使用脚本获取凭证

1. 运行以下命令以安装 jq 命令。该脚本使用此命令来解析 AWS CLI 命令的输出。

```
sudo yum -y install jq
```

2. 在文本编辑器中创建一个名为 `credentials.sh` 的文件并添加以下文本。将服务区域、角色 ARN、会话名称和配置文件替换为您自己的值。

```
#!/bin/bash

creds_json=$(aws sts assume-role  --role-arn  arn:aws:iam::123456789012:role/
NeptuneIAMAuthRole --role-session-name test --profile testprofile)
```

```
export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .Credentials.AccessKeyId |tr -d
 '"')
export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" |
 jq .Credentials.SecretAccessKey| tr -d '"')
export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Credentials.SessionToken|tr -d
 '"')

export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
central-1 or
                      sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
west-3 or eu-central-1 or me-south-1 or
                      me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
 ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
 or
                      cn-north-1 or cn-northwest-1 or
                      us-gov-east-1 or us-gov-west-1
```

3. 使用下列方法之一进行连接。

- the section called “Gremlin 控制台”
- the section called “Gremlin Java”
- the section called “SPARQL Java（RDF4J 和 Jena）”
- the section called “Python 示例”

为 Neptune 的 IAM 身份验证设置 AWS Lambda

AWS Lambda 每次运行 Lambda 函数时都会自动包含证书。

首先，您添加信任关系，该信任关系向 Lambda 服务授予代入此角色的权限。

向 Neptune IAM 身份验证角色添加以下信任关系。如果您没有 Neptune IAM 身份验证角色，请参阅the section called “IAM policy 的类型”。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

有关向角色添加信任关系的信息，请参阅《AWS Directory Service管理指南》中的编辑现有角色的信任关系。

如果 Neptune 策略尚未附加到角色，请创建一个新的角色。附加 Neptune IAM 身份验证策略，然后添加信任策略。有关创建新角色的信息，请参阅《AWS Directory Service 管理指南》中的创建新角色。

通过 Lambda 访问 Neptune

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为 https://console.aws.amazon.com/lambda/。
2. 为 Python 版本 3.6 创建新的 Lambda 函数。
3. 将 `AWSLambdaVPCAccessExecutionRole` 角色分配给 Lambda 函数。这是访问 Neptune 资源（仅 VPC）所必需的。
4. 将 Neptune 身份验证 IAM 角色分配给 Lambda 函数。

   有关授予权限的更多信息，请参阅《AWS Lambda 开发人员指南》中的 AWS 权限。
5. 将 IAM 身份验证 Python 示例复制到 Lambda 函数代码中。

   有关示例和示例代码的更多信息，请参阅the section called “Python 示例”。

为 Neptune IAM 身份验证设置 Amazon EC2

Amazon EC2 允许您使用实例配置文件自动提供凭证。有关更多信息，请参阅《IAM 用户指南》中的使用实例配置文件。

首先，您添加信任关系，该信任关系向 Amazon EC2 服务授予代入此角色的权限。

向 Neptune IAM 身份验证角色添加以下信任关系。如果您没有 Neptune IAM 身份验证角色，请参阅the section called “IAM policy 的类型”。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

有关向角色添加信任关系的信息，请参阅《AWS Directory Service 管理指南》中的编辑现有角色的信任关系。

如果 Neptune 策略尚未附加到角色，请创建一个新的角色。附加 Neptune IAM 身份验证策略，然后添加信任策略。有关创建新角色的信息，请参阅《AWS Directory Service 管理指南》中的创建新角色。

使用脚本获取凭证

1. 运行以下命令以安装 jq 命令。脚本使用此命令解析 curl 命令的输出。

   ```
   sudo yum -y install jq
   ```

2. 在文本编辑器中创建一个名为 `credentials.sh` 的文件并添加以下文本。将服务区域替换为您自己的值。

   ```
   role_name=$( curl -s http://169.254.169.254/latest/meta-data/iam/security-
   credentials/ )
   creds_json=$(curl -s http://169.254.169.254/latest/meta-data/iam/security-
   credentials/${role_name})

   export AWS_ACCESS_KEY_ID=$(echo "$creds_json" | jq .AccessKeyId |tr -d '"')
   export AWS_SECRET_ACCESS_KEY=$(echo "$creds_json" | jq .SecretAccessKey| tr -d '"')
   export AWS_SESSION_TOKEN=$(echo "$creds_json" | jq .Token|tr -d '"')

   export SERVICE_REGION=us-east-1 or us-east-2 or us-west-1 or us-west-2 or ca-
   central-1 or
                         sa-east-1 or eu-north-1 or eu-west-1 or eu-west-2 or eu-
   west-3 or eu-central-1 or me-south-1 or
                         me-central-1 or il-central-1 or af-south-1 or ap-east-1 or
    ap-northeast-1 or ap-northeast-2 or ap-southeast-1 or ap-southeast-2 or ap-south-1
    or
                         cn-north-1 or cn-northwest-1 or
   ```

```
                    us-gov-east-1 or us-gov-west-1
```

3. 使用 `source` 命令在 bash shell 中运行脚本：

```
source credentials.sh
```

更好的办法是将此脚本中的命令添加到 EC2 实例上的 `.bashrc` 文件中，以便在您登录时自动调用这些命令，从而使临时凭证可用于 Gremlin 控制台。

4. 使用下列方法之一进行连接。

- the section called “Gremlin 控制台”
- the section called “Gremlin Java”
- the section called “SPARQL Java（RDF4J 和 Jena）”
- the section called “Python 示例”

记录和监控 Amazon Neptune 资源

Amazon Neptune 支持多种用于监控性能和使用情况的方法。

- 集群状态 – 检查 Neptune 集群的图形数据库引擎的运行状况。有关更多信息，请参阅 the section called “实例状态”。
- 亚马逊 CloudWatch — Neptune 会自动向警报发送指标 CloudWatch 并支持 CloudWatch 警报。有关更多信息，请参阅 the section called “使用 CloudWatch”。
- 审计日志文件 – 使用 Neptune 控制台来查看、下载或监视数据库日志文件。有关更多信息，请参阅 the section called “使用 Neptune 审计日志”。
- 将日志发布到 Amazon CloudWatch 日志 — 您可以将 Neptune 数据库集群配置为将审核日志数据发布到 Ama CloudWatch zon Logs 中的日志组。借助 CloudWatch 日志，您可以对日志数据进行实时分析，用于 CloudWatch 创建警报和查看指标，并使用 CloudWatch 日志将日志记录存储在高度耐用的存储中。有关更多信息，请参阅 Neptun CloudWatch e 日志。
- AWS CloudTrail— Neptune 支持使用 API 日志记录。 CloudTrail有关更多信息，请参阅 the section called “使用记录 Neptune API 调用 AWS CloudTrail”。
- 标记 – 使用标签向 Neptune 资源添加元数据并基于标签跟踪使用情况。有关更多信息，请参阅 the section called “给 Neptune 资源加标签”。

Amazon Neptune 的合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在这些基础上 AWS 部署以安全性和合规性为重点的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规性 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

Amazon Neptune 中的恢复能力

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

只能在 Amazon VPC 中创建 Amazon Neptune 数据库集群，该集群在至少两个可用区中包含至少两个子网。通过跨至少两个可用区分配您的集群实例，Neptune 可帮助确保数据库集群中有可用的实例，避免出现可用区故障。Neptune 数据库集群的集群卷始终跨三个可用区提供持久性存储，数据丢失的可能性很小。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

将现有图形迁移到 Amazon Neptune

有许多工具和技术可以帮助您将现有图形数据从其它数据存储迁移到 Amazon Neptune。

简单的迁移工作流程涉及以下步骤：

- 将数据从其现有存储导出到 Amazon Simple Storage Service (Amazon S3)。
- 清理数据并进行格式化以供导入。
- 使用Neptune 批量加载程序将数据加载到 Neptune 数据库集群中。
- 将您的 Gremlin 或 SPARQL 应用程序配置为使用 Neptune 提供的相应端点。

> **Note**
>
> 您的 Neptune 集群必须在您的应用程序可以访问的 VPC 中运行。

有一些方法可以简化和自动执行其中一些步骤，具体取决于数据的存储位置：

主题

- 从 Neo4j 迁移到 Amazon Neptune
- 将现有图形从 Apache TinkerPop Gremlin 服务器迁移到 Amazon Neptune
- 将现有图形从 RDF 三重存储迁移到 Amazon Neptune
- 使用 AWS Database Migration Service（AWS DMS）从关系数据库或 NoSQL 数据库迁移到 Amazon Neptune
- 从 Blazegraph 迁移到 Amazon Neptune

从 Neo4j 迁移到 Amazon Neptune

Neo4j 和 Amazon Neptune 都是图形数据库，专为支持带标签的属性图数据模型的在线事务图形工作负载而设计。这些相似之处使 Neptune 成为寻求迁移其当前 Neo4j 应用程序的客户的常见选择。但是，这些迁移并不仅仅是直接迁移，因为两个数据库之间在语言和特征支持、操作特性、服务器架构和存储能力方面存在差异。

本页概述了迁移过程，并提出了在将 Neo4j 图形应用程序迁移到 Neptune 之前需要考虑的事项。这些注意事项通常适用于任何 Neo4j 图形应用程序，无论是由社区、企业还是 Aura 数据库提供支持。尽管每种解决方案都是独一无二的，可能需要额外的过程，但所有迁移都遵循相同的一般模式。

以下各节中描述的每个步骤都包括简化迁移过程的注意事项和建议。此外，还有开源工具和描述流程的博客文章，以及包含推荐架构选项的特征兼容性部分。

主题

- 有关从 Neo4j 迁移到 Neptune 的一般信息
- 准备从 Neo4j 迁移到 Neptune
- 从 Neo4j 迁移到 Neptune 时预调配基础设施
- 从 Neo4j 到 Neptune 的数据迁移
- 应用程序从 Neo4j 迁移到 Neptune
- Neptune 与 Neo4j 的兼容性
- 重写 Cypher 查询以在 Neptune 上的 openCypher 中运行
- 用于从 Neo4j 迁移到 Neptune 的资源

有关从 Neo4j 迁移到 Neptune 的一般信息

借助 Neptune 对 openCypher 查询语言的支持，您可以将大多数使用 Bolt 协议或 HTTPS 的 Neo4j 工作负载迁移到 Neptune。但是，openCypher 是一个开源规范，它包含其它数据库（例如 Neo4j）支持的大部分（但不是全部）功能。

尽管在许多方面兼容，但 Neptune 并不是 Neo4j 的直接替代。Neptune 是一项完全托管式图形数据库服务，具有高可用性和高耐久性等企业级特征，在架构上与 Neo4j 不同。Neptune 基于实例，具有一个主写入器实例和最多 15 个只读副本实例（可让您水平扩展读取容量）。使用 Neptune 无服务器，您可以根据查询量自动横向扩展和缩减计算容量。这与 Neptune 存储无关，Neptune 存储会随着您添加数据而自动扩展。

Neptune 支持开源 openCypher 标准规范，版本 9。在 AWS，我们相信开源对每个人都有好处，我们致力于将开源的价值带给我们的客户，并为开源社区带来 AWS 的运营卓越性。

但是，许多在 Neo4j 上运行的应用程序也使用非开源且 Neptune 不支持的专有特征。例如，Neptune 不支持 APOC 过程、某些特定于 Cypher 的子句和函数以及 `Char`、`Date` 或 `Duration` 数据类型。Neptune 确实会将缺失的数据类型自动转换为支持的数据类型。

除了 openCypher 之外，Neptune 还支持用于属性图的 Apache TinkerPop Gremlin 查询语言（以及用于 RDF 数据的 SPARQL）。Gremlin 可以在同一个属性图上与 openCypher 互操作，在许多情况下，您可以使用 Gremlin 来提供 openCypher 不提供的功能。以下是两种语言的快速比较：

	openCypher	Gremlin
样式	声明式	命令式

语法	模式匹配
	`Match p=(a)-[:route]->(d)`
	`WHERE a.code='ANC'`
	`RETURN p`
		基于遍历
		`g.V().has('code', 'ANC').`
		`out('route').path().`
		`by(elementMap())`

易于使用	SQL 启发式，非程序员也可阅读	更陡峭的学习曲线，类似于 Java 等编程语言
弹性	低	高
查询支持	基于字符串的查询	客户端库支持的基于字符串的查询或内联代码
客户	HTTPS 和 Bolt	HTTPS 和 Websocket

通常，无需更改数据模型即可从 Neo4j 迁移到 Neptune，因为 Neo4j 和 Neptune 都支持带标签的属性图 (LPG) 数据。但是，Neptune 有一些架构和数据模型差异，您可以利用这些差异来优化性能。例如：

- Neptune ID 被视为一等公民。
- Neptune 使用 AWS Identity and Access Management (IAM) 策略以灵活而精细的方式保护对您的图形数据的访问。

- Neptune 提供了多种使用 Jupyter 笔记本运行查询和可视化结果的方法。Neptune 还可以与第三方可视化工具配合使用。
- 尽管 Neptune 无法替代 Neo4j 图形数据科学 (GDS) 库，但 Neptune 通过各种解决方案支持当今的图形分析。例如，一些示例笔记本演示了如何在 Python 环境中利用 Neptune 与 AWS Pandas SDK 的集成来对图形数据进行分析。

如果您还有问题，请联系 AWS Support 或联系您的 AWS 客户团队。我们会根据您的反馈来确定满足您需求的新特征的优先顺序。

准备从 Neo4j 迁移到 Neptune

迁移方法

将 Neo4j 应用程序迁移到 Neptune 时，我们推荐两种策略之一：要么重构平台，要么重构/转换架构。有关迁移策略的更多信息，请参阅 Stephen Orban 撰写的博客文章将应用程序迁移到云中的 6 种策略。

重构平台方法（有时称为 lift-tinker-and-shift）涉及以下步骤：

- 确定您的应用程序要满足的用例。
- 使用 Neptune 的功能修改现有的图形数据模型和应用程序架构，以最好地满足这些工作负载需求。
- 确定如何将数据、查询和源应用程序的其它部分迁移到目标模型和架构中。

如果这是一个全新的项目，这种逆向工作的方法可以让您将应用程序迁移到您可能设计的这种 Neptune 解决方案中。

相比之下，重构方法涉及：

- 确定现有实施的组件，包括基础设施、数据、查询和应用程序功能。
- 在 Neptune 中寻找可用于构建类似实现的等效项。

这种向前推进的方法试图将一种实现换成另一种实现。

实际上，您很可能会混合使用这两种方法。您可以从一个用例开始，设计目标 Neptune 架构，但然后转向现有的 Neo4j 实现来确定您必须维护的约束和不变量。例如，您可能必须继续与其它外部系统集成，或者继续向图形应用程序的使用者提供特定的 API。利用这些信息，您可以确定哪些数据已经存在可移动到目标模型，哪些数据必须从其它地方获取。

在其它时候，您可以从分析 Neo4j 实现的特定部分开始，以此作为有关您的应用程序打算完成的任务的最佳信息来源。现有应用程序中的这种考古学可以帮助定义一个用例，然后您可以设计这个用例来使用 Neptune 的功能。

无论您是使用 Neptune 构建新的应用程序，还是从 Neo4j 迁移现有应用程序，我们都建议您从用例向后倒推，以设计可满足您业务需求的数据模型、一组查询和应用程序架构。

Neptune 和 Neo4j 之间的架构差异

当客户第一次考虑将应用程序从 Neo4j 迁移到 Neptune 时，通常很容易根据实例大小进行同类比较。但是，Neo4j 和 Neptune 的架构有根本的区别。Neo4j 基于一体化方法，其中数据加载、数据 ETL、应用程序查询、数据存储和管理操作都发生在同一组计算资源中，例如 EC2 实例。

相比之下，Neptune 是一个以 OLTP 为中心的图形数据库，其中，架构将职责分开，且资源是解耦的，因此它们可以动态和独立地扩展。

从 Neo4j 迁移到 Neptune 时，请确定应用程序的数据持久性、可用性和可扩展性要求。Neptune 的集群架构简化了需要高耐久性、高可用性和高可扩展性的应用程序的设计。了解 Neptune 的集群架构后，就可以设计出满足这些要求的 Neptune 集群拓扑了。

Neo4j 的集群架构

许多生产应用程序使用 Neo4j 的因果集群来提供数据持久性、高可用性和可扩展性。Neo4j 的集群架构使用核心服务器和只读副本实例：

- 核心服务器通过使用 Raft 协议复制数据，提供数据持久性和容错能力。
- 只读副本使用事务日志传送，以异步复制高读取吞吐量工作负载的数据。

集群中的每个实例，无论是核心服务器还是只读副本，都包含图形数据的完整副本。

Neptune 的集群架构

Neptune 集群由一个主写入器实例和最多 15 个只读副本实例组成。集群中的所有实例共享与实例分开的相同底层分布式存储服务。

- 主写入器实例协调对于数据库的所有写入操作，并且可以垂直扩展，为不同的写入工作负载提供灵活的支持。它还支持读取操作。
- 只读副本实例支持从底层存储卷进行读取操作，并允许您水平扩展以支持高的读取工作负载。它们还通过充当主实例的失效转移目标来提供高可用性。

> **Note**
>
> 对于繁重的写入工作负载，最好将只读副本实例扩展到与写入器实例相同的大小，以确保读取器能够与数据变化保持一致。

- 随着数据库中数据的增加，底层存储卷自动扩展存储容量，最多可达 128TiB 存储空间。

实例大小是动态且独立的。可以在集群运行时调整每个实例的大小，也可以在集群运行时添加或移除只读副本。

随着需求上升和下降，Neptune 无服务器特征可以自动纵向扩展和缩减您的计算容量。这不仅可以减少管理开销，还可以让您将数据库配置为在不降低性能或要求您过度预调配的情况下处理需求激增的状况。

您可以停止 Neptune 数据库集群最多 7 天。

Neptune 还支持自动扩缩，可根据工作负载自动调整读取器实例的大小。

使用 Neptune 的全球数据库特征，您最多可以在其它 5 个区域中镜像集群。

Neptune 在设计上也是容错的：

- 为集群中的所有实例提供数据存储的集群卷跨单个 AWS 区域中的多个可用区 (AZ)。每个可用区均包含集群数据的一个完整副本。
- 如果主实例变得不可用，Neptune 会自动失效转移到现有的只读副本，数据丢失为零，通常在 30 秒内完成。如果集群中没有现有的只读副本，Neptune 会自动预调配一个新的主实例，同样不会丢失任何数据。

所有这些都意味着，从 Neo4j 因果集群迁移到 Neptune 时，您不必为了获得较高的数据持久性和高可用性而显式设计集群拓扑。这使您可以通过以下几种方式调整集群的大小，以满足预期的读取和写入工作负载以及任何可能增加的可用性要求：

- 要扩展读取操作，请添加只读副本实例或启用 Neptune 无服务器功能。
- 要提高可用性，请将集群中的主实例和只读副本分布在多个可用区 (AZ) 上。
- 要缩短任何失效转移时间，请预调配至少一个可作为主实例的失效转移目标的只读副本实例。您可以通过为每个副本分配优先级，确定在发生故障后将只读副本实例提升为主实例的顺序。最佳实践是确保失效转移目标所具有的实例类在提升为主实例后，能够处理应用程序的写入工作负载。

Neptune 和 Neo4j 之间的数据存储区别

Neptune 使用基于原生四元组模型的图形数据模型。将数据迁移到 Neptune 时，为了以最佳的方式利用 Neptune 提供的分布式和可扩展共享存储，您应该注意数据模型和存储层的架构存在一些差异：

- Neptune 不使用任何显式定义的架构或约束。它允许您动态添加节点、边缘和属性，而无需提前定义架构。Neptune 不限制所存储数据的值和类型，除非在 Neptune 限制中另有说明。作为 Neptune

存储架构的一部分，还会自动对数据编制索引，以处理许多最常见的访问模式。这种存储架构消除了创建和管理数据库架构和索引优化的操作开销。

- Neptune 提供了一种独特的分布式共享存储架构，这种架构可随着数据库存储需求的增长以 10GB 的区块进行扩展，最高可达 128TiB。该存储层可靠、耐用且具有容错性，可在 3 个可用区中复制数据 6 次，每个可用区复制两次。默认情况下，它为所有 Neptune 集群提供了高可用性和容错性的数据存储层。Neptune 的存储架构降低了成本，并且无需为应对未来的数据增长而预调配或过度预调配存储。

在将数据迁移到 Neptune 之前，最好先熟悉 Neptune 的属性图数据模型和事务语义。

Neptune 和 Neo4j 之间的操作差异

Neptune 是一项完全托管式服务，可自动执行您在使用本地或自行管理的数据库（例如 Neo4j Enterprise 或 Community Edition）时必须执行的许多正常操作任务：

- 自动备份 – Neptune 自动备份您的集群卷，并将备份保留您指定的保留期（从 1 到 35 天不等）。这些备份是连续且递增的，因此，您可以快速还原到保留期内的任何时间点。在写入备份数据时，不会发生任何性能影响或数据库服务中断。
- 手动快照 – Neptune 允许您创建数据库集群的存储卷快照，以备份整个数据库集群。然后，这种快照可用于还原数据库、制作数据库副本并在多个账户之间共享。
- 克隆 – Neptune 支持克隆特征，可让您快速创建经济实惠的数据库克隆。克隆使用写入时复制协议，在创建后只需要极少的额外空间。数据库克隆是在不中断原始集群的情况下试用 Neptune 新特征或升级的高效方法。
- 监控 – Neptune 提供了多种方法来监控集群的性能和使用情况，包括：
  - 实例状态
  - 与 Amazon CloudWatch 和 AWS CloudTrail 集成
  - 审计日志功能
  - 事件通知
  - Tagging
- 安全 – Neptune 默认提供安全的环境。集群位于私有 VPC 中，该私有 VPC 可与其它资源进行网络隔离。所有流量均通过 SSL 加密，所有数据都使用 AWS KMS 静态加密。

  此外，Neptune 还与 AWS Identity and Access Management (IAM) 集成以提供身份验证。通过指定 IAM 条件键，您可以使用 IAM policy 对数据操作进行精细的访问控制。

Neptune 和 Neo4j 之间的工具和集成差异

Neptune 的集成和工具架构与 Neo4j 不同，这可能会影响您的应用程序架构。Neptune 使用集群的计算资源来处理查询，但利用其它一流的 AWS 服务来实现诸如全文搜索（使用 OpenSearch）、ETL（使用 Glue）等功能。有关这些集成的完整列表，请参阅Neptune 集成。

从 Neo4j 迁移到 Neptune 时预调配基础设施

Amazon Neptune 集群构建为从三个维度进行扩展：存储、写入容量和读取容量。以下各节讨论了迁移时要考虑的特定选项。

预调配存储

任何 Neptune 集群的存储都是自动预调配的，您无需承担任何管理开销。随着集群的存储需求增加，它会以 10GB 的区块动态调整大小。因此，无需为应对未来的数据增长而估计和预调配或过度预调配存储。

预调配写入容量

Neptune 提供了一个写入器实例，可以将其垂直扩展到 Neptune 定价页面上提供的任何实例大小。在向写入器实例读取和写入数据时，所有事务都符合 ACID 标准，数据隔离如Neptune 中的事务隔离级别中所定义。

为写入器实例选择最佳大小要求运行负载测试，以确定工作负载的最佳实例大小。通过修改数据库实例类，可以随时调整 Neptune 中任何实例的大小。您可以根据并发和平均查询延迟来估算起始实例的大小，如下面的在预调配集群时估算最佳实例大小所述。

预调配读取容量

Neptune 构建为水平扩展只读副本实例，方法是在集群内添加最多 15 个只读副本实例（在 Neptune 全球数据库中可添加更多），也可以垂直扩展到 Neptune 定价页面上提供的任何实例大小。所有 Neptune 只读副本实例都使用相同的底层存储卷，从而以最小的滞后实现数据的透明复制。

除了在 Neptune 集群中启用读取请求的水平扩展外，只读副本还充当写入器实例的失效转移目标，以实现高可用性。有关如何确定集群中只读副本的适当数量和位置的建议，请参阅Amazon Neptune 基本操作指导。

对于连接和工作负载不可预测的应用程序，Neptune 还支持自动扩缩特征，该特征可以根据您指定的标准自动调整 Neptune 副本的数量。

要确定只读副本实例的最佳大小和数量，需要运行负载测试以确定它们必须支持的读取工作负载的特性。通过修改数据库实例类，可以随时调整 Neptune 中任何实例的大小。您可以根据并发和平均查询延迟来估算起始实例的大小，如下一节所述。

使用 Neptune 无服务器根据需要自动扩展读取器和写入器实例

虽然能够估算预期工作负载所需的计算容量通常很有帮助，但您可以配置 Neptune 无服务器特征，以自动纵向扩展和缩减读取和写入容量。这可以帮助您满足峰值需求，同时还可以在需求减少时自动缩减规模。

在预调配集群时估算最佳实例大小

要估算最佳实例大小，需要知道 Neptune 中的平均查询延迟、工作负载何时运行，以及正在处理的并发查询数量。实例大小的粗略估计值可以通过将平均查询延迟乘以并发查询数来计算。这为您提供了处理工作负载所需的平均并发线程数。

Neptune 实例中的每个 vCPU 可以支持两个并发查询线程，因此将线程除以 2 即可得出所需的 vCPU 数量，然后可以将其与 Neptune 定价页面上的相应实例大小相关联。例如：

```
Average Query Latency:          30ms (0.03s)
Number of concurrent queries:   1000/second

Number of threads needed:       0.03 x 1000 = 30 threads
Number of vCPUs needed:         30 / 2 = 15 vCPUs
```

将其与实例中的 vCPU 数量相关联，我们可以得到粗略的估计，`r5.4xlarge` 将是针对此工作负载尝试的推荐实例。这个估计是粗略的，仅用于为实例大小选择提供初步指导。任何应用程序都应通过调整大小操作来确定适合工作负载的适当实例数量和一种（或多种）实例类型。

还应考虑内存要求以及处理要求。当通过查询访问的数据在主内存缓冲池缓存中可用时，Neptune 的性能最高。预调配足够的内存还可以显著降低 I/O 成本。

有关在 Neptune 集群中调整实例大小的更多详细信息和指导，请访问调整 Neptune 数据库集群中数据库实例的大小页面。

从 Neo4j 到 Neptune 的数据迁移

在从 Neo4j 迁移到 Amazon Neptune 时，迁移数据是该过程中的一个主要步骤。有多种迁移数据的方法。正确的方法取决于应用程序的需求、数据大小和所需的迁移类型。但是，其中许多迁移都需要对相同的注意事项进行评测，下文将重点介绍其中一些注意事项。

> **Note**
>
> 有关如何执行离线数据迁移的一个示例的完整分步演练，请参阅 AWS 数据库博客中的使用全自动实用程序将 Neo4j 图形数据库迁移到 Neptune。

评测从 Neo4j 到 Neptune 的数据迁移

评测任何数据迁移的第一步是确定如何迁移数据。这些选项取决于要迁移的应用程序的架构、数据大小和迁移期间的可用性需求。通常，迁移往往分为两种类别之一：在线或离线。

离线迁移往往是最容易完成的，因为应用程序在迁移期间不接受读取或写入流量。应用程序停止接受流量后，可以在重新启用应用程序之前导出、优化、导入数据并测试应用程序。

在线迁移则更复杂，因为在迁移数据时，应用程序仍需要接受读取和写入流量。每次在线迁移的确切需求可能有所不同，但总体架构通常类似于以下架构：

- 需要通过将 Neo4j 流配置为 Kafka 集群的源，在 Neo4j 中启用对数据库进行持续更改的源。
- 完成此操作后，可以按照迁移到 Neptune 时从 Neo4j 导出数据中的说明导出正在运行的系统，并注明稍后与 Kafka 主题相关的时间。
- 然后，按照迁移到 Neptune 时从 Neo4j 导入数据中的说明将导出的数据导入到 Neptune 中。
- 然后，可以使用类似于从 Amazon Kinesis Data Streams 写入 Amazon Neptune 中描述的架构，将来自 Kafka 流的已更改数据复制到 Neptune 集群。请注意，更改复制可以并行运行，以验证新的应用程序架构和性能。
- 验证数据迁移后，可以将应用程序流量重定向到 Neptune 集群，并且可以停用 Neo4j 实例。

从 Neo4j 迁移到 Neptune 的数据模型优化

Neptune 和 Neo4j 都支持已标注的属性图 (LPG)。但是，Neptune 有一些架构和数据模型的差异，您可以利用这些差异来优化性能：

优化节点和边缘 ID

Neo4j 会自动生成数字长型 ID。使用 Cypher，您可以通过 ID 引用节点，但通常不鼓励这样做，而倾向于通过索引属性查找节点。

Neptune 允许您为顶点和边缘提供自己的基于字符串的 ID。如果您不提供自己的 ID，Neptune 会自动为新的边缘和顶点生成 UUID 的字符串表示形式。

如果您通过从 Neo4j 导出数据，然后批量导入到 Neptune，以便将数据从 Neo4j 迁移到 Neptune，则可以保留 Neo4j 的 ID。Neo4j 生成的数值在导入到 Neptune 时可以用作用户提供的 ID，在 Neptune 中，它们以字符串而不是数值表示。

但是，在某些情况下，您可能需要将顶点属性提升为顶点 ID。正如使用索引属性查找节点是在 Neo4j 中查找节点的最快方法一样，通过 ID 查找顶点也是在 Neptune 中找到顶点的最快方法。因此，如果您能识别出包含唯一值的合适顶点属性，您应该考虑在大容量加载 CSV 文件中用指定的属性值替换顶点 ~id。如果您这样做，您还必须在 CSV 文件中重写任何相应的 ~from 和 ~to 边缘值。

将数据从 Neo4j 迁移到 Neptune 时的架构约束

在 Neptune 中，唯一可用的架构约束是节点或边缘的 ID 的唯一性。鼓励需要利用唯一性约束的应用程序考虑使用这种方法，通过指定节点或边缘 ID 来实现唯一性约束。如果应用程序使用多列作为唯一性约束，则可以将 ID 设置为这些值的组合。例如，`id=123, code='SEA'` 可以表示为 `ID='123_SEA'`，以实现复杂的唯一性约束。

将数据从 Neo4j 迁移到 Neptune 时的边缘方向优化

当将节点、边缘或属性添加到 Neptune 时，将以三种不同的方式自动为它们编制索引，并具有可选的第四个索引。由于 Neptune 构建和使用索引的方式，遵循传出边缘的查询比使用传入边缘的查询效率更高。就 Neptune 的图形数据存储模型而言，这些是使用 SPOG 索引的基于主题的搜索。

如果在将数据模型和查询迁移到 Neptune 时，您发现最重要的查询依赖于遍历扇出程度很高的传入边缘，则可能需要考虑更改模型，以便这些遍历改为遵循传出边缘，尤其是在您无法指定要遍历哪些边缘标签时。为此，请反转相关边缘的方向并更新边缘标签，以反映此方向变化的语义。例如，您可能更改：

```
person_A — parent_of — person_B
   to:
person_B — child_of — person_A
```

要在批量加载边缘 CSV 文件中进行此更改，只需交换 `~from` 和 `~to` 列标题，然后更新 `~label` 列的值即可。

作为反转边缘方向的替代方案，您可以启用第四个 Neptune 索引，即 OSGP 索引，这样可以更高效地遍历传入的边缘或基于对象的搜索。但是，启用该第四个索引将降低插入速率并需要更多的存储空间。

将数据从 Neo4j 迁移到 Neptune 时进行筛选优化

Neptune 经过优化，当将属性筛选为最具选择性的可用属性时，效果最佳。当使用多个筛选条件时，会为每个筛选条件找到一组匹配的项目，然后计算出所有匹配集的重叠度。如果可能，将多个属性合并为一个属性，可以最大限度地减少索引查找次数并减少查询的延迟。

例如，此查询使用两个索引查找和一个联接：

```
MATCH (n) WHERE n.first_name='John' AND n.last_name='Doe' RETURN n
```

此查询使用单个索引查找来检索相同的信息：

```
MATCH (n) WHERE n.name='John Doe' RETURN n
```

Neptune 与 Neo4j 支持不同的数据类型。

Neo4j 数据类型映射到 Neptune 支持的数据类型

- 逻辑：`Boolean`

  在 Neptune 中将其映射到 `Bool` 或 `Boolean`。
- 数值：`Number`

  在 Neptune 中将其映射到以下 Neptune openCypher 类型中最窄的一个，这些类型可以支持相关数值属性的所有值：

  ```
  Byte
  Short
  Integer
  Long
  Float
  Double
  ```
- 文本：`String`

  在 Neptune 中将其映射到 `String`。

- 时间点：

```
Date
Time
LocalTime
DateTime
LocalDateTime
```

  使用 Neptune 支持的以下 ISO-8601 格式之一，在 Neptune 中将其映射到 `Date` (UTC)：

```
yyyy-MM-dd
yyyy-MM-ddTHH:mm
yyyy-MM-ddTHH:mm:ss
yyyy-MM-ddTHH:mm:ssZ
```

- 持续时间：`Duration`

  如有必要，在 Neptune 中将其映射到日期算术的数值。

- 空间：`Point`

  在 Neptune 中将其映射为组件数值，然后每个数值都成为一个单独的属性，或者表示为字符串值以由客户端应用程序解释。请注意，Neptune 的使用 OpenSearch 的全文搜索集成允许您对地理位置属性编制索引。

将多值属性从 Neo4j 迁移到 Neptune

Neo4j 允许将简单类型的同构列表存储为节点和边缘的属性。这些列表可以包含重复的值。

但是，Neptune 只允许对顶点属性使用集基数或单基数，而在属性图数据中允许对边缘属性使用单基数。因此，无法将包含重复值的 Neo4j 节点列表属性直接迁移到 Neptune 顶点属性，也无法将 Neo4j 关系列表属性直接迁移到 Neptune 边缘属性。

将具有重复值的 Neo4j 多值节点属性迁移到 Neptune 的一些可能策略如下：

- 丢弃重复值并将多值 Neo4j 节点属性转换为集基数 Neptune 顶点属性。请注意，Neptune 集可能无法反映原始 Neo4j 多值属性中项目的顺序。
- 将多值 Neo4j 节点属性转换为 Neptune 顶点字符串属性中 JSON 格式列表的字符串表示形式。
- 将每个多值属性值提取到具有值属性的单独顶点中，然后使用标有属性名称的边缘将这些顶点连接到父顶点。

同样，将 Neo4j 多值关系属性迁移到 Neptune 的可能策略如下：

- 将多值 Neo4j 关系属性转换为 JSON 格式列表的字符串表示形式，并将其存储为 Neptune 边缘字符串属性。
- 将 Neo4j 关系重构为连接到中间顶点的传入和传出 Neptune 边缘。使用标有属性名称的边缘，将每个多值关系属性值提取到具有值属性的单独顶点中，并将这些顶点提取到该中间顶点。

请注意，尽管 openCypher 包含一个允许在字符串值中进行简单搜索的 `CONTAINS` 谓词，但 JSON 格式列表的字符串表示形式对于 openCypher 查询语言来说是不透明的。

迁移到 Neptune 时从 Neo4j 导出数据

从 Neo4j 导出数据时，请使用 APOC 过程导出到 CSV 或 GraphML。尽管可以导出到其它格式，但有一些开源工具可以将从 Neo4j 导出的 CSV 数据转换为 Neptune 批量加载格式，还有一些开源工具可以将从 Neo4j 导出的 GraphML 数据转换为 Neptune 批量加载格式。

也可以使用各种 APOC 过程将数据直接导出到 Amazon S3。默认情况下，导出到 Amazon S3 桶处于禁用状态，但可以使用 Neo4j APOC 文档的导出到 Amazon S3 中重点介绍的过程将其启用。

迁移到 Neptune 时从 Neo4j 导入数据

您可以使用 Neptune 批量加载程序或使用支持的查询语言（例如 openCypher）中的应用程序逻辑，将数据导入 Neptune。

Neptune 批量加载程序是导入大量数据的首选方法，因为如果您遵循最佳实践，它可以提供优化的导入性能。批量加载程序支持两种不同的 CSV 格式，可以使用导出数据一节中提到的开源实用程序将从 Neo4j 导出的数据转换为这些格式。

还可以使用 openCypher 导入具有自定义逻辑的数据，用于解析、转换和导入。您可以通过 HTTPS 端点（推荐使用）或使用 bolt 驱动程序提交 openCypher 查询。

应用程序从 Neo4j 迁移到 Neptune

将数据从 Neo4j 迁移到 Neptune 之后，下一步是迁移应用程序本身。与数据一样，根据您使用的工具、要求、架构差异等，有多种方法可以迁移应用程序。下面概述了在此过程中通常需要考虑的事项。

从 Neo4j 移到 Neptune 时迁移连接

如果您目前不使用 Bolt 驱动程序，或者想使用其它驱动程序，则可以连接到 HTTPS 端点，该端点提供对返回数据的完全访问权限。

如果您的应用程序使用 Bolt 协议，则可以将这些连接迁移到 Neptune，并让应用程序使用与在 Neo4j 中使用的相同驱动程序进行连接。要连接到 Neptune，您可能需要对应用程序进行以下一项或多项更改：

- 需要更新 URL 和端口才能使用集群端点和集群端口（默认为 8182）。
- Neptune 要求所有连接都使用 SSL，因此您需要为每个连接指定对其进行加密。
- Neptune 通过分配 IAM policy 和角色来管理身份验证。IAM policy 和角色在应用程序中提供了极其灵活的用户管理级别，因此在配置集群之前，请务必阅读和理解 IAM 概述中的信息。
- 如Neptune 中的 Bolt 连接行为中所述，Neptune 中 Bolt 连接的行为与它在 Neo4j 中的行为在几个方面有所不同。
- 您可以在使用 openCypher 和 Bolt 的 Neptune 最佳实践中找到更多信息和建议。

使用 Bolt 协议向 Neptune 进行 openCypher 查询中提供了常用语言（例如 Java、Python、.NET 和 NodeJS）的代码示例，以及使用 IAM 身份验证等连接场景的代码示例。

从 Neo4j 移到 Neptune 时将查询路由到集群实例

Neo4j 客户端应用程序使用路由驱动程序并指定访问模式，将读取和写入请求路由到因果集群中的相应服务器。

将客户端应用程序迁移到 Neptune 时，请使用 Neptune 端点将查询高效地路由到集群中的相应实例：

- 所有与 Neptune 的连接都应在 URL 中使用 `bolt://`，而不是 `bolt+routing://` 或 `neo4j://`。
- 集群端点连接到集群中的当前主实例。使用集群端点将写入请求路由到主实例。
- 读取器端点在集群中的只读副本实例之间分配连接。如果您的单实例集群没有只读副本实例，则读取器端点将连接到主实例，而主实例支持写入操作。如果集群确实包含一个或多个只读副本实例，则向读取器端点发送写入请求会生成异常。

- 集群中的每个实例也可以有自己的实例端点。如果您的客户端应用程序需要向集群中的特定实例发送请求，请使用实例端点。

有关更多信息，请参阅Neptune 端点注意事项。

Neptune 中的数据一致性

使用 Neo4j 因果集群时，只读副本最终与核心服务器保持一致，但是客户端应用程序可以通过使用因果链来确保因果关系的一致性。因果链需要在事务之间传递书签，这允许客户端应用程序写入核心服务器，然后从只读副本中读取自己的写入内容。

在 Neptune 中，只读副本实例最终与写入器一致，副本滞后通常小于 100 毫秒。但是，在复制更改之前，对现有边缘和顶点的更新以及新边缘和顶点的添加在副本实例上不可见。因此，如果您的应用程序需要通过读取每次写入操作在 Neptune 上即时保持一致，请使用集群端点执行先写后读操作。这是唯一一次使用集群端点进行读取操作。在所有其它情况下，请使用读取器端点进行读取。

将查询从 Neo4j 迁移到 Neptune

尽管 Neptune 对 openCypher 的支持大大减少了从 Neo4j 迁移查询所需的工作量，但在迁移时仍有一些差异需要评测：

- 如上面的数据模型优化所述，可能需要对数据模型进行修改，以便为 Neptune 创建优化的图形数据模型，而这反过来又需要对查询进行更改并进行测试。
- Neo4j 提供了各种特定于 Cypher 的语言扩展，这些扩展未包含在 Neptune 实现的 openCypher 规范中。根据用例和使用的特征，openCypher 语言中可能有一些变通办法，或者使用 Gremlin 语言或通过重写 Cypher 查询以在 Neptune 上的 openCypher 中运行中描述的其它机制。
- 应用程序通常使用其它中间件组件来与数据库进行交互，而不是 Bolt 驱动程序本身。请检查Neptune 与 Neo4j 的兼容性以了解是否支持您正在使用的工具或中间件。
- 在失效转移的情况下，Bolt 驱动程序可能会继续连接到之前的写入器或读取器实例，因为提供给连接的集群端点已解析为 IP 地址。应用程序中的正确错误处理应该可以解决这个问题，如在失效转移后创建新连接中所述。
- 当由于无法解决的冲突或锁定等待超时而取消事务时，Neptune 将使用 `ConcurrentModificationException` 进行响应。有关更多信息，请参阅引擎错误代码。作为最佳实践，客户端应始终捕获并处理这些异常。

  当多个线程或多个应用程序同时写入系统时，偶尔会出现 `ConcurrentModificationException`。由于事务隔离级别，这些冲突有时可能是不可避免的。

- Neptune 支持对相同的数据同时运行 Gremlin 和 openCypher 查询。这意味着，在某些情况下，您可能需要考虑使用具有更强大查询功能的 Gremlin 来执行查询的某些功能。

如上面的预调配基础设施所述，每个应用程序都应通过调整大小操作来确保实例数量、实例大小和集群拓扑都针对应用程序的特定工作负载进行了优化。

此处讨论的迁移应用程序的注意事项是最常见的注意事项，但这并不是一个详尽的清单。每个应用程序都是唯一的。如果您还有其它问题，请联系 AWS Support 或联系您的客户团队。

迁移特定于 Neo4j 的特征和工具

Neo4j 具有各种自定义特征和附加组件，它们具有您的应用程序可能依赖的功能。在评估是否需要迁移此功能时，通常需要调查 AWS 中是否有更好的方法来实现相同的目标。考虑到 Neo4j 和 Neptune 之间的架构差异，您通常可以找到利用其它 AWS 服务或集成的有效替代方案。

有关特定于 Neo4j 的特征和建议的解决方法的列表，请参阅Neptune 与 Neo4j 的兼容性。

Neptune 与 Neo4j 的兼容性

Neo4j 采用一体化架构方法，其中数据加载、数据 ETL、应用程序查询、数据存储和管理操作都发生在同一组计算资源中，例如 EC2 实例。Amazon Neptune 是一个以 OLTP 为中心的开放规范图形数据库，其中，架构将操作分开并解耦资源，因此它们可以动态扩展。

Neo4j 中有各种特征和工具，包括第三方工具，它们不属于 openCypher 规范，与 openCypher 不兼容，或者与 Neptune 的 openCypher 实现不兼容。下面列出了其中一些最常见的特征和工具。

Neptune 中不存在的特定于 Neo4j 的特征

- **`LOAD CSV`** – Neptune 加载数据的架构方法与 Neo4j 不同。为了实现更好的扩展和成本优化，Neptune 对围绕资源的问题进行了分离，并建议使用其中一种 AWS 服务集成（例如 AWS Glue）执行所需的 ETL 流程，以 Neptune 批量加载程序支持的格式准备数据。

  另一个选项是使用在 AWS 计算资源（例如 Amazon EC2 实例、Lambda 函数、Amazon Elastic Container Service、AWS Batch 任务等）上运行的应用程序代码来做同样的事情。该代码可以使用 Neptune 的 HTTPS 端点或 Bolt 端点。
- 精细的访问控制 – Neptune 支持使用 IAM 条件键对数据访问操作进行精细的访问控制。可以在应用程序层实施额外的精细访问控制。
- Neo4j 结构 – Neptune 确实支持使用 SPARQL `SERVICE` 关键字跨数据库对 RDF 工作负载进行查询联合身份验证。由于目前没有针对属性图工作负载的查询联合身份验证的开放标准或规范，因此该功能需要在应用程序层实现。
- 基于角色的访问控制 (RBAC) – Neptune 通过分配 IAM policy 和角色来管理身份验证。IAM policy 和角色在应用程序中提供了极其灵活的用户管理级别，因此在配置集群之前，值得阅读和理解 IAM 概述中的信息。
- 加入书签 – Neptune 集群由单个写入器实例和最多 15 个只读副本实例组成。写入到写入器实例的数据符合 ACID 标准，并为后续读取提供了强有力的一致性保证。只读副本使用与写入器实例相同的存储卷并保持最终一致，通常在写入数据后不到 100 毫秒。如果您的用例迫切需要保证新写入的读取一致性，则应将这些读取定向到集群端点，而不是读取器端点。
- APOC 过程 – 由于 APOC 过程未包含在 openCypher 规范中，因此 Neptune 不为外部过程提供直接支持。相反，Neptune 依靠与其它 AWS 服务的集成，以可扩展、安全和稳健的方式实现类似的最终用户功能。有时 APOC 过程可以用 openCypher 或 Gremlin 重写，有些过程与 Neptune 应用程序无关。

  一般而言，APOC 过程分为以下几类：

- 导入 – Neptune 支持使用查询语言、Neptune 批量加载程序或作为 AWS Database Migration Service 的目标使用各种格式导入数据。可以使用 AWS Glue 和 `neptune-python-utils` 开源软件包对数据执行 ETL 操作。
- 导出 – Neptune 支持使用 `neptune-export` 实用程序导出数据，该实用程序支持多种常见的导出格式和方法。
- 数据库集成 – Neptune 支持使用 ETL 工具（如 AWS Glue）或迁移工具（如 AWS Database Migration Service）与其它数据库集成。
- 图形更新 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- 数据结构 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- 时态（日期时间） – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- 数学 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- 高级图形查询 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- 比较图形 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。
- Cypher 执行 – Neptune 通过支持 openCypher 和 Gremlin 查询语言，支持一系列用于更新属性图数据的丰富特征。有关重写常用过程的示例，请参阅Cypher 重写。

- 自定义过程 – Neptune 不支持用户创建的自定义过程。此功能必须在应用程序层实现。
- 地理空间 – 尽管 Neptune 不为地理空间特征提供原生支持，但类似的功能可以通过将 AWS 与其它服务集成来实现，如以下博客文章所示：Ross Gabay 和 Abhilash Vinod 撰写的结合使用 Amazon Neptune 和 Amazon OpenSearch Service 进行地理空间查询。
- 图形数据科学 - Neptune 目前通过 Neptune Analytics 支持图形分析，Neptune Analytics 是一款支持图形分析算法库的内存优化型引擎。

  此外，Neptune 还提供与 AWS Pandas SDK 的集成以及几个示例笔记本，这些笔记本展示了如何在 Python 环境中利用这种集成对图形数据进行分析。

- 架构约束 – 在 Neptune 中，唯一可用的架构约束是节点或边缘的 ID 的唯一性。没有特征可以为图形中的元素指定任何其它架构约束或任何其它唯一性或值约束。Neptune 中的 ID 值是字符串，可以使用 Gremlin 进行设置，如下所示：

```
g.addV('person').property(id, '1') )
```

鼓励需要利用 ID 作为唯一性约束的应用程序尝试使用这种方法来实现唯一性约束。如果应用程序使用多列作为唯一性约束，则可以将 ID 设置为这些值的组合。例如，`id=123, code='SEA'` 可以表示为 `ID='123_SEA'`，以实现复杂的唯一性约束。

- 多租赁 – Neptune 仅支持每个集群单个图形。要使用 Neptune 构建多租户系统，要么使用多个集群，要么在单个图形中对租户进行逻辑分区，然后使用应用程序端逻辑强制分离。例如，添加属性 `tenantId` 并将其包含在每个查询中，如下所示：

```
MATCH p=(n {tenantId:1})-[]->({tenantId:1}) RETURN p LIMIT 5)
```

Neptune 无服务器使得使用多个数据库集群实现多租赁变得相对容易，每个数据库集群都可以根据需要独立自动扩展。

Neptune 对 Neo4j 工具的支持

Neptune 为 Neo4j 工具提供了以下替代方案：

- Neo4j 浏览器 – Neptune 提供开源图形笔记本，提供以开发人员为中心的 IDE，用于运行查询和可视化结果。
- Neo4j Bloom – Neptune 支持使用第三方可视化解决方案进行丰富的图形可视化，例如 Graph-explorer、Tom Sawyer、Cambridge Intelligence、Graphistry、metaphacts 和 G.V()。
- GraphQL – Neptune 目前通过自定义 AWS AppSync 集成支持 GraphQL。请参阅使用 Amazon Neptune 和 AWS Amplify 构建图形应用程序博客文章，以及示例项目使用 AWS AppSync 和 Amazon Neptune 构建无服务器卡路里跟踪器应用程序。
- NeoSemantics – Neptune 原生支持 RDF 数据模型，因此建议希望运行 RDF 工作负载的客户使用 Neptune 的 RDF 模型支持。
- Arrows.app – 使用导出命令导出模型时创建的密码与 Neptune 兼容。
- Linkurious Ogma – 此处提供了与 Linkurious Ogma 的集成示例。
- Spring Data Neo4j – 这目前与 Neptune 不兼容。
- Neo4j Spark 连接器 – Neo4j Spark 连接器可以在 Spark 任务中使用，以使用 openCypher 连接到 Neptune。以下是一些示例代码和应用程序配置：

  示例代码：

```
SparkSession spark = SparkSession
            .builder()
            .config("encryption.enabled", "true")
            .appName("Simple Application").config("spark.master",
 "local").getOrCreate();

Dataset<Row> df = spark.read().format("org.neo4j.spark.DataSource")
            .option("url", "bolt://(your cluster endpoint):8182")
            .option("encryption.enabled", "true")
            .option("query", "MATCH (n:airport) RETURN n")
            .load();

System.out.println("TOTAL RECORD COUNT: " + df.count());
spark.stop();
```

应用程序配置：

```
<dependency>
    <groupId>org.neo4j</groupId>
    <artifactId>neo4j-connector-apache-spark_2.12-4.1.0</artifactId>
    <version>4.0.1_for_spark_3</version>
</dependency>
```

此处未列出的 Neo4j 特征和工具

如果您使用的工具或特征未在此处列出，则我们不确定它是否与 Neptune 或 AWS 中的其它服务兼容。如果您还有其它问题，请联系 AWS Support 或联系您的客户团队。

重写 Cypher 查询以在 Neptune 上的 openCypher 中运行

openCypher 语言是一种用于属性图的声明式查询语言，最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。在 AWS，我们相信开源对每个人都有好处，我们致力于将开源的价值带给我们的客户，为开源社区带来 AWS 的运营卓越性。

openCypher 语法在 Cypher 查询语言参考版本 9 中介绍。

由于 openCypher 包含 Cypher 查询语言的语法和特征的子集，因此某些迁移方案需要以兼容 openCypher 的形式重写查询，或者研究其它方法来实现所需的特征。

本节包含处理常见差异的建议，但绝非详尽无遗。您应该使用这些重写对任何应用程序进行全面测试，以确保结果符合您的预期。

重写 `None`、`All` 和 `Any` 谓词函数

这些函数不是 openCypher 规范的一部分。使用列表推导可以在 openCypher 中获得类似的结果。

例如，找到从节点 `Start` 到节点 `End` 的所有路径，但不允许任何旅程通过一个类属性为 `D` 的节点：

```
# Neo4J Cypher code
match p=(a:Start)-[:HOP*1..]->(z:End)
where none(node IN nodes(p) where node.class ='D')
return p

# Neptune openCypher code
match p=(a:Start)-[:HOP*1..]->(z:End)
where size([node IN nodes(p) where node.class = 'D']) = 0
return p
```

列表推导可以实现以下结果：

```
all  => size(list_comprehension(list)) = size(list)
any  => size(list_comprehension(list)) >= 1
none => size(list_comprehension(list)) = 0
```

在 openCypher 中重写 Cypher `reduce()` 函数

`reduce()` 函数不是 openCypher 规范的一部分。它通常用于从列表中的元素创建数据的聚合。在许多情况下，您可以结合使用列表推导和 `UNWIND` 子句来获得相似的结果。

例如，以下 Cypher 查询查找安克雷奇 (ANC) 和奥斯汀 (AUS) 之间有一到三个停靠点的路径上的所有机场，并返回每条路径的总距离：

```
MATCH p=(a:airport {code: 'ANC'})-[r:route*1..3]->(z:airport {code: 'AUS'})
RETURN p, reduce(totalDist=0, r in relationships(p) | totalDist + r.dist) AS totalDist
ORDER BY totalDist LIMIT 5
```

您可以在 openCypher 中为 Neptune 编写相同的查询，如下所示：

```
MATCH p=(a:airport {code: 'ANC'})-[r:route*1..3]->(z:airport {code: 'AUS'})
UNWIND [i in relationships(p) | i.dist] AS di
RETURN p, sum(di) AS totalDist
ORDER BY totalDist
LIMIT 5
```

在 openCypher 中重写 Cypher FOREACH 子句

FOREACH 子句不是 openCypher 规范的一部分。它通常用于在查询过程中更新数据，通常来自路径中的聚合或元素。

作为路径示例，找到安克雷奇 (ANC) 和奥斯汀 (AUS) 之间不超过两站的路径上的所有机场，并在每个机场上设置一个已访问的属性：

```
# Neo4J Example
MATCH p=(:airport {code: 'ANC'})-[*1..2]->({code: 'AUS'})
FOREACH (n IN nodes(p) | SET n.visited = true)

# Neptune openCypher
MATCH p=(:airport {code: 'ANC'})-[*1..2]->({code: 'AUS'})
WITH nodes(p) as airports
UNWIND airports as a
SET a.visited=true
```

另一个示例是：

```
# Neo4J Example
MATCH p=(start)-[*]->(finish)
WHERE start.name = 'A' AND finish.name = 'D'
FOREACH (n IN nodes(p) | SET n.marked = true)

# Neptune openCypher
```

```
MATCH p=(start)-[*]->(finish)
WHERE start.name = 'A' AND finish.name = 'D'
UNWIND nodes(p) AS n
SET n.marked = true
```

在 Neptune 中重写 Neo4j APOC 过程

以下示例使用 openCypher 来取代一些最常用的 APOC 过程。这些示例仅供参考，旨在提供一些有关如何处理常见场景的建议。实际上，每个应用程序都是不同的，您必须制定自己的策略来提供所需的所有功能。

重写 `apoc.export` 过程

Neptune 使用 neptune-export 实用程序为完整图形和基于查询的导出提供了一系列选项，且采用 CSV 和 JSON 等各种输出格式（请参阅从 Neptune 数据库集群中导出数据）。

重写 `apoc.schema` 过程

Neptune 没有显式定义的架构、索引或约束，因此不再需要许多 `apoc.schema` 过程。示例为：

- `apoc.schema.assert`
- `apoc.schema.node.constraintExists`
- `apoc.schema.node.indexExists`,
- `apoc.schema.relationship.constraintExists`
- `apoc.schema.relationship.indexExists`
- `apoc.schema.nodes`
- `apoc.schema.relationships`

Neptune openCypher 确实支持检索与这些过程检索的值相似的值，如下所示，但在较大的图形上可能会遇到性能问题，因为这样做需要扫描图形的很大一部分才能返回答案。

```
# openCypher replacement for apoc.schema.properties.distinct
MATCH (n:airport)
RETURN DISTINCT n.runways
```

```
# openCypher replacement for apoc.schema.properties.distinctCount
MATCH (n:airport)
RETURN DISTINCT n.runways, count(n.runways)
```

apoc.do 过程的替代方案

这些过程用于提供条件查询执行，这使用其它 openCypher 子句即可轻松实现。在 Neptune 中，至少有两种方法可以实现类似的行为：

- 一种方法是将 openCypher 的列表推导功能与子句 UNWIND 结合起来。
- 另一种方法是使用 Gremlin 中的 choose() 和 coalesce() 步骤。

这些方法的示例如下所示。

apoc.do.when 的替代方案

```
# Neo4J Example
MATCH (n:airport {region: 'US-AK'})
CALL apoc.do.when(
 n.runways>=3,
 'SET n.is_large_airport=true RETURN n',
 'SET n.is_large_airport=false RETURN n',
 {n:n}
) YIELD value
WITH collect(value.n) as airports
RETURN size([a in airports where a.is_large_airport]) as large_airport_count,
size([a in airports where NOT a.is_large_airport]) as small_airport_count


# Neptune openCypher
MATCH (n:airport {region: 'US-AK'})
WITH n.region as region, collect(n) as airports
WITH [a IN airports where a.runways >= 3] as large_airports,
[a IN airports where a.runways < 3] as small_airports, airports
UNWIND large_airports as la
SET la.is_large_airport=true
WITH DISTINCT small_airports, airports
UNWIND small_airports as la
    SET la.small_airports=true
WITH DISTINCT airports
RETURN size([a in airports where a.is_large_airport]) as large_airport_count,
size([a in airports where NOT a.is_large_airport]) as small_airport_count

#Neptune Gremlin using choose()
g.V().
  has('airport', 'region', 'US-AK').
```

```
  choose(
    values('runways').is(lt(3)),
    property(single, 'is_large_airport', false),
    property(single, 'is_large_airport', true)).
  fold().
  project('large_airport_count', 'small_airport_count').
    by(unfold().has('is_large_airport', true).count()).
    by(unfold().has('is_large_airport', false).count())

 #Neptune Gremlin using coalesce()
g.V().
  has('airport', 'region', 'US-AK').
  coalesce(
    where(values('runways').is(lt(3))).
    property(single, 'is_large_airport', false),
    property(single, 'is_large_airport', true)).
  fold().
  project('large_airport_count', 'small_airport_count').
    by(unfold().has('is_large_airport', true).count()).
    by(unfold().has('is_large_airport', false).count())
```

apoc.do.case 的替代方案

```
# Neo4J Example
MATCH (n:airport {region: 'US-AK'})
CALL apoc.case([
 n.runways=1, 'RETURN "Has one runway" as b',
 n.runways=2, 'RETURN "Has two runways" as b'
 ],
 'RETURN "Has more than 2 runways" as b'
) YIELD value
RETURN {type: value.b,airport: n}

# Neptune openCypher
MATCH (n:airport {region: 'US-AK'})
WITH n.region as region, collect(n) as airports
WITH [a IN airports where a.runways =1] as single_runway,
[a IN airports where a.runways =2] as double_runway,
[a IN airports where a.runways >2] as many_runway
UNWIND single_runway as sr
    WITH {type: "Has one runway",airport: sr} as res, double_runway, many_runway
WITH DISTINCT double_runway as double_runway, collect(res) as res, many_runway
UNWIND double_runway as dr
```

```
    WITH {type: "Has two runways",airport: dr} as two_runways, res, many_runway
WITH collect(two_runways)+res as res, many_runway
UNWIND many_runway as mr
    WITH {type: "Has more than 2 runways",airport: mr} as res2, res, many_runway
WITH collect(res2)+res as res
UNWIND res as r
RETURN r

#Neptune Gremlin using choose()
g.V().
  has('airport', 'region', 'US-AK').
  project('type', 'airport').
    by(
      choose(values('runways')).
        option(1, constant("Has one runway")).
        option(2, constant("Has two runways")).
        option(none, constant("Has more than 2 runways"))).
    by(elementMap())

 #Neptune Gremlin using coalesce()
 g.V().
  has('airport', 'region', 'US-AK').
  project('type', 'airport').
    by(
      coalesce(
        has('runways', 1).constant("Has one runway"),
        has('runways', 2).constant("Has two runways"),
        constant("Has more than 2 runways"))).
    by(elementMap())
```

基于列表的属性的替代方案

Neptune 目前不支持存储基于列表的属性。但是，通过将列表值存储为逗号分隔的字符串，然后使用 `join()` 和 `split()` 函数来构造和解构列表属性，也可以获得类似的结果。

例如，如果我们想将标签列表另存为属性，我们可以使用示例重写，它演示如何检索逗号分隔的属性，然后使用 `split()` 和 `join()` 函数以及列表推导来获得同等的结果：

```
# Neo4j Example (In this example, tags is a durable list of string.
MATCH (person:person {name: "TeeMan"})
WITH person, [tag in person.tags WHERE NOT (tag IN ['test1', 'test2', 'test3'])] AS
 newTags
SET person.tags = newTags
```

```
RETURN person

# Neptune openCypher
MATCH (person:person {name: "TeeMan"})
WITH person, [tag in split(person.tags, ',') WHERE NOT (tag IN ['test1', 'test2',
 'test3'])] AS newTags
SET person.tags = join(newTags,',')
RETURN person
```

用于从 Neo4j 迁移到 Neptune 的资源

Neptune 提供了多种工具和资源，可以为迁移过程提供帮助。

帮助从 Neo4j 迁移到 Neptune 的工具

- openCypher CheatSheet。
- neo4j-to-neptune – 一款用于将数据从 Neo4j 迁移到 Neptune 的命令行实用程序。
- fully-automated-neo4j-to-neptune – 一款 AWS CDK 应用程序，向您展示如何将简单的 Neo4j 数据库迁移到 Amazon Neptune。
- csv-to-neptune-bulk-format – 此工具采用基于配置的方法，将一个或多个 CSV 文件重新格式化为受支持的 Neptune 批量加载格式。

博客文章

- 使用 Amazon Managed Streaming for Apache Kafka 将数据捕获从 Neo4j 更改为 Amazon Neptune，作者 Sanjeet Sahay（2020 年 6 月 22 日）
- 通过全自动实用工具将 Neo4j 图形数据库迁移到 Amazon Neptune，作者 Sanjeet Sahay（2020 年 4 月 13 日）

将现有图形从 Apache TinkerPop Gremlin 服务器迁移到 Amazon Neptune

如果您想迁移到 Amazon Neptune 的 Apache TinkerPop Gremlin 服务器中有图形数据，则需要采取以下步骤：

1. 将数据从 Gremlin 服务器导出到 Amazon Simple Storage Service (Amazon S3)。
2. 将导出的数据转换为 Neptune 批量加载程序可以导入的 CSV 格式。
3. 使用Neptune 批量加载程序，将数据导入到您准备好的 Neptune 数据库集群中。
4. 修改现有应用程序以连接到 Neptune 的 Gremlin 端点，并进行任何必要的更改以符合 Neptune Gremlin 实现差异。

将现有图形从 RDF 三重存储迁移到 Amazon Neptune

如果您在 RDF/SPARQL 中有图形数据要迁移到 Amazon Neptune，则需要执行以下步骤：

1. 从 RDF 三重存储中导出数据。
2. 将导出的数据转换为 Neptune 批量加载程序可以导入的格式。
3. 将要导入的数据存储在 Amazon Simple Storage Service (Amazon S3) 中。
4. 使用Neptune 批量加载程序，将数据从 Amazon S3 导入到您已准备好的 Neptune 数据库集群中。
5. 修改现有应用程序以连接到 Neptune 的 SPARQL 端点。

如果您想尝试将属性图 CSV 数据迁移到 RDF 中，您可以使用 Amazon Neptune CSV 到 RDF 转换器。

使用 AWS Database Migration Service（AWS DMS）从关系数据库或 NoSQL 数据库迁移到 Amazon Neptune

AWS Database Migration Service (AWS DMS) 是一项云服务，可轻松迁移关系数据库、数据仓库、NoSQL 数据库及其他类型的数据存储。如果您将图形数据存储在 AWS DMS 支持的关系数据库或 NoSQL 数据库之一中，则 AWS DMS 可以帮助您快速安全地迁移到 Neptune，而不要求当前数据库停机。有关详细信息，请参阅AWS Database Migration Service 用于将来自其他数据存储的数据加载到 Amazon Neptune。

使用 AWS DMS 的迁移数据流如下所示：

- 创建 AWS DMS 表映射对象。这个 JSON 对象指定应该从源数据库中读取哪些表，以什么顺序读取，以及如何命名它们的列。它还可以筛选正在复制的行，并提供简单的值转换，例如转换为小写或舍入。
- 创建一个 Neptune `GraphMappingConfig`，以指定如何将从源数据库中提取的数据加载到 Neptune。
  - 对于 RDF 数据（使用 SPARQL 查询），`GraphMappingConfig` 使用 W3 的标准 R2RML 映射语言编写。
  - 对于属性图数据（使用 Gremlin 进行查询），`GraphMappingConfig` 是 JSON 对象，如GraphMappingConfig Property-Graph/Gremlin 数据的布局中所述。
- 在与 Neptune 数据库集群相同的 VPC 中创建 AWS DMS 复制实例以执行迁移。
- 创建 Amazon S3 桶，用作暂存正在迁移的数据的中间存储。
- 运行 AWS DMS 迁移任务。

有关详细信息，请参阅AWS Database Migration Service 用于将来自其他数据存储的数据加载到 Amazon Neptune，以及 Chris Smith 撰写的由四个部分组成的博客文章“在 Amazon Neptune 中使用 AWS Database Migration Service (DMS) 从关系数据库填充图形”：

- 第 1 部分：设置阶段
- 第 2 部分：设计属性图模型
- 第 3 部分：设计 RDF 模型
- 第 4 部分：将所有内容组合在一起

从 Blazegraph 迁移到 Amazon Neptune

如果您在开源 Blazegraph RDF 三重存储中有图形，则可以通过以下步骤将图形数据迁移到 Amazon Neptune：

- 预调配 AWS 基础设施。首先，使用 AWS CloudFormation 模板预调配所需的 Neptune 基础设施（请参阅创建数据库集群）。
- 从 Blazegraph 导出数据。从 Blazegraph 导出数据有两种主要方法，即使用 SPARQL CONSTRUCT 查询或使用 Blazegraph Export 实用程序。
- 将数据导入 Neptune。然后，您可以使用 Neptune Workbench 和Neptune 批量加载程序将导出的数据文件加载到 Neptune 中。

这种方法通常也适用于从其它 RDF 三重存储数据库迁移。

Blazegraph 与 Neptune 的兼容性

在将图形数据迁移到 Neptune 之前，您应该注意 Blazegraph 和 Neptune 之间有几个显著的差异。这些差异可能需要更改查询和/或应用程序架构，甚至使迁移变得不切实际：

- **`Full-text search`** – 在 Blazegraph 中，您可以通过与 Apache Solr 集成，使用内部全文搜索或外部全文搜索功能。如果您使用其中任何一项特征，请随时了解 Neptune 支持的全文搜索特征的最新更新。请参阅Neptune 全文搜索。
- **`Query hints`** – Blazegraph 和 Neptune 都使用查询提示的概念扩展 SPARQL。在迁移过程中，您需要迁移所使用的任何查询提示。有关 Neptune 支持的最新查询提示的信息，请参阅SPARQL 查询提示。
- 推理 – Blazegraph 在三元组模式下支持推理作为可配置选项，但在四元组模式下不支持。Neptune 尚不支持推理。
- 地理空间搜索 – Blazegraph 支持配置启用地理空间的命名空间。此特征在 Neptune 中尚不可用。
- 多租赁 – Blazegraph 支持在单个数据库中使用多租赁。在 Neptune 中，通过将数据存储在命名图形中并使用 USING NAMED 子句进行 SPARQL 查询，或者为每个租户创建单独的数据库集群，来支持多租赁。
- 联合身份验证 – Neptune 目前支持与 Neptune 实例可访问的位置进行 SPARQL 1.1 联合身份验证，例如私有 VPC 内、VPC 之间或外部互联网端点。根据特定的设置和所需的联合身份验证端点，您可能需要一些额外的网络配置。
- Blazegraph 标准扩展 – Blazegraph 包括对 SPARQL 和 REST API 标准的多个扩展，而 Neptune 仅与标准规范本身兼容。这可能需要更改您的应用程序，否则会使迁移变得困难。

为 Neptune 预调配 AWS 基础设施

尽管您可以通过 AWS Management Console 或 AWS CLI 手动构建所需的 AWS 基础设施，但改用 CloudFormation 模板通常更方便，如下所述：

使用 CloudFormation 模板预调配 Neptune：

1. 导航到 使用 AWS CloudFormation 堆栈创建 Neptune 数据库集群。
2. 在您的首选区域中选择启动堆栈。
3. 设置所需的参数（堆栈名称和 `EC2SSHKeyPairName`）。还要设置以下可选参数以简化迁移过程：
   - 将 `AttachBulkloadIAMRoleToNeptuneCluster` 设置为 true。此参数允许创建相应的 IAM 角色并将其附加到您的集群，以允许批量加载数据。
   - 将 `NotebookInstanceType` 设置为您的首选实例类型。此参数创建一个 Neptune 工作簿，用于批量加载到 Neptune 中并验证迁移。
4. 选择 Next（下一步）。
5. 设置您想要的任何其它堆栈选项。
6. 选择 Next（下一步）。
7. 查看您的选项并选中两个复选框，以确认 AWS CloudFormation 可能需要其它功能。
8. 选择创建堆栈。

堆栈创建过程可能耗时数分钟。

从 Blazegraph 中导出数据

下一步是以与 Neptune 批量加载程序兼容的格式从 Blazegraph 中导出数据。

根据数据在 Blazegraph 中的存储方式（三元组或四元组）以及正在使用的命名图形的数量，Blazegraph 可能需要您多次执行导出过程并生成多个数据文件：

- 如果数据存储为三元组，则需要为每个命名的图形运行一次导出。
- 如果数据存储为四元组，则可以选择以 N-Quads 格式导出数据，也可以选择以三元组格式导出每个命名图形。

下面我们假设您将单个命名空间导出为 N-Quads，但您可以重复该过程以获取其它命名空间或所需的导出格式。

如果您需要 Blazegraph 在迁移期间保持在线状态并可用，请使用 SPARQL CONSTRUCT 查询。这要求您安装、配置和运行带有可访问的 SPARQL 端点的 Blazegraph 实例。

如果您不需要 Blazegraph 在线，请使用 BlazeGraph 导出实用程序。为此，您必须下载 Blazegraph，并且需要可以访问数据文件和配置文件，但服务器不需要运行。

使用 SPARQL CONSTRUCT 从 Blazegraph 导出数据

SPARQL CONSTRUCT 是 SPARQL 的一项特征，它返回与指定查询模板匹配的 RDF 图形。在此用例中，您可以使用如下查询逐个命名空间导出数据：

```
CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query
 hint:constructDistinctSPO "false" . ?s ?p ?o }
```

尽管还有其它 RDF 工具可以导出这些数据，但运行此查询的最简单方法是使用 Blazegraph 提供的 REST API 端点。以下脚本演示了如何使用 Python (3.6+) 脚本将数据导出为 N-Quads：

```
import requests

# Configure the URL here: e.g. http://localhost:9999/sparql
url = "http://localhost:9999/sparql"
payload = {'query': 'CONSTRUCT WHERE { hint:Query hint:analytic "true" . hint:Query
 hint:constructDistinctSPO "false" . ?s ?p ?o }'}
# Set the export format to be n-quads
headers = {
'Accept': 'text/x-nquads'
}
# Run the http request
response = requests.request("POST", url, headers=headers, data = payload, files = [])
#open the file in write mode, write the results, and close the file handler
f = open("export.nq", "w")
f.write(response.text)
f.close()
```

如果数据存储为三元组，则需要更改 `Accept` 标头参数，以便使用在 Blazegraph GitHub 存储库上指定的值以适当的格式（N-Triples、RDF/XML 或 Turtle）导出数据。

使用 Blazegraph 导出实用程序导出数据

Blazegraph 包含一个用于导出数据的实用程序方法，即 `ExportKB` 类。`ExportKB` 便于从 Blazegraph 导出数据，但与之前的方法不同，它要求服务器在导出运行时处于离线状态。当您可以在迁移期间使 Blazegraph 离线时，或者迁移可以通过数据备份进行时，这成为理想的方法。

您可以在安装了 Blazegraph 但未运行的计算机上通过 Java 命令行运行该实用程序。运行此命令的最简单方法是下载位于 GitHub 上的最新 blazegraph.jar 版本。运行此命令需要几个参数：

- **`log4j.primary.configuration`** - log4j 属性文件的位置。
- **`log4j.configuration`** - log4j 属性文件的位置。
- **`output`** – 导出数据的输出目录。文件以 `tar.gz` 的形式位于命名的子目录中，如知识库中所述。
- **`format`** – 所需的输出格式，后跟 `RWStore.properties` 文件的位置。如果您使用的是三元组，则需要将 `-format` 参数更改为 `N-Triples`、`Turtle` 或 `RDF/XML`。

例如，如果您有 Blazegraph 日志文件和属性文件，请使用以下代码将数据导出为 N-Quads：

```
java -cp blazegraph.jar \
    com.bigdata.rdf.sail.ExportKB \
    -outdir ~/temp/ \
    -format N-Quads \
    ./RWStore.properties
```

如果导出成功，您将看到如下输出：

```
Exporting kb as N-Quads on /home/ec2-user/temp/kb
Effective output directory: /home/ec2-user/temp/kb
Writing /home/ec2-user/temp/kb/kb.properties
Writing /home/ec2-user/temp/kb/data.nq.gz
Done
```

创建 Amazon Simple Storage Service (Amazon S3) 桶，并将导出的数据复制到该桶

从 Blazegraph 导出数据后，在与目标 Neptune 数据库集群相同的区域中创建一个 Amazon Simple Storage Service (Amazon S3) 桶，供 Neptune 批量加载程序用于从中导入数据。

有关如何创建 Amazon S3 桶的说明，请参阅 Amazon Simple Storage Service 用户指南中的如何创建 S3 桶，以及 Amazon Simple Storage Service 用户指南中的创建存储桶的示例。

有关如何将导出的数据文件复制到新的 Amazon S3 桶的说明，请参阅 Amazon Simple Storage Service 用户指南中的将对象上传到桶，或将高级 (s3) 命令与 AWS CLI 一起使用。也可以使用如下所示的 Python 代码逐个复制文件：

```
import boto3
```

```
region = 'region name'
bucket_name = 'bucket name'
s3 = boto3.resource('s3')
s3.meta.client.upload_file('export.nq', bucket_name, 'export.nq')
```

使用 Neptune 批量加载程序将数据导入 Neptune

从 Blazegraph 导出数据并将其复制到 Amazon S3 桶后，就可以将数据导入 Neptune 了。Neptune 有一个批量加载程序，与使用 SPARQL 执行加载操作相比，它加载数据的速度更快，开销更少。批量加载程序进程通过调用加载程序端点 API 启动，将存储在已识别的 S3 桶中的数据加载到 Neptune 中。

尽管您可以通过直接调用加载程序 REST 端点来实现此目的，但您必须有权访问在其中运行目标 Neptune 实例的私有 VPC。您可以设置堡垒主机，通过 SSH 连接到该计算机，然后运行 cURL 命令，但是使用 Neptune Workbench 会更容易。

Neptune Workbench 是一款预配置的 Jupyter 笔记本，作为 Amazon SageMaker 笔记本运行，安装了若干 Neptune 专用的笔记本魔术命令。这些魔术命令简化了常见的 Neptune 操作，例如检查集群状态、运行 SPARQL 和 Gremlin 遍历以及运行批量加载操作。

要启动批量加载过程，请使用 `%load` 魔术命令，它提供了一个运行 Neptune 加载程序命令 的接口：

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 选择 aws-neptune-blazegraph-to-neptune。
3. 选择打开笔记本。
4. 在 Jupyter 的运行实例中，要么选择现有笔记本，要么使用 Python 3 内核创建一个新笔记本。
5. 在笔记本中，打开一个单元格，输入 `%load` 并运行该单元格。
6. 为批量加载程序设置参数：
   a. 在源中，输入要导入的源文件的位置：`s3://{bucket_name}/{file_name}`。
   b. 对于格式，选择相应的格式，在本示例中为 `nquads`。
   c. 对于加载 ARN，输入 `IAMBulkLoad` 角色的 ARN（此信息位于 IAM 控制台的角色下）。
7. 选择提交。

结果包含请求的状态。批量加载通常是长时间运行的进程，因此响应并不意味着加载已完成，而只是表示加载已经开始。此状态信息会定期更新，直到它报告任务已完成。

> **Note**
>
> 这些信息也可以在博客文章迁移到云端：将 Blazegraph 迁移到 Amazon Neptune 中找到。

将数据加载到 Amazon Neptune 中

有几种不同的方法可以将图形数据加载到 Amazon Neptune：

- 如果只需要加载相对少量的数据，则可以使用 SPARQL `INSERT` 语句或 Gremlin `addV` 和 `addE` 步骤等查询。
- 您可以利用 Neptune 批量加载程序 提取驻留在外部文件中的大量数据。相比查询语言命令，批量加载器命令速度更快，开销更少。它针对大型数据集进行了优化，并且支持 RDF（资源描述框架）数据和 Gremlin 数据。
- 您可以使用 AWS Database Migration Service (AWS DMS) 从其他数据存储中导入数据（参见AWS Database Migration Service 用于将来自其他数据存储的数据加载到 Amazon Neptune和AWS Database Migration Service 用户指南）。
- 最后，您可以使用 Gremlin 的 `g.io(URL).read()` 步骤来读取 GraphML（一种 XML 格式）、GraphSON（JSON 格式）和其它格式的数据文件。有关详细信息，请参阅TinkerPop文档。

主题

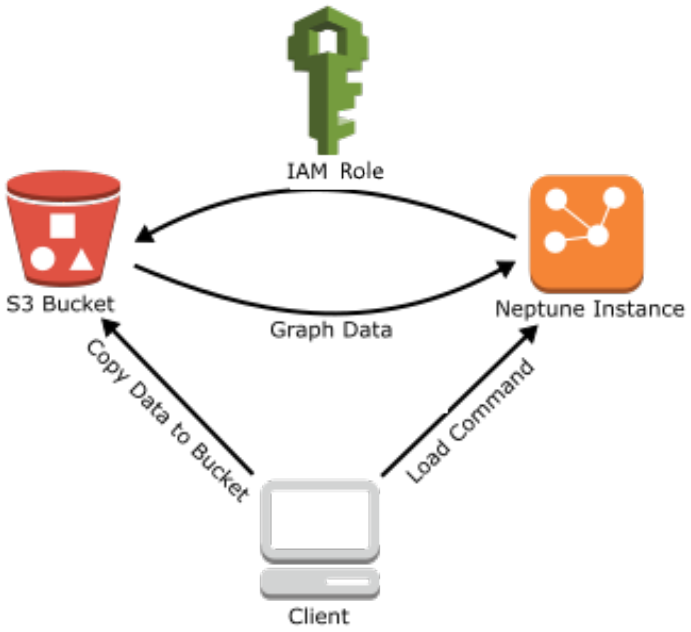
- 使用 Amazon Neptune 批量加载程序收集数据
- AWS Database Migration Service 用于将来自其他数据存储的数据加载到 Amazon Neptune

使用 Amazon Neptune 批量加载程序收集数据

Amazon Neptune 提供将数据从外部文件直接加载到 Neptune 数据库集群中的 `Loader` 命令。可使用此命令代替执行大量 `INSERT` 语句、`addV` 和 `addE` 步骤或其它 API 调用。

Neptune 加载程序命令更快、开销更低，已针对大型数据集进行优化，并且支持 Gremlin 数据和 SPARQL 使用的 RDF（资源描述框架）数据。

以下关系图概述了加载过程：



下面是加载过程的步骤：

1. 将数据文件复制到 Amazon Simple Storage Service (Amazon S3) 桶。
2. 创建对此存储桶具有读取和列出访问权限的 IAM 角色。
3. 创建 Amazon S3 VPC 端点。
4. 通过经由 HTTP 将请求发送到 Neptune 数据库实例来启动 Neptune 加载程序。
5. Neptune 数据库实例代入 IAM 角色以从桶加载数据。

> **Note**
>
> 如果使用 Amazon S3 SSE-S3 或 SSE-KMS 模式加密了数据，则可以从 Amazon S3 加载加密数据，前提是用于批量加载的角色可以访问 Amazon S3 对象，在 SSE-KMS 的情况下，也可以访问 `kms:decrypt`。然后，Neptune 可以模拟您的凭证并代表您发出 `s3:getObject` 调用。
>
> 但是，Neptune 当前不支持加载使用 SSE-C 模式加密的数据。

以下各节提供有关准备数据并将数据加载到 Neptune 中的说明。

主题

- 先决条件：IAM 角色和 Amazon S3 访问权限
- 加载数据格式
- 示例：将数据加载到 Neptune 数据库实例中
- 优化 Amazon Neptune 批量加载
- Neptune 加载程序参考

先决条件：IAM 角色和 Amazon S3 访问权限

从亚马逊简单存储服务 (Amazon S3) 存储桶加载数据需要 AWS Identity and Access Management 一个有权访问该存储桶的 (IAM) 角色。Amazon Neptune 代入此角色以加载数据。

> **Note**
>
> 在使用 Amazon S3 SSE-S3 模式加密数据的情况下，您可从 Amazon S3 加载加密数据。在这种情况下，Neptune 能够模拟您的凭证并代表您发出 `s3:getObject` 调用。
> 只要您的 IAM 角色包括访问 AWS KMS的必要权限，您还可以从 Amazon S3 加载使用 SSE-KMS 模式加密的加密数据。如果没有适当的 AWS KMS 权限，批量加载操作将失败并返回`LOAD_FAILED`响应。
> Neptune 目前不支持加载使用 SSE-C 模式加密的 Amazon S3 数据。

以下各部分说明了如何使用托管式 IAM policy 创建用于访问 Amazon S3 资源的 IAM 角色，然后将此角色附加到您的 Neptune 集群。

主题

- 创建 IAM 角色以允许 Amazon Neptune 访问 Amazon S3 资源
- 将 IAM 角色添加到 Amazon Neptune 集群
- 创建 Amazon S3 VPC 端点
- 在 Amazon Neptune 中串联 IAM 角色

> **Note**
>
> 这些说明需要具有对 IAM 控制台的访问权限以及管理 IAM 角色和策略的权限。有关更多信息，请参阅 IAM 用户指南中的在 AWS 管理控制台中工作的权限。
> Amazon Neptune 控制台需要用户具有以下 IAM 权限，以将角色附加到 Neptune 集群：

```
iam:GetAccountSummary on resource: *
iam:ListAccountAliases on resource: *
iam:PassRole on resource: * with iam:PassedToService restricted to
 rds.amazonaws.com
```

创建 IAM 角色以允许 Amazon Neptune 访问 Amazon S3 资源

使用 AmazonS3ReadOnlyAccess 托管式 IAM policy 创建一个新的 IAM 角色，该角色将允许 Amazon Neptune 访问 Amazon S3 资源。

创建新的 IAM 角色以允许 Neptune 访问 Amazon S3

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择 Roles（角色）。
3. 选择 创建角色。
4. 在 AWS 服务下，选择 S3。
5. 选择下一步: 权限。
6. 使用筛选框按术语 S3 进行筛选，然后选中 A mazonS3 A ReadOnly cces s 旁边的复选框。

> **Note**
>
> 此策略向所有存储桶授予 s3:Get* 和 s3:List* 权限。后续步骤使用信任策略限制对角色的访问。
> 加载程序只需要从中加载的桶的 s3:Get* 和 s3:List* 权限，这样您也可以通过 Amazon S3 资源限制这些权限。
> 如果 S3 存储桶已加密，您需要添加 kms:Decrypt 权限

7. 选择 下一步: 审核。
8. 将角色名称设置为您的 IAM 角色的名称，例如：NeptuneLoadFromS3。也可以添加可选的角色描述值，例如，“允许 Neptune 代表您访问 Amazon S3 资源。”
9. 请选择 创建角色。
10. 在导航窗格中，选择角色。
11. 在Search (搜索) 字段中，输入已创建的角色的名称，然后选择列表中显示的角色。
12. 在 Trust Relationships (信任关系) 选项卡上，选择 Edit trust relationship (编辑信任关系)。

13. 在文本字段中，粘贴以下信任策略。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

14. 选择更新信任策略。
15. 完成将 IAM 角色添加到 Amazon Neptune 集群中的步骤。

将 IAM 角色添加到 Amazon Neptune 集群

使用控制台将 IAM 角色添加到 Amazon Neptune 集群。这允许集群中的任何 Neptune 数据库实例代入此角色并从 Amazon S3 加载。

> **Note**
>
> Amazon Neptune 控制台需要用户具有以下 IAM 权限，以将角色附加到 Neptune 集群：
>
> ```
> iam:GetAccountSummary on resource: *
> iam:ListAccountAliases on resource: *
> iam:PassRole on resource: * with iam:PassedToService restricted to
>  rds.amazonaws.com
> ```

将 IAM 角色添加到 Amazon Neptune 集群

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。

2. 在导航窗格中，选择数据库。
3. 为要修改的集群选择集群标识符。
4. 选择 “连接和安全” 选项卡。
5. 在 IAM 角色部分，选择您在上一节中创建的角色。
6. 选择 Add role (添加角色)。
7. 请耐心等待，直到 IAM 角色可供集群使用，方可使用它。

创建 Amazon S3 VPC 端点

Neptune 加载程序需要 Amazon S3 的 VPC 端点（类型为网关）。

设置 Amazon S3 访问权限

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择端点。
3. 选择 Create Endpoint（创建端点）。
4. 为网关类型的端点选择服务名称 `com.amazonaws.region.s3`。

> **Note**
>
> 如果此处的区域不正确，请确保控制台区域正确。

5. 选择包含 Neptune 数据库实例的 VPC（它在 Neptune 控制台中为您的数据库实例列出）。
6. 选中与子网 (与集群相关) 关联的路由表旁边的复选框。如果只有一个路由表，则必须选中该框。
7. 选择 Create Endpoint（创建端点）。

有关创建端点的信息，请参阅《Amazon VPC 用户指南》中的 VPC 端点。有关 VPC 端点的限制的信息，请参阅 Amazon S3 的 VPC 端点。

后续步骤

现在已授予对 Amazon S3 桶的访问权限，可准备加载数据。有关所支持格式的信息，请参阅加载数据格式。

在 Amazon Neptune 中串联 IAM 角色

> **Important**
>
> 在某些情况下，引擎版本 1.2.1.0.R3 中引入的利用串联 IAM 角色的新批量加载跨账户特征可能会导致您观察到批量加载性能下降。因此，支持此特征的引擎版本的升级已暂停，直到此问题得到解决。

当您将角色附加到集群时，集群可以代入该角色来访问存储在 Amazon S3 中的数据。从引擎版本 1.2.1.0.R3 开始，如果该角色无法访问您需要的所有资源，则可以串联一个或多个其它角色，您的集群可以代入这些角色来访问其它资源。链中的每个角色都会代入链中的下一个角色，直到集群代入位于链尾的角色。

要串联角色，您可以在角色之间建立信任关系。例如，要将 `RoleB` 串联到 `RoleA`，`RoleA` 必须有允许它代入 `RoleB` 的权限策略，并且 `RoleB` 必须有允许它将其权限传回给 `RoleA` 的信任策略。有关更多信息，请参阅使用 IAM 角色。

链中的第一个角色必须附加到正在加载数据的集群。

第一个角色以及代入链中以下角色的每个后续角色必须具有：

- 一项策略，其中包含对 `sts:AssumeRole` 操作产生 `Allow` 影响的特定语句。
- `Resource` 元素中下一个角色的 Amazon 资源名称 (ARN)。

> **Note**
>
> 目标 Amazon S3 存储桶必须与集群位于同一 AWS 区域。

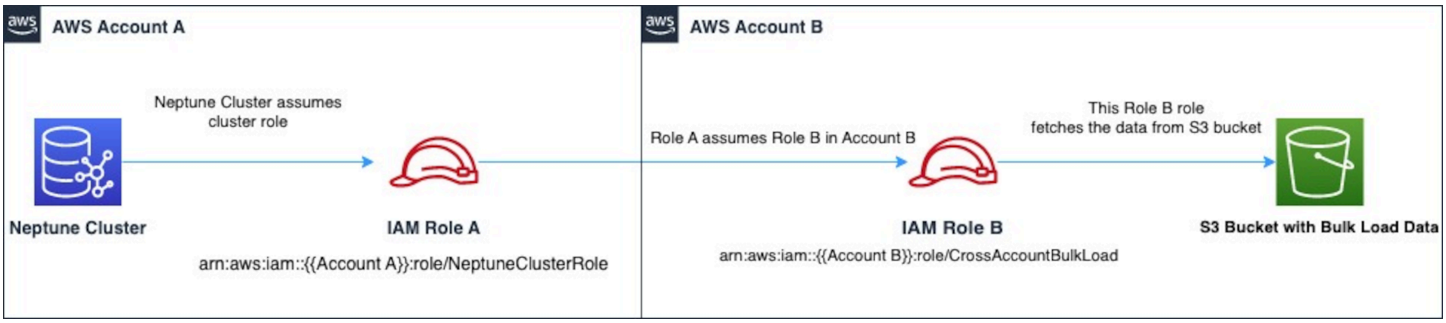
使用串联角色进行跨账户访问

您可以通过串联属于另一个账户的一个或多个角色来授予跨账户访问权限。当您的集群暂时代入属于另一个账户的角色时，它可以访问那里的资源。

例如，假设账户 A 想要访问属于账户 B 的 Amazon S3 桶中的数据：

- 账户 A 为 Neptune 创建一个名为的 AWS 服务角色`RoleA`并将其附加到集群。
- 账户 B 创建一个名为 `RoleB` 的角色，该角色有权访问账户 B 桶中的数据。

- 账户 A 向 RoleA 附加了允许它代入 RoleB 的权限策略。
- 账户 B 向 RoleB 附加了允许它将其权限传递回 RoleA 的信任策略。
- 要访问账户 B 桶中的数据，账户 A 需要使用串联 RoleA 和 RoleB 的 iamRoleArn 参数运行加载程序命令。在加载程序操作的持续时间内，RoleA 将临时代入 RoleB 以访问账户 B 中的 Amazon S3 桶。



例如，RoleA 具有一项与 Neptune 建立信任关系的信任策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
          "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

RoleA 还将有一个允许它代入 RoleB 的权限策略，该策略归账户 B 所有：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1487639602000",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
```

```
      "Resource": "arn:aws:iam::(Account B ID):role/RoleB"
    }
  ]
}
```

相反，RoleB 将具有一项信任策略来与 RoleA 建立信任关系：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::(Account A ID):role/RoleA"
      }
    }
  ]
}
```

RoleB 还需要获得权限才能访问位于账户 B 的 Amazon S3 桶中的数据。

创建 AWS Security Token Service (STS) VPC 终端节点

Neptune 加载器需要一个 VPC 终端节点 AWS STS ，以便您链接 IAM 角色才能通过私有 IP 地址私有访问 AWS STS API。您可以 AWS STS 通过安全且可扩展的方式从 Amazon VPC 直接连接到 VPC 终端节点。当您使用接口 VPC 端点时，它提供更好的安全态势，因为您无需打开出站流量防火墙。它还提供使用 Amazon VPC 端点的其它好处。

使用 VPC 终端节点时，流向的流量 AWS STS 不会通过互联网传输，也永远不会离开 Amazon 网络。您的 VPC 已安全连接， AWS STS 不会对您的网络流量造成可用性风险或带宽限制。有关更多信息，请参阅使用 AWS STS 接口 VPC 端点。

要为 AWS Security Token Service (STS) 设置访问权限

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在导航窗格中，选择端点。
3. 选择 Create Endpoint（创建端点）。
4. 为接口类型的端点选择服务名称：com.amazonaws.region.sts。

5. 选择包含您的 Neptune 数据库实例和 EC2 实例的 VPC。
6. 选择您的 EC2 实例所在子网旁边的复选框。您无法从同一可用区中选择多个子网。
7. 对于 IP address type（IP 地址类型），可从以下选项中进行选择：
   - IPv4 – 将 IPv4 地址分配给端点网络接口。仅当所有选定子网都具有 IPv4 地址范围时，才支持此选项。
   - IPv6 – 将 IPv6 地址分配给端点网络接口。仅当所有选定子网均为仅限 IPv6 的子网时，才支持此选项。
   - Dualstack（双堆栈）– 将 IPv4 和 IPv6 地址分配给端点网络接口。仅当所有选定子网都具有 IPv4 和 IPv6 地址范围时，才支持此选项。
8. 对于 Security groups（安全组），选择要与 VPC 端点的端点网络接口关联的安全组。您需要选择附加到您的 Neptune 数据库实例和 EC2 实例的所有安全组。
9. 对于 Policy（策略），选择 Full access（完全访问权限）以允许所有主体通过 VPC 端点对所有资源执行所有操作。否则，选择 Custom（自定义）以附加 VPC 端点策略，该策略控制主体通过 VPC 端点对资源执行操作的权限。该选项仅在服务支持 VPC 端点策略时可用。有关更多信息，请参阅端点策略。
10. （可选）要添加标签，请选择添加新标签，然后输入您需要的标签键和标签值。
11. 选择创建端点。

有关创建端点的信息，请参阅《Amazon VPC 用户指南》中的 VPC 端点。请注意，Amazon STS VPC 端点是 IAM 角色串联所需的先决条件。

现在，您已授予对 AWS STS 终端节点的访问权限，可以准备加载数据了。有关支持的格式的信息，请参阅加载数据格式。

在加载程序命令中串联角色

当运行加载程序命令时，可以通过在 `iamRoleArn` 参数中包含一个逗号分隔的角色 ARN 列表来指定角色串联。

虽然大多数情况下在一个链中只需要有两个角色，但也可以将三个或更多的角色串联在一起。例如，这个加载程序命令串联三个角色：

```
curl -X POST https://localhost:8182/loader \
  -H 'Content-Type: application/json' \
  -d '{
        "source" : "s3://(the target bucket name)/(the target date file name)",
```

```
        "iamRoleArn" : "arn:aws:iam::(Account A ID):role/(RoleA),arn:aws:iam::(Account
B ID):role/(RoleB),arn:aws:iam::(Account C ID):role/(RoleC)",
        "format" : "csv",
        "region" : "us-east-1"
      }'
```

加载数据格式

Amazon Neptune Load API 支持加载各种格式的数据。

属性图加载格式

然后，可以同时使用 Gremlin 和 openCypher 查询按以下属性图格式之一加载的数据：

- Gremlin 加载数据格式 (`csv`)：逗号分隔值 (CSV) 格式。
- openCypher 数据加载格式 (`opencypher`)：逗号分隔值 (CSV) 格式。

RDF 加载格式

要加载使用 SPARQL 查询的资源描述框架 (RDF) 数据，可以按照万维网联盟 (W3C) 的规定使用以下标准格式之一：

- 规范中的 N -Triples (`ntriples`)（位于 https://www.w3.org/TR/n-triples/ ）
- 规范中的 N-Quads (`nquads`)（位于 https://www.w3.org/TR/n-quads/ ）
- 规范中的 RDF/XML (`rdfxml`)（位于 https://www.w3.org/TR/rdf-syntax-grammar/ ）
- 规范中的 Turtle (`turtle`)（位于 https://www.w3.org/TR/turtle/ ）

加载数据必须使用 UTF-8 编码

> **Important**
>
> 所有加载数据文件必须采用 UTF-8 格式编码。如果文件不是 UTF-8 编码的，Neptune 无论如何都会尝试将其加载为 UTF-8。

对于包含 Unicode 字符的 N-Quads 和 N-triples 数据，支持 \u*xxxxx* 转义序列。但是，Neptune 不支持标准化。如果存在需要归一化的值，则 byte-to-byte 在查询期间该值将不匹配。有关标准化的更多信息，请参阅 Unicode.org 上的标准化页面。

如果数据的格式不受支持，则必须先转换数据，然后加载它。

Graphml2CSV 项目中提供了将 GraphML 转换为 Neptune CSV 格式的工具。GitHub

对加载数据文件的压缩支持

Neptune 支持以 `gzip` 或 `bzip2` 格式压缩各个文件。

压缩文件必须具有 `.gz` 或 `.bz2` 扩展名，并且必须是以 UTF-8 格式编码的单个文本文件。可以加载多个文件，但每个文件必须是单独的 `.gz`、`.bz2` 或未压缩的文本文件。不支持扩展名如 `.tar`、`.tar.gz` 和 `.tgz` 的归档文件。

以下各节对格式进行了详述。

主题

- Gremlin 加载数据格式
- openCypher 数据的加载格式
- RDF 加载数据格式

Gremlin 加载数据格式

要使用 CSV 格式加载 Apache TinkerPop Gremlin 数据，必须在单独的文件中指定顶点和边。

加载程序可以在单个加载任务中从多个顶点文件和多个边缘文件加载。

对于每个加载命令，要加载的文件集必须与 Amazon S3 桶位于同一文件夹中，并且为 `source` 参数指定文件夹名称。文件名和文件扩展名不重要。

Amazon Neptune CSV 格式遵循 RFC 4180 CSV 规范。有关更多信息，请参阅 Internet Engineering Task Force (IETF) 网站上的 CSV 文件的一般格式和 MIME 类型。

> **Note**
>
> 所有文件必须采用 UTF-8 格式编码。

每个文件都包含一个逗号分隔的标题行。此标题行由系统列标题和属性列标题组成。

系统列标题

顶点文件和边文件需要和允许的系统列标题不同。

每个系统列在标题中只能出现一次。

所有标签都区分大小写。

顶点标题

- `~id` - 必需

  顶点的 ID。

- `~label`

  顶点的标签。允许使用多个标签值，用分号 (`;`) 分隔。

  如果不存在，`~label`则为标签 TinkerPop 提供该值`vertex`，因为每个顶点必须至少有一个标签。

边标题

- `~id` - 必需

  边的 ID。

- `~from` - 必需

  源 顶点的顶点 ID。

- `~to` - 必需

  目标 顶点的顶点 ID。

- `~label`

  边的标签。边只能具有单个标签。

  如果不存在，`~label`则为标签 TinkerPop 提供该值`edge`，因为每条边都必须有一个标签。

属性列标题

可通过使用以下语法指定属性的列 (`:`)。类型名称不区分大小写。但请注意，如果属性名称中出现冒号，则必须在其前面加上反斜杠来对其进行转义：`\:`。

```
propertyname:type
```

> **Note**
>
> 列标题中不允许使用空格、逗号、回车符和换行符，因此属性名称不能包含这些字符。

您可以通过将 `[]` 添加到类型来指定数组列：

```
propertyname:type[]
```

> **Note**
>
> 边缘属性只能有一个值，如果指定了数组类型或指定了另一个值，则会导致错误。

以下示例显示了名为 `age`、类型为 `Int` 的属性的列标题。

```
age:Int
```

文件中的每行都需要在该位置具有整数或保留为空。

允许使用字符串数组，但是数组中的字符串不能包含分号 (`;`) 字符，除非使用反斜杠对其进行转义（如 `\;`）。

指定列的基数

从 版本 1.0.1.0.200366.0（2019 年 7 月 26 日）开始，列标题可用于为该列标识的属性指定基数。这使得批量加载程序可以按照类似于 Gremlin 请求所用的方法来遵循基数。

可以如下所示指定列的基数：

```
propertyname:type(cardinality)
```

该*##* 值可以是 `single` 或 `set`。默认值为 `set`，表示列可以接受多个值。对于边缘文件，基数始终是单一的，并且指定任何其他基数都会导致加载程序引发异常。

当基数是 `single` 时，如果加载某个值时以前的值已存在，或者如果加载了多个值，则加载程序引发错误。此行为可以覆盖，因此在使用 `updateSingleCardinalityProperties` 标记加载新值时，将替换现有值。请参阅 加载程序命令。

可以将基数设置用于数组类型，虽然通常并不需要这样做。以下是可能的组合：

- `name:type` – 基数为 `set`，内容为单值。
- `name:type[]` – 基数为 `set`，内容为多值。
- `name:type(single)` – 基数为 `single`，内容为单值。
- `name:type(set)` – 基数为 `set`，这与默认值相同，内容为单值。
- `name:type(set)[]` – 基数为 `set`，内容为多值。
- `name:type(single)[]` – 这自相矛盾，导致引发错误。

以下部分列出了所有可用 Gremlin 数据类型。

Gremlin 数据类型

下面列出了允许的属性值，以及对每种类型的描述。

Bool (或 Boolean)

指示 Boolean 字段。允许的值：`false`、`true`

> **Note**
>
> `true` 以外的任何值都将被视为 False。

整数类型

超出所定义范围的值将导致错误。

类型	Range
字节	-128 到 127
短型	-32768 到 32767
Int	-2^31 到 2^31-1
Long	-2^63 到 2^63-1

小数类型

支持十进制记数法或科学记数法。此外允许使用符号 [如 (+/-) Infinity 或 NaN]。不支持 INF。

类型	Range
浮点型	32 位 IEEE 754 浮点
Double	64 位 IEEE 754 浮点

对于太长的浮点值和双精度值，将加载并四舍五入到最近的 24 位 (浮点) 和 53 位 (双精度) 值。对于位级别的最后剩余数位，中间值将四舍五入到 0。

String

引号是可选的。如果双引号 (") 括起来的字符串中包含逗号字符、换行符和回车符，则将自动对这些字符进行转义。示例：`"Hello, World"`

要在用引号引起来的字符串中包含引号，可通过在一行中使用两个引号来对引号进行转义：示例：`"Hello ""World"""`

允许使用字符串数组，但是数组中的字符串不能包含分号 (;) 字符，除非使用反斜杠对其进行转义（如 `\;`）。

如果要使用引号将数组内的字符串括起来，则必须使用一组引号将整个数组括起来。示例：`"String one; String 2; String 3"`

Date

ISO-8601 格式的Java 日期。支持以下格式：`yyyy-MM-dd`、`yyyy-MM-ddTHH:mm`、`yyyy-MM-ddTHH:mm:ss`、`yyyy-MM-ddTHH:mm:ssZ`

Gremlin 行格式

分隔符

行中的字段是用逗号分隔的。记录是用换行符或换行符后跟回车符来分隔的。

空白字段

非必需列 (如用户定义的属性) 允许使用空白字段。空白字段仍需要逗号分隔符。必填列上的空白字段将导致解析错误。空字符串值被解释为该字段的空字符串值；而不是空白字段。下一节中的示例在每个示例顶点中都有一个空白字段。

顶点 ID

每个顶点文件中所有顶点的 ~id 值都必须是唯一的。~id 值相同的多个顶点行适用于图形中的单个顶点。空字符串 ("") 是有效的 id，顶点是使用空字符串作为 id 创建的。

边 ID

此外，每个边文件中所有边的 ~id 值都必须是唯一的。~id 值相同的多个边行适用于图形中的单个边。空字符串 ("") 是有效的 ID，边缘是使用空字符串作为 ID 创建的。

标签

标签区分大小写，不能为空。值为""将导致错误。

字符串值

引号是可选的。如果双引号 (") 括起来的字符串中包含逗号字符、换行符和回车符，则将自动对这些字符进行转义。空字符串值("")被解释为该字段的空字符串值；而不是空白字段。

CSV 格式规范

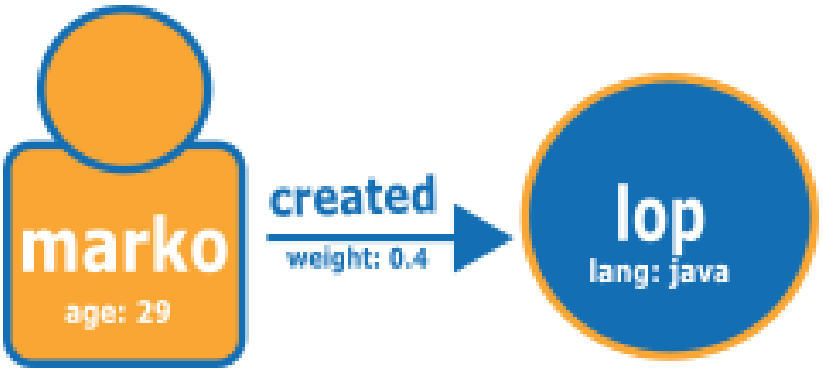
Neptune CSV 格式遵循 RFC 4180 CSV 规范，其中包含以下要求。

- 支持 Unix 和 Windows 样式行结尾 (\n 或 \r\n)。
- 任何字段都可以使用引号引起来 (使用双引号)。
- 包含换行符、双引号或逗号的字段必须用引号引起来。(如果未引起来，加载将立即中止。)
- 字段中的双引号字符 (") 必须用两个 (双) 引号字符表示。例如，字符串在数据中 `Hello "World"` 必须显示为 `"Hello ""World"""`。
- 将忽略分隔符之间的周围空格。如果某行显示为`value1, value2`，则它们存储为`"value1"`和`"value2"`。
- 任何其他转义字符将以逐字字符串形式存储。例如，`"data1\tdata2"` 将以 `"data1\tdata2"` 的形式存储。只要这些字符围在引号内，就不需要任何其他转义。
- 允许使用空白字段。空白字段将视为空值。
- 字段的多个值是在值之间使用分号 (`;`) 来指定的。

有关更多信息，请参阅 Internet Engineering Task Force (IETF) 网站上的 CSV 文件的一般格式和 MIME 类型。

Gremlin 示例

下图显示了取自 TinkerPop 现代图的两个顶点和一条边的示例。



以下是 Neptune CSV 加载格式的图形。

顶点文件：

```
~id,name:String,age:Int,lang:String,interests:String[],~label
v1,"marko",29,,"sailing;graphs",person
v2,"lop",,"java",,software
```

顶点文件的表格视图：

~id	name:String	age:Int	lang:String	兴趣:字符串 []	~label
v1	"marko"	29		[“帆船”，“图表”]	person
v2	"lop"		"java"		软件

边文件：

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

边文件的表格视图：

~id	~from	~到	~label	weight:Double

e1	v1	v2	created	0.4

后续步骤

现在要了解有关加载格式的更多信息，请参阅示例：将数据加载到 Neptune 数据库实例中。

openCypher 数据的加载格式

要使用 openCypher CSV 格式加载 openCypher 数据，必须在单独的文件中指定节点和关系。加载程序可以在单个加载任务中从多个节点文件和关系文件中加载。

对于每个加载命令，要加载的文件集在 Amazon Simple Storage Service 桶中必须具有相同的路径前缀。您可以在源参数中指定该前缀。实际的文件名和扩展名不重要。

在 Amazon Neptune 中，openCypher CSV 格式符合 RFC 4180 CSV 规范。有关更多信息，请参阅 Internet Engineering Task Force (IETF) 网站上的 CSV 文件的一般格式和 MIME 类型 (https://tools.ietf.org/html/rfc4180)。

> **Note**
>
> 这些文件必须采用 UTF-8 格式编码。

每个文件都有一个以逗号分隔的标题行，其中包含系统列标题和属性列标题。

openCypher 数据加载文件中的系统列标题

给定的系统列在标题中只能出现一次。所有系统列标题标签均区分大小写。

openCypher 节点加载文件和关系加载文件所需的和允许的系统列标题不同：

节点文件中的系统列标题

- **`:ID`** – (必需) 节点的 ID。

  可以在节点 `:ID` 列标题中添加可选的 ID 空间，例如：`:ID(`*`ID Space`*`)`。例如，`:ID(movies)`。

  加载连接此文件中节点的关系时，请在关系文件的 `:START_ID` 和/或 `:END_ID` 列中使用相同的 ID 空间。

  可以选择将节点 `:ID` 列存储为采用 *`property name`*`:ID` 格式的属性。例如，`name:ID`。

在当前和之前加载的所有节点文件中，节点 ID 应是唯一的。如果使用 ID 空间，则在当前和之前的加载中使用相同 ID 空间的所有节点文件中，节点 ID 应是唯一的。

- **`:LABEL`** – 节点的标签。

  允许使用多个标签值，用分号 (`;`) 分隔。

关系文件中的系统列标题

- **`:ID`** – 关系的 ID。当 `userProvidedEdgeIds` 为 true（默认）时，这是必需的，但当 `userProvidedEdgeIds` 为 `false` 时，则无效。

  在当前和之前加载的所有关系文件中，关系 ID 应是唯一的。
- **`:START_ID`** –（必需）此关系的起始节点的节点 ID。

  或者，ID 空间可以与格式为 `:START_ID(`*`ID Space`*`)` 的起始 ID 列相关联。分配给起始节点 ID 的 ID 空间应与在节点文件中分配给该节点的 ID 空间相匹配。
- **`:END_ID`** –（必需）此关系的结束节点的节点 ID。

  或者，ID 空间可以与格式为 `:END_ID(`*`ID Space`*`)` 的结束 ID 列相关联。分配给结束节点 ID 的 ID 空间应与在其节点文件中分配给该节点的 ID 空间相匹配。
- **`:TYPE`** – 关系的类型。关系只能有单一类型。

> **Note**
>
> 有关批量加载过程如何处理重复的节点 ID 或关系 ID 的信息，请参阅加载 openCypher 数据。

openCypher 数据加载文件中的属性列标题

您可以使用以下格式的属性列标题来指定某一列包含特定属性的值：

```
propertyname:type
```

列标题中不允许使用空格、逗号、回车符和换行符，因此属性名称不能包含这些字符。以下是名为 `age` 且类型为 `Int` 的属性的列标题示例：

```
age:Int
```

然后，age:Int 作为列标题的列必须在每行中包含一个整数或一个空值。

Neptune openCypher 数据加载文件中的数据类型

- **Bool** 或 **Boolean** – 布尔型字段。允许的值包括 true和 false。

  除 true 之外的任何值都视为 false。
- **Byte** – 范围为 -128 至 127 的整数。
- **Short** – 范围为 -32,768 至 32,767 的整数。
- **Int** – 范围为 -2^31 至 2^31 - 1 的整数。
- **Long** – 范围为 -2^63 至 2^63 - 1 的整数。
- **Float** – 32 位 IEEE 754 浮点数。支持十进制记数法和科学记数法。可全部识别 Infinity、-Infinity、和 NaN，但不识别 INF。

  位数太多而无法容纳的值将四舍五入到最接近的值（对于位级别的最后一个剩余数字，中间值将舍入为 0）。
- **Double** – 64 位 IEEE 754 浮点数。支持十进制记数法和科学记数法。可全部识别 Infinity、-Infinity、和 NaN，但不识别 INF。

  位数太多而无法容纳的值将四舍五入到最接近的值（对于位级别的最后一个剩余数字，中间值将舍入为 0）。
- **String** – 引号是可选的。如果双引号 (") 括起来的字符串中包含逗号字符、换行符和回车符（如 "Hello, World"），则将自动对这些字符进行转义。

  您可以通过连续使用两个引号在带引号的字符串中加入引号，比如 "Hello ""World"""。
- **DateTime** – 采用以下 ISO-8601 格式之一的 Java 日期：
  - yyyy-MM-dd
  - yyyy-MM-ddTHH:mm
  - yyyy-MM-ddTHH:mm:ss
  - yyyy-MM-ddTHH:mm:ssZ

Neptune openCypher 数据加载文件中的自动转换数据类型

提供自动转换数据类型是为了加载 Neptune 目前原生不支持的数据类型。此类列中的数据以字符串形式存储，逐字存储，无需根据其预期格式进行验证。允许使用以下自动转换数据类型：

- **Char** – Char 字段。存储为字符串。

- **`Date`、`LocalDate` 和 `LocalDateTime`** – 有关 `date`、`localdate` 和 `localdatetime` 类型的描述，请参阅 Neo4j 瞬时。这些值作为字符串逐字加载，无需验证。
- **`Duration`** – 请参阅 Neo4j 持续时间格式。这些值作为字符串逐字加载，无需验证。
- 点 – 用于存储空间数据的点字段。请参阅空间瞬时。这些值作为字符串逐字加载，无需验证。

openCypher 加载格式示例

下图取自 M TinkerPop odern Graph，显示了两个节点和一个关系的示例：

以下是正常的 Neptune openCypher 加载格式的图形。

节点文件：

```
:ID,name:String,age:Int,lang:String,:LABEL
v1,"marko",29,,person
v2,"lop",,"java",software
```

关系文件：

```

:ID,:START_ID,:END_ID,:TYPE,weight:Double
e1,v1,v2,created,0.4
```

或者，您可以使用 ID 空间和 ID 作为属性，如下所示：

第一个节点文件：

```

name:ID(person),age:Int,lang:String,:LABEL
"marko",29,,person
```

第二个节点文件：

```
name:ID(software),age:Int,lang:String,:LABEL
"lop",,"java",software
```

关系文件：

```
:ID,:START_ID,:END_ID,:TYPE,weight:Double
e1,"marko","lop",created,0.4
```

RDF 加载数据格式

要加载资源描述框架 (RDF) 数据，可以按照万维网联盟 (W3C) 的规定使用以下标准格式之一：

- 规范中的 N -Triples (`ntriples`) (位于 https://www.w3.org/TR/n-triples/)
- 规范中的 N-Quads (`nquads`) (位于 https://www.w3.org/TR/n-quads/)
- 规范中的 RDF/XML (`rdfxml`) (位于 https://www.w3.org/TR/rdf-syntax-grammar/)
- 规范中的 Turtle (`turtle`) (位于 https://www.w3.org/TR/turtle/)

> **Important**
>
> 所有文件必须采用 UTF-8 格式编码。
> 对于包含 Unicode 字符的 N-Quads 和 N-triples 数据，支持 \u*xxxxx* 转义序列。但是，Neptune 不支持标准化。如果存在需要归一化的值，则 byte-to-byte 在查询期间该值将不匹配。有关标准化的更多信息，请参阅 Unicode.org 上的标准化页面。

后续步骤

现在要了解有关加载格式的更多信息，请参阅示例：将数据加载到 Neptune 数据库实例中。

示例：将数据加载到 Neptune 数据库实例中

此示例演示如何将数据加载到 Amazon Neptune 中。除非另有说明，否则您必须在与 Neptune 数据库实例相同的 Amazon Virtual Private Cloud (VPC) 中的 Amazon Elastic Compute Cloud (Amazon EC2) 实例中执行这些步骤。

数据加载示例的先决条件

在开始之前，您必须具有以下内容：

- 一个 Neptune 数据库实例。

  有关启动 Neptune 数据库实例的信息，请参阅创建新的 Neptune 数据库集群。
- 将数据文件放入其中的 Amazon Simple Storage Service (Amazon S3) 桶。

  您可以使用现有存储段。如果没有 S3 桶，请参阅 Amazon S3 入门指南中的创建桶。
- 为以 Neptune 加载程序支持的格式之一加载的数据绘制图形：

  如果您使用 Gremlin 来查询图表，Neptune 可以加载 comma-separated-values (CSV) 格式的数据，如中所述。Gremlin 加载数据格式

  如果您使用 openCypher 查询图形，Neptune 还可以按特定于 openCypher 的 CSV 格式加载数据，如openCypher 数据的加载格式中所述。

  如果您使用 SPARQL，则 Neptune 可加载采用大量 RDF 格式的数据，如RDF 加载数据格式中所述。
- Neptune 数据库实例要代入的 IAM 角色，此角色具有允许访问 S3 桶中的数据文件的 IAM policy。此策略必须授予读取和列出权限。

  有关创建对 Amazon S3 具有访问权限的角色，然后将其与 Neptune 集群关联的信息，请参阅先决条件：IAM 角色和 Amazon S3 访问权限。

  > **Note**
  >
  > Neptune Load API 只需要对数据文件的读取访问权限。IAM policy 不需要允许对整个桶的写入访问权限或访问权限。
- Amazon S3 VPC 端点。想要了解更多信息，请参阅创建 Amazon S3 VPC 端点部分。

创建 Amazon S3 VPC 端点

Neptune 加载程序需要 Amazon S3 的 VPC 端点。

设置 Amazon S3 访问权限

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，网址为 https://console.aws.amazon.com/vpc/。
2. 在左侧导航窗格中，选择终端节点。
3. 选择 Create Endpoint（创建端点）。

4. 选择 Service Name (服务名称) `com.amazonaws.region.s3`。

   > **Note**
   >
   > 如果此处的区域不正确，请确保控制台区域正确。

5. 选择包含您的 Neptune 数据库实例的 VPC。
6. 选中与子网 (与集群相关) 关联的路由表旁边的复选框。如果只有一个路由表，则必须选中该框。
7. 选择 Create Endpoint (创建端点) 。

有关创建端点的信息，请参阅《Amazon VPC 用户指南》中的 VPC 端点。有关 VPC 端点的限制的信息，请参阅 Amazon S3 的 VPC 端点。

将数据加载到 Neptune 数据库实例中

1. 将数据文件复制到 Amazon S3 桶。S3 存储桶必须与加载数据的集群位于同一 AWS 区域。

   您可以使用以下 AWS CLI 命令将文件复制到存储桶。

   > **Note**
   >
   > 此命令不需要从 Amazon EC2 实例运行。

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

   > **Note**
   >
   > 在 Amazon S3 中，对象键名称是文件的整个路径，其中包含文件名。
   > 示例：在命令 `aws s3 cp datafile.txt s3://examplebucket/mydirectory/datafile.txt` 中，对象键名称为 **`mydirectory/datafile.txt`**。

   或者，您可以使用将文件上传 AWS Management Console 到 S3 存储桶。请通过以下网址打开 Amazon S3 控制台：https://console.aws.amazon.com/s3/，然后选择桶。在左上角，选择 Upload (上传) 以上传文件。

2. 从命令行窗口中，使用端点、Amazon S3 路径、格式和 IAM 角色 ARN 的正确值输入以下内容，以运行 Neptune 加载程序。

   format 参数可以是下列任意值：csv（对于 Gremlin），opencypher（对于 openCypher），或者 ntriples、nquads、turtle 和 rdfxml（对于 RDF）。有关其他参数的信息，请参阅Neptune 加载程序命令。

   有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点部分。

   区域参数必须与集群和 S3 存储桶的区域匹配。

   亚马逊 Neptune 在以下 AWS 地区上市：

   - 美国东部（弗吉尼亚州北部）：us-east-1
   - 美国东部（俄亥俄州）：us-east-2
   - 美国西部（北加利福尼亚）：us-west-1
   - 美国西部（俄勒冈州）：us-west-2
   - 加拿大（中部）：ca-central-1
   - 南美洲（圣保罗）：sa-east-1
   - 欧洲地区（斯德哥尔摩）：eu-north-1
   - 欧洲地区（爱尔兰）：eu-west-1
   - 欧洲地区（伦敦）：eu-west-2
   - 欧洲地区（巴黎）：eu-west-3
   - 欧洲地区（法兰克福）：eu-central-1
   - 中东（巴林）：me-south-1
   - 中东（阿联酋）：me-central-1
   - 以色列（特拉维夫）：il-central-1
   - 非洲（开普敦）：af-south-1
   - 亚太地区（香港）：ap-east-1
   - 亚太地区（东京）：ap-northeast-1
   - 亚太地区（首尔）：ap-northeast-2
   - 亚太地区（大阪）：ap-northeast-3
   - 亚太地区（新加坡）：ap-southeast-1
   - 亚太地区（悉尼）：ap-southeast-2

- 亚太地区（孟买）：ap-south-1
- 中国（北京）：cn-north-1
- 中国（宁夏）：cn-northwest-1
- AWS GovCloud（美国西部）：us-gov-west-1
- AWS GovCloud（美国东部）：us-gov-east-1

```
curl -X POST \
    -H 'Content-Type: application/json' \
    https://your-neptune-endpoint:port/loader -d '
    {
      "source" : "s3://bucket-name/object-key-name",
      "format" : "format",
      "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
      "region" : "region",
      "failOnError" : "FALSE",
      "parallelism" : "MEDIUM",
      "updateSingleCardinalityProperties" : "FALSE",
      "queueRequest" : "TRUE",
      "dependencies" : ["load_A_id", "load_B_id"]
    }'
```

有关创建 IAM 角色并将其与 Neptune 集群关联的信息，请参阅先决条件：IAM 角色和 Amazon S3 访问权限。

> **Note**
>
> 有关加载请求参数的详细信息，请参阅Neptune 加载程序请求参数）。简而言之：
> source 参数接受指向单个文件或文件夹的 Amazon S3 URI。如果指定文件夹，Neptune 将加载该文件夹中的每个数据文件。
> 文件夹可包含多个顶点文件和多个边缘文件。
> URI 可以采用以下任意格式。
>
> - 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 可以是下列项之一：

- Gremlin 属性图的 Gremlin CSV 格式 (`csv`)
- openCypher 属性图的 openCypher CSV 格式 (`opencypher`)
- RDF 的 N -Triples (`ntriples`) 格式 / SPARQL
- RDF 的 N-Quads (`nquads`) 格式 / SPARQL
- RDF 的 RDF/XML (`rdfxml`) 格式 / SPARQL
- RDF 的 Turtle (`turtle`) 格式 / SPARQL

可选 `parallelism` 参数，您可用于限制批量加载进程中使用的线程数。它可以设置为 `LOW`、`MEDIUM`、`HIGH` 或 `OVERSUBSCRIBE`。
将 `updateSingleCardinalityProperties` 设置为 `"FALSE"` 时，如果在为边缘或单基数顶点属性加载的源文件中提供了多个值，则加载程序将返回错误。
如果已有加载作业正在运行，则将 `queueRequest` 设置为 `"TRUE"` 会将加载请求放入队列中。
`dependencies` 参数使加载请求的执行取决于成功完成已放入队列中的一个或多个加载作业。

3. Neptune 加载程序将返回允许检查状态或取消加载过程的任务 `id`，例如：

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

4. 在步骤 3 中输入以下命令以获取具有 `loadId` 的加载的状态：

```
curl -G 'https://your-neptune-endpoint:port/loader/ef478d76-
d9da-4d94-8ff1-08d9d4863aa5'
```

如果加载的状态列出错误，则可以请求更详细的状态和错误的列表。有关更多信息以及示例，请参阅 Neptune 加载程序获取状态 API。

5. (可选) 取消 `Load` 任务。

在步骤 3 中输入以下命令以通过任务 `id` `Delete` 加载程序任务：

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

DELETE 命令将在成功取消后返回 HTTP 代码 200 OK。

不会回滚已完成加载的加载任务中文件的数据。数据仍保留在 Neptune 数据库实例中。

优化 Amazon Neptune 批量加载

使用以下策略将 Neptune 批量加载的加载时间保持在最低限度：

- 清理您的数据：
  - 在加载之前，请务必将数据转换为支持的数据格式。
  - 删除所有重复项或已知错误。
  - 尽可能减少唯一谓词（例如边缘和顶点的属性）的数量。
- 优化您的文件：
  - 如果您从 Amazon S3 桶加载大型文件（例如 CSV 文件），则加载程序会将它们解析为可以并行加载的块，从而为您管理并发性。使用非常大量的小文件可能会减慢此过程。
  - 如果您从 Amazon S3 文件夹加载多个文件，加载程序会自动先加载顶点文件，然后再加载边缘文件。
  - 压缩文件可缩短传输时间。加载程序支持对源文件进行 gzip 压缩。
- 检查您的加载程序设置：
  - 如果在加载期间不需要执行任何其它操作，请使用 OVERSUBSCRIBEparallelism 参数。此参数设置使批量加载程序在运行时使用所有可用的 CPU 资源。它通常需要 60%-70% 的 CPU 容量，才能使操作在 I/O 约束允许的情况下尽可能快地运行。

> **Note**
>
> 如果 parallelism 设置为 OVERSUBSCRIBE 或 HIGH（默认设置），则在加载 openCypher 数据时，线程可能会遇到争用条件和死锁，从而导致 LOAD_DATA_DEADLOCK 错误。在这种情况下，请将 parallelism 设为较低的设置并重试加载。

- 如果您的加载任务将包含多个加载请求，请使用 `queueRequest` 参数。将 `queueRequest` 设置为 `TRUE` 允许 Neptune 对您的请求进行排队，这样您就不必等到一个请求完成后再发出另一个请求。
- 如果您的加载请求正在排队，则可以使用 `dependencies` 参数设置依赖级别，这样一个任务的失败就会导致相关任务失败。这样可以防止加载的数据出现不一致。
- 如果加载任务将涉及更新先前加载的值，请务必将 `updateSingleCardinalityProperties` 参数设置为 `TRUE`。如果不这样做，加载程序会将更新现有单个基数值的尝试视为错误。对于 Gremlin 数据，还会在属性列标题中指定基数（请参阅属性列标题）。

  > **Note**
  >
  > `updateSingleCardinalityProperties` 参数不适用于资源描述框架 (RDF) 数据。

- 您可以使用 `failOnError` 参数来确定当遇到错误时批量加载操作是失败还是继续。此外，您还可以使用 `mode` 参数来确保加载任务从上一个任务失败的地方恢复加载，而不是重新加载已经加载的数据。

- 纵向扩展 - 在批量加载之前，将数据库集群的写入器实例设置为最大大小。请注意，如果执行此操作，则必须同时纵向扩展数据库集群中的所有只读副本实例，或者在加载完数据之前将其移除。

  批量加载完成后，请务必再次缩减写入器实例。

> **Important**
>
> 如果您在批量加载期间由于复制滞后而经历了重复重启只读副本的循环，则您的副本可能无法跟上数据库集群中写入器的速度。要么将读取器扩展到比写入器大，要么在批量加载期间暂时将它们移除，然后在完成后重新创建。

有关设置加载程序请求参数的更多详细信息，请参阅请求参数。

Neptune 加载程序参考

本节描述 Neptune 数据库实例的 HTTP 端点中提供的 Amazon Neptune 的 `Loader` API。

> **Note**
>
> 有关加载程序在出现错误时返回的错误和馈送消息的列表，请参阅Neptune 加载程序错误和源消息。

目录

- 取消任务请求参数
- 取消任务响应语法
- 取消任务错误
- 取消任务错误消息
- 取消任务示例

Neptune 加载程序命令

将 Amazon S3 桶中的数据加载到 Neptune 数据库实例中。

要加载数据，必须将 HTTP POST 请求发送到 `https://your-neptune-endpoint:port/loader` 终端节点。`loader` 请求的参数可在 POST 正文中或作为 URL 编码的参数发送。

> **Important**
>
> MIME 类型必须为 `application/json`。

S3 存储桶必须与集群位于同一 AWS 区域。

> **Note**
>
> 在使用 Amazon S3 SSE-S3 模式加密数据的情况下，您可从 Amazon S3 加载加密数据。在这种情况下，Neptune 能够模拟您的凭证并代表您发出 `s3:getObject` 调用。
> 只要您的 IAM 角色包括访问 AWS KMS的必要权限，您还可以从 Amazon S3 加载使用 SSE-KMS 模式加密的加密数据。如果没有适当的 AWS KMS 权限，批量加载操作将失败并返回`LOAD_FAILED`响应。
> Neptune 目前不支持加载使用 SSE-C 模式加密的 Amazon S3 数据。

您不必等到一个加载任务完成后再开始另一个加载任务。Neptune 可以一次对多达 64 个任务请求进行排队，前提是它们的 `queueRequest` 参数全部设置为 `"TRUE"`。任务的队列顺序将是 first-in-first-out (FIFO)。另一方面，如果您不希望对某个加载任务排队，则可以将其 `queueRequest` 参数设置为 `"FALSE"`（默认值），这样，如果另一个加载任务已在进行中，该加载任务将失败。

您可以使用 dependencies 参数对只有在队列中指定的先前作业成功完成后才能运行的作业进行排队。如果执行此操作，并且这些指定的任何作业失败，则您的作业将不会运行，其状态将设置为 LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED。

Neptune 加载程序请求语法

```
{
  "source" : "string",
  "format" : "string",
  "iamRoleArn" : "string",
  "mode": "NEW|RESUME|AUTO",
  "region" : "us-east-1",
  "failOnError" : "string",
  "parallelism" : "string",
  "parserConfiguration" : {
    "baseUri" : "http://base-uri-string",
    "namedGraphUri" : "http://named-graph-string"
  },
  "updateSingleCardinalityProperties" : "string",
  "queueRequest" : "TRUE",
  "dependencies" : ["load_A_id", "load_B_id"]
}
```

Neptune 加载程序请求参数

- **`source`** – Amazon S3 URI。

  SOURCE 参数接受标识单个文件、多个文件、一个文件夹或多个文件夹的 Amazon S3 URI。Neptune 将每个数据文件加载到任何指定的文件夹中。

  URI 可以采用以下任意格式。

  - 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*

  URI 的object-key-name元素等同于 Amazon S3 ListObjectsAPI 调用中的前缀参数。它可以识别指定的 Amazon S3 桶中名称以该前缀开头的所有对象。可以是单个文件或文件夹，也可以是多个文件和/或文件夹。

  指定的一个或多个文件夹可以包含多个顶点文件和多个边缘文件。

例如，如果您在名为的 Amazon S3 存储桶中有以下文件夹结构和文件bucket-name：

```
s3://bucket-name/a/bc
s3://bucket-name/ab/c
s3://bucket-name/ade
s3://bucket-name/bcd
```

如果将 source 参数指定为s3://bucket-name/a，则将加载前三个文件。

```
s3://bucket-name/a/bc
s3://bucket-name/ab/c
s3://bucket-name/ade
```

- **format** – 数据的格式。有关 Neptune Loader 命令的数据格式的更多信息，请参阅使用 Amazon Neptune 批量加载程序收集数据。

  允许值

  - **csv** 适用于 Gremlin CSV 数据格式。
  - **opencypher** 适用于 openCypher CSV 数据格式。
  - **ntriples** 适用于 N-Triples RDF 数据格式。
  - **nquads** 适用于 N-Quads RDF 数据格式。
  - **rdfxml** 适用于 RDF\XML RDF 数据格式。
  - **turtle** 适用于 Turtle RDF 数据格式。

- **iamRoleArn** – Neptune 数据库实例为访问 S3 桶而代入的 IAM 角色的 Amazon 资源名称 (ARN)。有关创建对 Amazon S3 具有访问权限的角色，然后将其与 Neptune 集群关联的信息，请参阅先决条件：IAM 角色和 Amazon S3 访问权限。

  从引擎版本 1.2.1.0.R3 开始，如果 Neptune 数据库实例和 Amazon S3 存储桶位于不同的账户中，您还可以链接多个 IAM 角色。 AWS 在本例中，iamRoleArn 包含一个逗号分隔的角色 ARN 列表，如在 Amazon Neptune 中串联 IAM 角色中所述。例如：

```
curl -X POST https://localhost:8182/loader \
  -H 'Content-Type: application/json' \
  -d '{
        "source" : "s3://(the target bucket name)/(the target date file name)",
```

```
       "iamRoleArn" : "arn:aws:iam::(Account A
ID):role/(RoleA),arn:aws:iam::(Account B ID):role/(RoleB),arn:aws:iam::(Account C
ID):role/(RoleC)",
       "format" : "csv",
       "region" : "us-east-1"
     }'
```

- **region**— 该region参数必须与集群的 AWS 区域和 S3 存储桶相匹配。

  Amazon Neptune 在以下 区域中推出：

  - 美国东部（弗吉尼亚州北部）：us-east-1
  - 美国东部（俄亥俄州）：us-east-2
  - 美国西部（北加利福尼亚）：us-west-1
  - 美国西部（俄勒冈州）：us-west-2
  - 加拿大（中部）：ca-central-1
  - 南美洲（圣保罗）：sa-east-1
  - 欧洲地区（斯德哥尔摩）：eu-north-1
  - 欧洲地区（爱尔兰）：eu-west-1
  - 欧洲地区（伦敦）：eu-west-2
  - 欧洲地区（巴黎）：eu-west-3
  - 欧洲地区（法兰克福）：eu-central-1
  - 中东（巴林）：me-south-1
  - 中东（阿联酋）：me-central-1
  - 以色列（特拉维夫）：il-central-1
  - 非洲（开普敦）：af-south-1
  - 亚太地区（香港）：ap-east-1
  - 亚太地区（东京）：ap-northeast-1
  - 亚太地区（首尔）：ap-northeast-2
  - 亚太地区（大阪）：ap-northeast-3
  - 亚太地区（新加坡）：ap-southeast-1
  - 亚太地区（悉尼）：ap-southeast-2
  - 亚太地区（孟买）：ap-south-1
  - 中国（北京）：cn-north-1

- 中国（宁夏）：cn-northwest-1
- AWS GovCloud（美国西部）：us-gov-west-1
- AWS GovCloud（美国东部）：us-gov-east-1

- **mode** – 加载任务模式。

  允许的值：RESUME、NEW、AUTO

  默认值：AUTO

  - RESUME – 在 RESUME 模式下，加载程序查找来自此源的先前加载，如果找到一个加载，则恢复该加载任务。如果未找到之前的加载作业，则加载程序将停止。

    加载程序将避免重新加载之前作业中已成功加载的文件。它仅尝试处理失败的文件。如果您从 Neptune 集群中删除了之前加载的数据，则此模式下不加载该数据。如果之前的加载任务成功加载了来自同一源的所有文件，则不会重新加载任何文件，并且加载程序返回成功。
  - NEW – 在 NEW 模式下，将创建新的加载请求而不管任何之前的加载。您可以使用此模式在从 Neptune 集群删除之前加载的数据之后从源重新加载数据，或加载同一个源处可用的新数据。
  - AUTO – 在 AUTO 模式下，加载程序查找来自此同一个源的先前加载任务，如果找到一个加载任务，则恢复该任务，如同在 RESUME 模式中一样。

    如果加载程序没有找到来自同一源的先前加载作业，则会从源加载所有数据，就像在 NEW 模式中一样。

- **failOnError** – 用于在出错时切换为完全停止的标志。

  允许的值："TRUE"、"FALSE"。

  默认值："TRUE"。

  如果此参数设置为 "FALSE"，则加载程序尝试加载指定位置中的所有数据并跳过任何出错的条目。

  将此参数设置为 "TRUE" 时，加载程序会在遇到错误时立即停止。截至该点加载的数据仍然存在。

- **parallelism** – 这是一个可选参数，可以设置用于减少批量加载过程使用的线程数。

  允许的值：

  - LOW – 所使用的线程数是可用 vCPU 数除以 8。
  - MEDIUM – 所使用的线程数是可用 vCPU 数除以 2。
  - HIGH – 所使用的线程数与可用 vCPU 数相同。

- OVERSUBSCRIBE – 所使用的线程数是可用 vCPU 数乘以 2。如果使用此值，则批量加载程序将占用所有可用资源。

  但是，这并不意味着 OVERSUBSCRIBE 设置会导致 100% 的 CPU 利用率。由于负载操作受到 I/O 限制，因此预期的最高 CPU 利用率在 60% 到 70% 之间。

默认值：HIGH

在加载 openCypher 数据时，parallelism 设置有时会导致线程之间出现死锁。发生这种情况时，Neptune 会返回 LOAD_DATA_DEADLOCK 错误。通常，您可以通过将 parallelism 设为较低的设置并重试加载命令来解决此问题。

- **parserConfiguration** – 具有额外解析程序配置值的可选对象。每个子参数也是可选的：

名称	示例值	描述
namedGraphUri	*http://aws.amazon.com/neptune/vocab/v01/ DefaultNamedGraph*	未指定任何图形时所有 RDF 格式的默认图形（适用于非 quads 格式和无图形的 NQUAD 条目）。默认值为 http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph
baseUri	*http://aws.amazon.com/neptune/default*	RDF/XML 和 Turtle 格式的基本 URI。默认值为 http://aws.amazon.com/neptune/default 。
allowEmptyStrings	*true*	加载 CSV 数据时，Gremlin 用户需要能够将空字符串值 ("") 作为节点和边缘属性传递。如果 allowEmptyStrings 设置为 false（默认值），则此类空字符串将视为 null 且不会加载。

如果 allowEmptyStrings 设置为 true，则加载程序会将空字符串视为有效的属性值并相应地加载它们。

有关更多信息，请参阅 SPARQL 默认图形和命名图形。

- **updateSingleCardinalityProperties** – 这是一个可选参数，用于控制批量加载程序如何处理单基数顶点或边缘属性的新值。加载 openCypher 数据时不支持这样做（请参阅加载 openCypher 数据）。

  允许的值："TRUE"、"FALSE"。

  默认值："FALSE"。

  默认情况下，或者当 updateSingleCardinalityProperties 被显式设置为 "FALSE" 时，加载程序将新值视为错误，因为它违反了单个基数。

  另一方面，当 updateSingleCardinalityProperties 设置为 "TRUE" 时，批量加载程序会用新值替换现有值。如果在要加载的源文件中提供了多个边缘或单基数顶点属性值，则批量加载结束时的最终值可以是这些新值中的任何一个值。加载程序仅保证现有值已被其中一个新值替换。

- **queueRequest** – 这是一个可选的标志参数，用于指示是否可以对加载请求进行排队。

  您不必等到一个加载任务完成就可以发出下一个任务，因为 Neptune 可以一次对多达 64 个任务进行排队，前提是它们的 queueRequest 参数都设置为 "TRUE"。任务的队列顺序将是 first-in-first-out (FIFO)。

  如果省略 queueRequest 参数或将其设置为 "FALSE"，则如果另一个加载作业已在运行，该加载请求将失败。

  允许的值："TRUE"、"FALSE"。

  默认值："FALSE"。

- **dependencies** – 这是一个可选参数，可以使排队的加载请求取决于队列中的一个或多个先前任务的成功完成。

Neptune 可以一次对多达 64 个加载请求进行排队，前提是它们的 queueRequest 参数设置为 "TRUE"。dependencies 参数允许您使此类排队请求的执行取决于成功完成队列中的一个或多个指定的先前请求。

例如，如果加载 Job-A 和 Job-B 彼此独立，但加载 Job-C 在开始之前要求先完成 Job-A 和 Job-B，请按以下方式进行操作：

1. 不分顺序接连提交 load-job-A 和 load-job-B，并保存它们的加载 ID。
2. 提交 load-job-C，并在其 dependencies 字段中填入前两个作业的加载 ID：

```
"dependencies" : ["job_A_load_id", "job_B_load_id"]
```

由于 dependencies 参数，批量加载程序在 Job-A 和 Job-B 成功完成前将不会启动 Job-C。如果其中任何一个作业失败，则不会执行作业 C，并将其状态设置为 LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED。

您可以通过这种方式设置多个依赖关系级别，这样，一个作业的失败将导致所有直接或间接依赖于该作业的请求被取消。

- **userProvidedEdgeIds** – 只有在加载包含关系 ID 的 openCypher 数据时，才需要此参数。当加载数据中显式提供 openCypher 关系 ID 时，必须包含此参数并将它设置为 True（推荐）。

  如果 userProvidedEdgeIds 不存在或设置为 True，则加载过程中的每个关系文件中都必须存在 :ID 列。

  如果 userProvidedEdgeIds 存在且设置为 False，则加载中的关系文件不得包含 :ID 列。相反，Neptune 加载程序会自动为每个关系生成一个 ID。

  显式提供关系 ID 非常有用，这样加载程序就可以在 CSV 数据中的错误得到修复后恢复加载，而不必重新加载任何已经加载的关系。如果没有显式分配关系 ID，当必须更正任何关系文件时，加载程序将无法恢复失败的加载，而是必须重新加载所有关系。

- accessKey – [弃用] 对 S3 桶和数据文件具有访问权限的 IAM 角色的访问密钥 ID。

  建议改用 iamRoleArn 参数。有关创建对 Amazon S3 具有访问权限的角色，然后将其与 Neptune 集群关联的信息，请参阅先决条件：IAM 角色和 Amazon S3 访问权限。

  有关更多信息，请参阅访问密钥（访问密钥 ID 和秘密访问密钥）。

- secretKey – [已弃用] 建议改用 iamRoleArn 参数。有关创建对 Amazon S3 具有访问权限的角色，然后将其与 Neptune 集群关联的信息，请参阅先决条件：IAM 角色和 Amazon S3 访问权限。

有关更多信息，请参阅访问密钥（访问密钥 ID 和秘密访问密钥）。

加载 openCypher 数据的特殊注意事项

- 以 CSV 格式加载 openCypher 数据时，必须将格式参数设置为 opencypher。
- openCypher 加载不支持 updateSingleCardinalityProperties 参数，因为所有 openCypher 属性都具有单一基数。openCypher 加载格式不支持数组，如果一个 ID 值出现多次，则将其视为重复值或插入错误（见下文）。
- Neptune 加载程序按如下方式处理它在 openCypher 数据中遇到的重复项：
  - 如果加载程序遇到多个具有相同节点 ID 的行，则使用以下规则将它们合并：
    - 行中的所有标签都将添加到节点中。
    - 对于每个属性，只加载其中一个属性值。选择要加载哪个属性值是不确定的。
  - 如果加载程序遇到多个具有相同关系 ID 的行，则只加载其中一行。选择要加载哪一行是不确定的。
  - 如果加载程序遇到具有现有节点或关系 ID 的加载数据，则它从不会更新数据库中现有节点或关系的属性值。但是，它的确会加载现有节点或关系中不存在的节点标签和属性。
- 尽管您不必为关系分配 ID，但这通常是个好主意（请参阅上面的 userProvidedEdgeIds 参数）。如果没有显式关系 ID，加载程序必须重新加载所有关系，以防关系文件出现错误，而不是从失败的地方恢复加载。

  此外，如果加载数据不包含显式关系 ID，则加载程序无法检测重复的关系。

以下是 openCypher 加载命令的示例：

```
curl -X POST https://your-neptune-endpoint:port/loader \
     -H 'Content-Type: application/json' \
     -d '
     {
       "source" : "s3://bucket-name/object-key-name",
       "format" : "opencypher",
       "userProvidedEdgeIds": "TRUE",
       "iamRoleArn" : "arn:aws:iam::account-id:role/role-name",
       "region" : "region",
       "failOnError" : "FALSE",
       "parallelism" : "MEDIUM",
```

```
    }'
```

加载程序响应与正常响应相同。例如：

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

Neptune 加载程序响应语法

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

200 OK (200 确定)

成功启动的加载任务将返回 200 代码。

Neptune 加载程序错误

当错误出现后，响应的 BODY 中将返回 JSON 对象。message 对象包含对错误的描述。

错误类别

- Error 400 – 语法错误将返回 HTTP 400 错误请求错误。此消息描述错误。
- Error 500 – 无法处理的有效请求返回 HTTP 500 内部服务器错误。此消息描述错误。

以下是来自加载程序的可能的错误消息，其中包含对错误的描述。

加载程序错误消息

- Couldn't find the AWS credential for iam_role_arn (HTTP 400)

  未找到凭证。对照 IAM 控制台或 AWS CLI 输出验证提供的证书。确保您已将在 iamRoleArn 中指定的 IAM 角色添加到集群。

- `S3 bucket not found for source` (HTTP 400)

  S3 存储桶不存在。请检查存储桶的名称。

- `The source source-uri does not exist/not reachable` (HTTP 400)

  在 S3 存储桶中未找到匹配的文件。

- `Unable to connect to S3 endpoint. Provided source = source-uri and region = aws-region` (HTTP 500)

  无法连接到 Amazon S3。区域必须与集群区域匹配。确保具有 VPC 端点。有关创建 VPC 端点的信息，请参阅 创建 Amazon S3 VPC 端点。

- `Bucket is not in provided Region (aws-region)` (HTTP 400)

  存储桶必须与您的 Neptune 数据库实例位于同一 AWS 区域。

- `Unable to perform S3 list operation` (HTTP 400)

  提供的 IAM 用户或角色对存储桶或文件夹无 `List` 权限。检查存储桶上的此策略或访问控制列表 (ACL)。

- `Start new load operation not permitted on a read replica instance` (HTTP 405)

  加载是写入操作。在读取/写入集群终端节点上重试加载。

- `Failed to start load because of unknown error from S3` (HTTP 500)

  Amazon S3 返回了未知错误。联系 AWS Support。

- `Invalid S3 access key` (HTTP 400)

  访问密钥无效。请检查提供的凭证。

- `Invalid S3 secret key` (HTTP 400)

  秘密密钥无效。请检查提供的凭证。

- `Max concurrent load limit breached` (HTTP 400)

  如果提交的加载请求没有附带 `"queueRequest" : "TRUE"`，并且当前正在运行加载作业，则请求将失败并显示此错误。

- `Failed to start new load for the source "source name". Max load task queue size limit breached. Limit is 64` (HTTP 400)

Neptune 支持一次对多达 64 个加载程序任务进行排队。如果额外的加载请求提交到已经包含 64 个作业的队列，请求将失败并显示此消息。

Neptune 加载程序示例

Example 请求

下面是使用 `curl` 命令经由 HTTP POST 发送的请求。它加载 Neptune CSV 格式的文件。有关更多信息，请参阅 Gremlin 加载数据格式。

```
curl -X POST \
    -H 'Content-Type: application/json' \
    https://your-neptune-endpoint:port/loader -d '
    {
      "source" : "s3://bucket-name/object-key-name",
      "format" : "csv",
      "iamRoleArn" : "ARN for the IAM role you are using",
      "region" : "region",
      "failOnError" : "FALSE",
      "parallelism" : "MEDIUM",
      "updateSingleCardinalityProperties" : "FALSE",
      "queueRequest" : "FALSE"
    }'
```

Example 响应

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

Neptune 加载程序获取状态 API

获取 `loader` 任务的状态。

要获取加载状态，必须将 HTTP GET 请求发送到 `https://your-neptune-endpoint:port/loader` 终端节点。要获取特定加载请求的状态，必须包含 `loadId` 作为 URL 参数，也可将 `loadId` 附加到 URL 路径。

Neptune 仅跟踪最近的 1024 个批量加载任务，并且仅存储每个任务的最后 10000 个错误详细信息。

有关加载程序在出现错误时返回的错误和馈送消息的列表，请参阅Neptune 加载程序错误和源消息。

目录

Neptune 加载程序获取状态请求

加载程序获取状态请求语法

```
GET https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
GET https://your-neptune-endpoint:port/loader/loadId
```

```
GET https://your-neptune-endpoint:port/loader
```

Neptune 加载程序获取状态请求参数

- **`loadId`** – 加载任务的 ID。如果未指定 `loadId`，则返回加载 ID 的列表。

- **details** – 包括整体状态之外的详细信息。

  允许的值：TRUE、FALSE。

  默认值：FALSE。
- **errors** – 包含错误的列表。

  允许的值：TRUE、FALSE。

  默认值：FALSE。

  将对错误的列表进行分页。page 和 errorsPerPage 参数允许浏览所有错误。
- **page** – 错误页码。仅当 errors 参数设置为 TRUE 时有效。

  允许的值：正整数。

  默认值：1。
- **errorsPerPage** – 每页的错误数量。仅当 errors 参数设置为 TRUE 时有效。

  允许的值：正整数。

  默认值：10。
- **limit** – 要列出的加载 ID 的数量。仅当通过发送 GET 请求且未指定 loadId 来请求加载 ID 的列表时有效。

  允许的值：从 1 到 100 的正整数。

  默认值：100。
- **includeQueuedLoads** – 一个可选参数，可用于在请求加载 ID 列表时排除排队的加载请求的加载 ID。

  > **Note**
  >
  > 此参数从 Neptune 引擎版本 1.0.3.0 开始推出。

  默认情况下，状态为 LOAD_IN_QUEUE 的所有加载作业的加载 ID 都包含在此类列表中。它们显示在其他作业的加载 ID 之前，按照将它们添加到队列的时间从最近到最早进行排序。

  允许的值：TRUE、FALSE。

默认值：TRUE。

Neptune 加载程序获取状态响应

Neptune 加载程序获取状态响应 JSON 布局

加载程序状态响应的总体布局如下：

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

```
            }
        ],
        "errors" : {
            "startIndex" : number,
            "endIndex" : number,
            "loadId" : "string,
            "errorLogs" : [ ]
        }
    }
}
```

Neptune 加载程序获取状态 **overallStatus** 和 **failedFeeds** 响应对象

为每个失败馈送返回的可能响应（包括错误描述）与 Get-Status 响应中的 overallStatus 对象相同。

以下字段显示在所有负载的 overallStatus 对象中，以及每个失败馈送的 failedFeeds 对象中：

- **fullUri** – 要加载的一个或多个文件的 URI。

  类型：字符串

  格式：s3://*bucket*/*key*。
- **runNumber** – 此加载或馈送的运行编号。此编号将在加载重新启动时增加。

  类型：无符号整数
- **retryNumber** – 此加载或馈送的重试编号。此编号将在加载程序自动重试馈送或加载时递增。

  类型：无符号整数
- **status** – 返回的加载状态或馈送状态。LOAD_COMPLETED 指示加载成功，无问题。有关其它加载状态消息的列表，请参阅Neptune 加载程序错误和源消息。

  类型：字符串。
- **totalTimeSpent** – 解析并插入以进行加载或馈送的数据所耗费的时间（秒）。这不包括提取源文件列表所耗费时间。

  类型：无符号整数
- **totalRecords** – 已加载或尝试加载的记录总数。

  类型：无符号整数

请注意，从 CSV 文件加载时，记录计数不是指加载的行数，而是指这些行中的单个记录数。例如，以一个小的 CSV 文件为例：

```
~id,~label,name,team
'P-1','Player','Stokes','England'
```

Neptune 会认为这个文件包含 3 条记录，即：

```
P-1  label Player
P-1  name  Stokes
P-1  team  England
```

- **totalDuplicates** – 遇到的重复记录的数量。

  类型：无符号整数

  与 totalRecords 计数一样，此值包含 CSV 文件中单个重复记录的数量，而不是重复行的数量。以这个小的 CSV 文件为例：

```
~id,~label,name,team
P-2,Player,Kohli,India
P-2,Player,Kohli,India
```

  加载此文件后返回的状态如下所示，报告总共 6 条记录，其中 3 条是重复的：

```
{
  "status": "200 OK",
  "payload": {
    "feedCount": [
      {
        "LOAD_COMPLETED": 1
      }
    ],
    "overallStatus": {
      "fullUri": "(the URI of the CSV file)",
      "runNumber": 1,
      "retryNumber": 0,
      "status": "LOAD_COMPLETED",
      "totalTimeSpent": 3,
      "startTime": 1662131463,
```

```
      "totalRecords": 6,
      "totalDuplicates": 3,
      "parsingErrors": 0,
      "datatypeMismatchErrors": 0,
      "insertErrors": 0
    }
  }
}
```

对于 openCypher 加载，在以下情况下会计算重复记录：

- 加载程序检测到节点文件中的某行具有的 ID 没有 ID 空间，而该 ID 与另一个没有 ID 空间的 ID 值相同，无论是在另一行中还是属于现有节点。
- 加载程序检测到节点文件中的某行具有的 ID 有 ID 空间，而该 ID 与另一个有 ID 空间的 ID 值相同，无论是在另一行中还是属于现有节点。

请参阅 加载 openCypher 数据的特殊注意事项。

- **parsingErrors** – 遇到的解析错误的数量。

  类型：无符号整数

- **datatypeMismatchErrors** – 数据类型与给定的数据不匹配的记录的数量。

  类型：无符号整数

- **insertErrors** – 由于错误而无法插入的记录的数量。

  类型：无符号整数

Neptune 加载程序获取状态 **errors** 响应对象

错误分为以下几类：

- **Error 400** - 无效的 `loadId` 返回 HTTP `400` 错误请求错误。此消息描述错误。
- **Error 500** – 无法处理的有效请求返回 HTTP `500` 内部服务器错误。此消息描述错误。

有关加载程序在出现错误时返回的错误和馈送消息的列表，请参阅Neptune 加载程序错误和源消息。

当发生错误时，会在响应的 `BODY` 中返回一个 JSON `errors` 对象，其中包含以下字段：

- **startIndex** – 包含的第一个错误的索引。

类型：无符号整数

- **endIndex** – 包含的最后一个错误的索引。

  类型：无符号整数

- **loadId** – 加载的 ID。可通过将 errors 参数设置为 TRUE 来使用此 ID 打印错误。

  类型：字符串。

- **errorLogs** – 错误列表。

  类型：列表

Neptune 加载程序获取状态 **errorLogs** 响应对象

加载程序获取状态响应中 errors 下的 errorLogs 对象包含一个使用以下字段描述每个错误的对象：

- **errorCode** – 识别错误的性质。

  它可能采用下列值之一：

  - PARSING_ERROR
  - S3_ACCESS_DENIED_ERROR
  - FROM_OR_TO_VERTEX_ARE_MISSING
  - ID_ASSIGNED_TO_MULTIPLE_EDGES
  - SINGLE_CARDINALITY_VIOLATION
  - FILE_MODIFICATION_OR_DELETION_ERROR
  - OUT_OF_MEMORY_ERROR
  - INTERNAL_ERROR（当批量加载程序无法确定错误的类型时返回）。

- **errorMessage** – 描述错误的消息。

  这可以是与错误代码关联的通用消息，也可以是包含详细信息的特定消息，例如关于缺少起始顶点/结束顶点或解析错误的信息。

- **fileName** – 馈送的名称。

- **recordNum** – 如果出现解析错误，则这是记录文件中无法解析的记录号。如果记录号不适用于错误，或者无法确定，则将其设置为零。

例如，如果批量加载程序在 RDF nquads 文件中遇到诸如以下内容的错误行，则会生成解析错误：

```
<http://base#subject> |http://base#predicate> <http://base#true> .
```

如您所见，上面一行中的第二个 http 应该在前面加上 < 而不是 |。状态响应中 errorLogs 下生成的错误对象如下所示：

```
{
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Expected '<', found: |",
    "fileName" : "s3://bucket/key",
    "recordNum" : 12345
},
```

Neptune 加载程序获取状态示例

加载状态请求示例

下面是使用 curl 命令经由 HTTP GET 发送的请求。

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)'
```

Example 响应

```
{
    "status" : "200 OK",
    "payload" : {
        "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
        }
    }
}
```

loadId 请求示例

下面是使用 curl 命令经由 HTTP GET 发送的请求。

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

Example 响应

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

详细状态请求示例

下面是使用 curl 命令经由 HTTP GET 发送的请求。

```
curl -X GET 'https://your-neptune-endpoint:port/loader/loadId (a UUID)?details=true'
```

Example 响应

```
{
    "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
        }
    }
}
```

Neptune 加载程序获取状态 **`errorLogs`** 示例

错误发生时的详细状态响应示例

这是使用 `curl` 通过 HTTP `GET` 发送的请求：

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

Example 属于出现错误时的详细响应

这是您可能从上面的查询中得到的响应的示例，其中 `errorLogs` 对象列出了遇到的加载错误：

```
{
    "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
        },
        "errors" : {
            "endIndex" : 3,
            "errorLogs" : [
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
                    "recordNum" : 1
                },
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
```

```
                    "recordNum" : 2
                },
                {
                    "errorCode" : "PARSING_ERROR",
                    "errorMessage" : "Expected '<', found: |",
                    "fileName" : "s3://bucket/key",
                    "recordNum" : 3
                }
            ],
            "loadId" : "0a237328-afd5-4574-a0bc-c29ce5f54802",
            "startIndex" : 1
        }
    }
}
```

Data prefetch task interrupted 错误示例

在某些情况下，当您收到 LOAD_FAILED 状态，然后请求详细信息时，返回的错误可能是 PARSING_ERROR 以及 Data prefetch task interrupted 消息，如下所示：

```
"errorLogs" : [
    {
        "errorCode" : "PARSING_ERROR",
        "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467
 failed",
        "fileName" : "s3://some-source-bucket/some-source-file",
        "recordNum" : 0
    }
]
```

在数据加载过程中发生临时中断时会出现此错误，这种中断一般不是由您的请求或数据导致的。这通常可以通过再次运行批量上传请求加以解决。如果您使用的是默认设置，即 "mode":"AUTO" 和 "failOnError":"TRUE"，则在发生中断时，加载程序会跳过已经成功加载的文件，并继续加载尚未加载的文件。

Neptune 加载程序取消任务

取消加载任务。

要取消任务，必须将 HTTP DELETE 请求发送到 https://*your-neptune-endpoint*:*port*/loader 终端节点。loadId 可以追加到 /loader URL 路径，或作为变量包含在 URL 中。

取消任务请求语法

```
DELETE https://your-neptune-endpoint:port/loader?loadId=loadId
```

```
DELETE https://your-neptune-endpoint:port/loader/loadId
```

取消任务请求参数

loadId

加载任务的 ID。

取消任务响应语法

```
no response body
```

200 OK (200 确定)

成功删除的加载任务将返回 200 代码。

取消任务错误

当错误出现后，响应的 BODY 中将返回 JSON 对象。message 对象包含对错误的描述。

错误类别

- **Error 400** - 无效的 loadId 返回 HTTP 400 错误请求错误。此消息描述错误。
- **Error 500** – 无法处理的有效请求返回 HTTP 500 内部服务器错误。此消息描述错误。

取消任务错误消息

以下是来自取消 API 的可能的错误消息，其中包含对错误的描述。

- The load with id = *load_id* does not exist or not active (HTTP 404) – 找不到加载。检查 id 参数的值。
- Load cancellation is not permitted on a read replica instance. (HTTP 405) – 加载是写入操作。在读取/写入集群终端节点上重试加载。

取消任务示例

Example 请求

下面是使用 curl 命令经由 HTTP DELETE 发送的请求。

```
curl -X DELETE 'https://your-neptune-endpoint:port/loader/0a237328-afd5-4574-a0bc-c29ce5f54802'
```

AWS Database Migration Service 用于将来自其他数据存储的数据加载到 Amazon Neptune

AWS Database Migration Service (AWS DMS) 可以快速安全地将支持的源数据库中的数据加载到 Neptune。源数据库在迁移过程中可保持完全正常运行，从而最大程度减少依赖于该数据库的应用程序停机时间。

您可以在《AWS Database Migration Service 用户指南》和《AWS Database Migration Service API 参考》 AWS DMS 中找到相关的详细信息。特别是，您可以在使用 Amazon Neptune 作为 AWS Database Migration Service的目标中了解如何将 Neptune 集群设置为迁移目标。

以下是使用 AWS DMS将数据导入 Neptune 中的一些先决条件：

- 您需要创建一个 AWS DMS 表映射对象来定义如何从源数据库中提取数据（有关详细信息，请参阅《 AWS DMS 用户指南》中的 “使用 JSON 通过表映射指定表选择和转换”）。此表映射配置对象指定应读取哪些表、按何种顺序读取以及如何命名它们的列。它还可以筛选正在复制的行，并提供简单的值转换，例如转换为小写或舍入。
- 您需要创建一个 Neptune GraphMappingConfig，以指定如何将从源数据库中提取的数据加载到 Neptune。对于 RDF 数据（使用 SPARQL 查询），GraphMappingConfig 使用 W3 的标准 R2RML 映射语言编写。对于属性图数据（使用 Gremlin 进行查询），GraphMappingConfig 是 JSON 对象，如GraphMappingConfig Property-Graph/Gremlin 数据的布局中所述。
- 您必须使用在 AWS DMS 与 Neptune 数据库集群相同的 VPC 中创建复制实例，以调解数据的传输。
- 您还需要一个 Amazon S3 桶来用作中间存储，以暂存迁移数据。

创建海王星 GraphMappingConfig

您创建的 GraphMappingConfig 指定如何将从源数据存储中提取的数据加载到 Neptune 数据库集群中。它的格式根据要加载 RDF 数据还是要加载属性图数据而有所不同。

对于 RDF 数据，您可以使用 W3 R2RML 语言，将关系数据映射到 RDF。

如果要加载将使用 Gremlin 查询的属性图数据，则为 GraphMappingConfig 创建 JSON 对象。

GraphMappingConfig RDF/SPARQL 数据的布局

如果您正在使用 SPARQL 加载要查询的 RDF 数据，您可以在 R2RML 中编写 GraphMappingConfig。R2RML 是一种标准 W3 语言，用于将关系数据映射到 RDF。示例如下：

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/ns#> .

<#TriplesMap1>
    rr:logicalTable [ rr:tableName "nodes" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{id}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:name;
        rr:objectMap [ rr:column "label" ];
    ] .
```

以下是另一个示例：

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix ex: <http://example.com/#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<#TriplesMap2>
    rr:logicalTable [ rr:tableName "Student" ];
    rr:subjectMap   [ rr:template "http://example.com/{ID}{Name}";
                      rr:class foaf:Person ];
    rr:predicateObjectMap [
        rr:predicate ex:id ;
        rr:objectMap  [ rr:column "ID";
                        rr:datatype xsd:integer ]
```

```
    ];
    rr:predicateObjectMap [
        rr:predicate foaf:name ;
        rr:objectMap  [ rr:column "Name" ]
    ] .
```

位于 R2RML：RDB 到 RDF 映射语言的 W3 推荐提供了语言的详细信息。

GraphMappingConfig Property-Graph/Gremlin 数据的布局

与属性图形数据相当的 GraphMappingConfig 是一个 JSON 对象，该对象为要从源数据生成的每个图形实体提供映射规则。以下模板显示了此对象中的各个规则的情况：

```
{
  "rules": [
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "vertex_definitions": [
        {
          "vertex_id_template": "{col1}",
          "vertex_label": "(the vertex to create)",
          "vertex_definition_id": "(an identifier for this vertex)",
          "vertex_properties": [
            {
              "property_name": "(name of the property)",
              "property_value_template": "{col2} or text",
              "property_value_type": "(data type of the property)"
            }
          ]
        }
      ]
    },
    {
      "rule_id": "(an identifier for this rule)",
      "rule_name": "(a name for this rule)",
      "table_name": "(the name of the table or view being loaded)",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{col1}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
```

```
          },
          "to_vertex": {
            "vertex_id_template": "{col3}",
            "vertex_definition_id": "(an identifier for the vertex referenced above)"
          },
          "edge_id_template": {
            "label": "(the edge label to add)",
            "template": "{col1}_{col3}"
          },
          "edge_properties":[
            {
              "property_name": "(the property to add)",
              "property_value_template": "{col4} or text",
              "property_value_type": "(data type like String, int, double)"
            }
          ]
        }
      ]
    }
  ]
}
```

请注意，顶点标签的存在意味着在这里创建了顶点，而没有顶点标签则意味着顶点由不同的源创建，而此定义仅添加顶点属性。

以下是员工记录的示例规则：

```
{
  "rules": [
    {
      "rule_id": "1",
      "rule_name": "vertex_mapping_rule_from_nodes",
      "table_name": "nodes",
      "vertex_definitions": [
        {
          "vertex_id_template": "{emp_id}",
          "vertex_label": "employee",
          "vertex_definition_id": "1",
          "vertex_properties": [
            {
              "property_name": "name",
              "property_value_template": "{emp_name}",
              "property_value_type": "String"
```

```
            }
          ]
        }
      ]
    },
    {
      "rule_id": "2",
      "rule_name": "edge_mapping_rule_from_emp",
      "table_name": "nodes",
      "edge_definitions": [
        {
          "from_vertex": {
            "vertex_id_template": "{emp_id}",
            "vertex_definition_id": "1"
          },
          "to_vertex": {
            "vertex_id_template": "{mgr_id}",
            "vertex_definition_id": "1"
          },
          "edge_id_template": {
            "label": "reportsTo",
            "template": "{emp_id}_{mgr_id}"
          },
          "edge_properties":[
            {
              "property_name": "team",
              "property_value_template": "{team}",
              "property_value_type": "String"
            }
          ]
        }
      ]
    }
  ]
}
```

创建以 Neptune 为目标的 AWS DMS 复制任务

创建表映射和图形映射配置后，使用以下过程将数据从源存储加载到 Neptune。有关相关 API 的更多详细信息，请参阅 AWS DMS 文档。

步骤 1：创建 AWS DMS 复制实例

在运行 Neptune 数据库集群的 VPC 中创建 AWS DMS 复制实例（请参阅《AWS DMS 用户指南》中的“使用 AWS DMS 复制CreateReplication实例和实例”）。你可以使用如下 AWS CLI 命令来做到这一点：

```
aws dms create-replication-instance \
    --replication-instance-identifier (the replication instance identifier) \
    --replication-instance-class (the size and capacity of the instance, like
 'dms.t2.medium') \
    --allocated-storage (the number of gigabytes to allocate for the instance
 initially) \
    --engine-version (the DMS engine version that the instance should use) \
    --vpc-security-group-ids (the security group to be used with the instance)
```

第 2 步。为源数据库创建 AWS DMS 终端节点

下一步是为您的源数据存储创建一个 AWS DMS 终端节点。你可以 AWS CLI 这样使用 AWS DMS CreateEndpointAPI：

```
aws dms create-endpoint \
    --endpoint-identifier (source endpoint identifier) \
    --endpoint-type source \
    --engine-name (name of source database engine) \
    --username (user name for database login) \
    --password (password for login) \
    --server-name (name of the server) \
    --port (port number) \
    --database-name (database name)
```

第 3 步。设置 Amazon S3 桶供 Neptune 用于暂存数据

如果您没有可用于暂存数据的 Amazon S3 桶，请按照以下说明创建一个桶：《Amazon S3 入门指南》中的创建桶，或《控制台用户指南》中的如何创建 S3 桶？。

如果您还没有创建用于授予对桶的 `GetObject`、`PutObject`、`DeleteObject` 和 `ListObject` 权限的 IAM policy，则需要创建此策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "ListObjectsInBucket",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::(bucket-name)"
      ]
    },
    {
      "Sid": "AllObjectActions",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:ListObject"
       ],
      "Resource": [
        "arn:aws:s3:::(bucket-name)/*"
      ]
    }
  ]
}
```

如果 Neptune 数据库集群启用了 IAM 身份验证，则还需要包括以下策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "neptune-db:*",
      "Resource": "(the ARN of your Neptune DB cluster resource)"
    }
  ]
}
```

创建 IAM 角色作为将策略附加到的可信文档：

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "dms.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Sid": "neptune",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

将策略附加到角色后，将该角色附加到您的 Neptune 数据库集群。这将 AWS DMS 允许使用存储桶暂存正在加载的数据。

第 4 步。在 Neptune VPC 中创建 Amazon S3 端点

现在，在您的 Neptune 集群所在的 VPC 中，为中间 Amazon S3 桶创建 VPC 网关端点。您可以使用 AWS Management Console 或 AWS CLI 来执行此操作，如创建网关终端节点中所述。

第 5 步。为 Neptune 创建 AWS DMS 目标端点

为您的目标 Neptune 数据库集群创建 AWS DMS 终端节点。你可以将 AWS DMS CreateEndpointAPI 与`NeptuneSettings`参数配合使用，如下所示：

```
aws dms create-endpoint \
    --endpoint-identifier (target endpoint identifier) \
    --endpoint-type target \
    --engine-name neptune \
    --server-name (name of the server) \
    --port (port number) \
    --neptune-settings '{ \
      "ServiceAccessRoleArn": "(ARN of the service access role)", \
```

```
      "S3BucketName": "(name of S3 bucket to use for staging files when migrating)", \
      "S3BucketFolder": "(name of the folder to use in that S3 bucket)", \
      "ErrorRetryDuration": (number of milliseconds to wait between bulk-load retries),
\
      "MaxRetryCount": (the maximum number of times to retry a failing bulk-load job),
\
      "MaxFileSize": (maximum file size, in bytes, of the staging files written to S3),
\
      "isIamAuthEnabled": (set to true if IAM authentication is enabled on the Neptune
cluster) }'
```

在NeptuneSettings参数中传递给 AWS DMS CreateEndpoint API 的 JSON 对象具有以下字段：

- **ServiceAccessRoleArn** –（必需）IAM 角色的 ARN，该角色允许对用于暂存迁移到 Neptune 的数据的 S3 桶进行精细访问。如果对 Neptune 数据库集群启用了 IAM 授权，则此角色还应有权访问该数据库集群。
- **S3BucketName** –（必需）对于完全加载迁移，复制实例会将所有 RDS 数据转换为 CSV、四元组文件并将其上传到 S3 中的此暂存桶，然后将其批量加载到 Neptune。
- **S3BucketFolder** –（必需）要在 S3 暂存桶中使用的文件夹。
- **ErrorRetryDuration** –（可选）Neptune 请求失败后在发出重试请求之前等待的毫秒数。默认值是 250。
- **MaxRetryCount**—（可选）在可重试失败后 AWS DMS 应发出的最大重试请求数。默认值为 5。
- **MaxFileSize** –（可选）在迁移期间，保存到 S3 的各个暂存文件的最大大小（以字节为单位）。默认值是 1048576 KB（1 GB）。
- **IsIAMAuthEnabled** –（可选）如果在 Neptune 数据库集群上启用了 IAM 身份验证，则设置为 `true`，否则设置为 `false`。默认值为 `false`。

第 6 步。测试与新终端节点的连接

你可以使用 AWS DMS TestConnectionAPI 测试与每个新端点的连接，如下所示：

```
aws dms test-connection \
    --replication-instance-arn (the ARN of the replication instance) \
    --endpoint-arn (the ARN of the endpoint you are testing)
```

第 7 步。创建 AWS DMS 复制任务

成功完成上述步骤后，使用任务 API 创建用于将数据从源数据存储迁移到 Neptune 的复制 AWS DMS CreateReplication任务，如下所示：

```
aws dms create-replication-task \
    --replication-task-identifier (name for the replication task) \
    --source-endpoint-arn (ARN of the source endpoint) \
    --target-endpoint-arn (ARN of the target endpoint) \
    --replication-instance-arn (ARN of the replication instance) \
    --migration-type full-load \
    --table-mappings (table-mapping JSON object or URI like 'file:///tmp/table-
mappings,json') \
    --task-data (a GraphMappingConfig object or URI like 'file:///tmp/graph-mapping-
config.json')
```

`TaskData` 参数提供 GraphMappingConfig，用于指定所复制的数据应如何存储在 Neptune 中。

步骤 8：启动 AWS DMS 复制任务

现在，您可以启动复制任务：

```
aws dms start-replication-task
    --replication-task-arn (ARN of the replication task started in the previous step)
    --start-replication-task-type start-replication
```

查询 Neptune 图形

Neptune 支持使用以下图形查询语言来访问图形：

- Gremlin，由 Apache 定义， TinkerPop用于创建和查询属性图。

  Gremlin 中的查询是由离散步骤组成的遍历，每个步骤都沿着一个边缘到达一个节点。

  请参阅使用 Gremlin 访问 Neptune 图形以了解如何在 Neptune 中使用 Gremlin，并参阅Amazon Neptune 中的 Gremlin 标准合规性以查看有关 Gremlin 的 Neptune 实现的具体细节。
- openCypher 是一种用于属性图的声明式查询语言，最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。它的语法在 openCypher 规范中介绍。
- SPARQL 是一种基于图形模式匹配的声明性语言，用于查询 RDF 数据。它得到万维网联盟的支持。

  请参阅使用 SPARQL 访问 Neptune 图形以了解如何在 Neptune 中使用 SPARQL，并参阅Amazon Neptune 中的 SPARQL 标准合规性以查看有关 SPARQL 的 Neptune 实现的具体细节。

> **Note**
>
> Gremlin 和 openCypher 都可以用来查询存储在 Neptune 中的任何属性图数据，无论这些数据是如何加载的。

主题

Amazon Neptune 中的查询排队

在开发和调整图形应用程序时，了解数据库对查询进行排队的方式有何影响可能会有帮助。在 Amazon Neptune 中，查询按以下方式排队：

- 无论实例大小如何，每个实例可排队的最大查询数为 8192。超过该数字之后，系统将拒绝任意查询，查询失败并显示 `ThrottlingException`。

- 一次可以执行的最大查询数由分配的工作线程数决定，该数量通常设置为可用虚拟 CPU 核心 (vCPU) 数的两倍。
- 查询延迟包括查询在队列中花费的时间、往返行程时间以及实际执行所用的时间。

确定在给定时刻队列中有多少查询

该`MainRequestQueuePendingRequests` CloudWatch 指标以五分钟为粒度记录输入队列中等待的请求数（请参阅Neptun CloudWatch e 指标）。

对于 Gremlin，您可以使用 Gremlin 查询状态 API 返回的 `acceptedQueryCount` 值获取队列中的当前查询计数。但请注意，SPARQL 查询状态 API 返回的 `acceptedQueryCount` 值包括自服务器启动以来接受的所有查询，已完成的查询也计算在内。

查询排队如何影响超时

如上所述，查询延迟包括查询在队列中花费的时间以及执行所用的时间。

由于查询超时时段的测量通常是从进入队列时开始，因此进展缓慢的队列会导致许多查询一出队就超时。这显然是不可取的，所以除非查询可以快速执行，否则请尽量避免大量的查询排队。

使用 Gremlin 访问 Neptune 图形

亚马逊 Neptune 与 Apache TinkerPop 3 和 Gremlin 兼容。这意味着您可以连接到 Neptune 数据库实例并使用 Gremlin 遍历语言来查询图表（参见 Apache 3 文档中的图表）。 TinkerPop有关 Gremlin 的 Neptune 实施的差异，请参阅Gremlin 标准合规性。

不同的 Neptune 引擎版本支持不同的 Gremlin 版本。查看您正在运行的 Neptune 版本的引擎版本页面，以确定它支持哪个 Gremlin 版本。

Gremlin 中的遍历 是一系列连环步骤。它开始于顶点（或边缘）。它通过依次按照每个顶点的传出边缘，然后按接下来这些顶点的传出边缘来遍历图形。每个步骤都是遍历中的一个操作。有关更多信息，请参阅 TinkerPop 3 文档中的遍历。

Gremlin 语言有多个变体，并支持采用多种编程语言进行 Gremlin 访问。有关更多信息，请参阅 TinkerPop 3 文档中的 “关于 Gremlin 语言变体”。

本文档介绍如何使用以下变体和编程语言访问 Neptune。

如传输中加密：使用 SSL/HTTPS 连接到 Neptune中所讨论，在所有 AWS 区域中连接到 Neptune 时，必须使用传输层安全性协议/安全套接字层 (TLS/SSL)。

Gremlin-Groovy

此部分中的 Gremlin 控制台和 HTTP REST 示例使用 Gremlin-Groovy 变体。有关 Gremlin 控制台和 Amazon Neptune 的更多信息，请参阅快速入门的the section called “使用 Gremlin”部分。

Gremlin-Java

Java 示例是使用官方 TinkerPop 3 Java 实现编写的，并使用 Gremlin-Java 变体。

Gremlin-Python

Python 示例是使用官方 TinkerPop 3 Python 实现编写的，使用 Gremlin-Python 变体。

以下部分将演示如何使用 Gremlin 控制台、基于 HTTPS 的 REST 及各种编程语言来连接到 Neptune 数据库实例。

在开始之前，您必须具有以下内容：

- 一个 Neptune 数据库实例。有关创建 Neptune 数据库实例的信息，请参阅创建新的 Neptune 数据库集群。
- 与 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例。

有关将数据加载到 Neptune 中的更多信息，包括先决条件、加载格式和加载参数，请参阅将数据加载到 Amazon Neptune 中。

主题

- 设置 Gremlin 控制台以连接到 Neptune 数据库实例
- 使用 HTTPS REST 端点连接到 Neptune 数据库实例
- 要与 Amazon Neptune 结合使用的基于 Java 的 Gremlin 客户端
- 使用 Python 连接到 Neptune 数据库实例
- 使用 .NET 连接到 Neptune 数据库实例
- 使用 Node.js 连接到 Neptune 数据库实例
- 使用 Go 连接到 Neptune 数据库实例
- Gremlin 查询提示
- Gremlin 查询状态 API
- Gremlin 查询取消

- 对基于 Gremlin 脚本的会话的支持
- Neptune 中的 Gremlin 事务
- 将 Gremlin API 与 Amazon Neptune 结合使用
- 在 Amazon Neptune Gremlin 中缓存查询结果
- 使用 Gremlin mergeV() 和 mergeE() 步骤进行高效的更新插入
- 使用 fold()/coalesce()/unfold() 进行高效的 Gremlin 更新插入
- 使用 Gremlin explain 分析 Neptune 查询执行
- 在 Neptune DFE 查询引擎中使用 Gremlin

设置 Gremlin 控制台以连接到 Neptune 数据库实例

Gremlin 控制台允许您在 REPL（read-eval-print 循环）环境中尝试 TinkerPop 图形和查询。

安装 Gremlin 控制台并以常规方式连接到控制台

您可以使用 Gremlin 控制台连接到远程图形数据库。以下部分将引导您安装和配置 Gremlin 控制台以远程连接到 Neptune 数据库实例。必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

有关使用 SSL/TLS（这是必需的）连接到 Neptune 的帮助，请参阅SSL/TLS 配置。

> **Note**
>
> 如果您在 Neptune 数据库集群上启用了 IAM 身份验证，请按照使用 Gremlin 控制台和签名版本 4 签名连接到 Neptune中的说明安装 Gremlin 控制台，而不是按照此处的说明进行操作。

安装 Gremlin 控制台并连接到 Neptune

1. Gremlin 控制台二进制文件需要 Java 8 或 Java 11。这些说明假设使用的是 Java 11。您可以按如下方式在 EC2 实例上安装 Java 11：

   - 如果您使用的是 Amazon Linux 2 (AL2)：

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 如果您使用的是 Amazon Linux 2023 (AL2023)：

```
sudo yum install java-11-amazon-corretto-devel
```

- 对于其它发行版，请使用以下适当的发行版：

```
sudo yum install java-11-openjdk-devel
```

或者：

```
sudo apt-get install openjdk-11-jdk
```

2. 输入以下命令以在 EC2 实例上将 Java 11 设置为默认运行时系统。

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

在系统提示时，输入 Java 11 的版本号。

3. 从 Apache 网站下载相应版本的 Gremlin 控制台。您可以在引擎版本页面上查看当前正在运行的 Neptune 引擎版本，以确定它支持哪个 Gremlin 版本。例如，对于版本 3.6.5，您可以将 Gremlin 控制台从 Apache Tinkerpop3 网站下载到您的 EC2 实例上，如下所示：

```
wget https://archive.apache.org/dist/tinkerpop/3.6.5/apache-tinkerpop-gremlin-
console-3.6.5-bin.zip
```

4. 解压缩 Gremlin 控制台 zip 文件。

```
unzip apache-tinkerpop-gremlin-console-3.6.5-bin.zip
```

5. 将目录更改为解压缩的目录。

```
cd apache-tinkerpop-gremlin-console-3.6.5
```

6. 在提取的目录的 `conf` 子目录中，创建名为 `neptune-remote.yaml` 的包含以下文本的文件。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的主机名或 IP 地址。方括号 (`[ ]`) 是必需的。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点部分。

```
hosts: [your-neptune-endpoint]
port: 8182
connectionPool: { enableSsl: true }
serializer: { className:
 org.apache.tinkerpop.gremlin.util.ser.GraphBinaryMessageSerializerV1,
              config: { serializeResultToString: true }}
```

> **Note**
>
> 在 3.7.0 版本中，序列化器已从`gremlin-driver`模块移至新`gremlin-util`模块。软件包从 org.apache.tinkerpop.gremlin.driver.ser 更改为 org.apache.tinkerpop.gremlin.util.ser。

7. 在终端中，导航到 Gremlin 控制台目录 (`apache-tinkerpop-gremlin-console-3.6.5`)，然后输入以下命令来运行 Gremlin 控制台。

```
bin/gremlin.sh
```

您应看到以下输出：

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

您现在位于 `gremlin>` 提示符处。将在此提示符处输入剩余步骤。

8. 在 `gremlin>` 提示符处，输入以下命令以连接到 Neptune 数据库实例。

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

9. 在 `gremlin>` 提示符处，输入以下命令以切换到远程模式。这会将所有 Gremlin 查询发送到远程连接。

```
:remote console
```

10. 输入以下命令以将查询发送到 Gremlin 图形。

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

11. 完成后，输入以下命令以退出 Gremlin 控制台。

```
:exit
```

> **Note**
>
> 使用分号 (`;`) 或换行符 (\n) 分隔每个语句。
> 在最后遍历之前的每个遍历必须以要执行的 `next()` 结尾。仅返回最后遍历中的数据。

有关 Gremlin 的 Neptune 实现的更多信息，请参阅the section called “Gremlin 标准合规性”。

连接到 Gremlin 主机的备选方式

普通连接方法的缺点

连接到 Gremlin 控制台的最常见方法是上面解释的那种，在 `gremlin>` 提示符下使用如下命令：

```
gremlin> :remote connect tinkerpop.server conf/(file name).yaml
gremlin> :remote console
```

这很好用，可以让您向 Neptune 发送查询。但是，它将 Groovy 脚本引擎排除在循环之外，因此 Neptune 将所有查询都视为纯粹的 Gremlin。这意味着以下查询格式会失败：

```
gremlin> 1 + 1
gremlin> x = g.V().count()
```

当以这种方式连接时，最接近使用变量的方法是使用控制台维护的 `result` 变量，并使用 `:>` 发送查询，如下所示：

```
gremlin> :remote console
==>All scripts will now be evaluated locally - type ':remote console' to return
 to remote mode for Gremlin Server - [krl-1-cluster.cluster-ro-cm9t6tfwbtsr.us-
east-1.neptune.amazonaws.com/172.31.19.217:8182]
```

```
gremlin> :> g.V().count()
==>4249

gremlin> println(result)
[result{object=4249 class=java.lang.Long}]

gremlin> println(result['object'])
[4249]
```

另一种连接方式

您也可以用不同的方式连接到 Gremlin 控制台，您可能会觉得这样更好，比如这样：

```
gremlin> g = traversal().withRemote('conf/neptune.properties')
```

此处，`neptune.properties` 采用以下形式：

```
gremlin.remote.remoteConnectionClass=org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteCon
gremlin.remote.driver.clusterFile=conf/my-cluster.yaml
gremlin.remote.driver.sourceName=g
```

`my-cluster.yaml` 文件应如下所示：

```
hosts: [my-cluster-abcdefghijk.us-east-1.neptune.amazonaws.com]
port: 8182
serializer: { className:
 org.apache.tinkerpop.gremlin.util.ser.GraphBinaryMessageSerializerV1,
              config: { serializeResultToString: false } }
connectionPool: { enableSsl: true }
```

> **Note**
>
> 在 3.7.0 版本中，序列化器已从`gremlin-driver`模块移至新`gremlin-util`模块。软件包从 org.apache.tinkerpop.gremlin.driver.ser 更改为 org.apache.tinkerpop.gremlin.util.ser。

通过这样配置 Gremlin 控制台连接，可以成功进行以下类型的查询：

```
gremlin> 1+1
==>2

gremlin> x=g.V().count().next()
==>4249

gremlin> println("The answer was ${x}")
The answer was 4249
```

您可以避免显示结果，如下所示：

```
gremlin> x=g.V().count().next();[]
gremlin> println(x)
4249
```

所有常用的查询方式（不包括最终步骤）继续起作用。例如：

```
gremlin> g.V().count()
==>4249
```

您甚至可以使用 `g.io().read()` 步骤通过这种连接加载文件。

使用 HTTPS REST 端点连接到 Neptune 数据库实例

Amazon Neptune 为 Gremlin 查询提供 HTTPS 端点。REST 接口与您的数据库集群使用的任何 Gremlin 版本兼容（请参阅您正在运行的 Neptune 引擎版本的引擎版本页面，以确定它支持哪个 Gremlin 版本）。

> **Note**
>
> 如传输中加密：使用 SSL/HTTPS 连接到 Neptune中所述，Neptune 现在要求您使用 HTTPS 而不是 HTTP 进行连接。

以下说明将带您演练使用 `curl` 命令和 HTTPS 连接到 Gremlin 终端节点。必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

针对 Neptune 数据库实例的 Gremlin 查询的 HTTPS 端点为 `https://`*`your-neptune-endpoint`*`:`*`port`*`/gremlin`。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点。

使用 HTTP REST 端点连接到 Neptune

以下示例使用 curl 来通过 HTTP POST 提交 Gremlin 查询。该查询采用 JSON 格式在 POST 请求正文中提交为 `gremlin` 属性。

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

此示例通过使用 `g.V().limit(1)` 遍历返回图形中的第一个顶点。您可以通过将它替换为另一个 Gremlin 遍历来查询其它内容。

> **Important**
>
> 默认情况下，REST 端点在单个 JSON 结果集中返回所有结果。如果此结果集太大，Neptune 数据库实例可能会出现 `OutOfMemoryError` 异常。
> 您可以通过启用分块响应（在一系列单独的响应中返回的结果）来避免这种情况。请参阅 使用可选的 HTTP 尾随标头启用由多部分组成的 Gremlin 响应。

尽管建议在发送 Gremlin 查询时使用 HTTP POST 请求，但也可以使用 HTTP GET 请求：

```
curl -G "https://your-neptune-endpoint:port?gremlin=g.V().count()"
```

> **Note**
>
> Neptune 不支持 `bindings` 属性。

使用可选的 HTTP 尾随标头启用由多部分组成的 Gremlin 响应

默认情况下，对 Gremlin 查询的 HTTP 响应以单个 JSON 结果集返回。如果结果集非常大，这可能会导致数据库实例出现 `OutOfMemoryError` 异常。

但是，您可以启用分块响应（以多个单独部分返回的响应）。您可以通过在请求中包含传输编码 (TE) 尾随标头 (`te: trailers`) 来实现此目的。有关 TE 标头的更多信息，请参阅有关 TE 请求标头的 MDN 页面。

当响应分成多个部分返回时，可能很难诊断在收到第一部分之后出现的问题，因为第一部分到达时的 HTTP 状态代码为 `200` (OK)。随后的失败通常会导致消息正文包含损坏的响应，Neptune 会在消息正文末尾附加一条错误消息。

为了便于检测和诊断此类故障，Neptune 还在每个响应块的尾随标头中加入了两个新的标头字段：

- `X-Neptune-Status` – 包含响应代码后跟一个短名称。例如，如果成功，则尾随标头将是：`X-Neptune-Status: 200 OK`。如果出现故障，响应代码将是 Neptune 引擎错误代码之一，例如 `X-Neptune-Status: 500 TimeLimitExceededException`。
- `X-Neptune-Detail` – 对于成功的请求，为空。如果出现错误，则它包含 JSON 错误消息。由于 HTTP 标头值中只允许使用 ASCII 字符，因此 JSON 字符串是经过 URL 编码的。

> **Note**
>
> Neptune 目前不支持对分块响应进行 `gzip` 压缩。如果客户端同时请求分块编码和压缩，Neptune 会跳过压缩。

要与 Amazon Neptune 结合使用的基于 Java 的 Gremlin 客户端

你可以在 Amazon Neptune 上使用两个基于 Java 的开源 Gremlin 客户端：A pache TinkerPop Java Gremlin 客户端，或者亚马逊 Neptune 的 Gremlin 客户端。

Apache TinkerPop Java Gremlin 客户端

如果可以，请始终使用您的引擎版本支持的最新版本的 Apache TinkerPop Java Gremlin 客户端。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。

下表列出了不同 Neptune 引擎版本支持的最早和最新版本的 TinkerPop 客户端：

Neptune 引擎版本	最低 TinkerPop 版本	最大 TinkerPop 版本
`1.3.2.0`	`3.6.2`	`3.7.1`

Neptune 引擎版本	最低 TinkerPop 版本	最大 TinkerPop 版本
1.3.1.0	3.6.2	3.6.5
1.3.0.0	3.6.2	3.6.4
1.2.1.1	3.6.2	3.6.2
1.2.1.0	3.6.2	3.6.2
1.2.0.2	3.5.2	3.5.6
1.2.0.1	3.5.2	3.5.6
1.2.0.0	3.5.2	3.5.6
1.1.1.0	3.5.2	3.5.6
1.1.0.0	3.4.0	3.4.13
1.0.5.1 和更早	（已弃用）	（已弃用）

TinkerPop 客户机通常在一个系列中向后兼容（例如3.3.x，或3.4.x）。在某些特殊情况下，必须打破向后兼容性，因此最好在TinkerPop升级到新的客户端版本之前查看升级建议。

客户端可能无法使用在比服务器支持的版本更高的版本中引入的新步骤或新特征，但除非升级建议要求进行重大更改，否则您可以预期现有查询和特征正常工作。

> **Note**
>
> 从 Nep tune 引擎版本 1.1.1.0 开始，请不要使用低于 Neptune 的 TinkerPop 版本。3.5.2 Python 用户应避免使用 TinkerPop 版本，3.4.9因为默认的超时设置需要直接配置（参见 TINKERPOP-2505）。

适用于 Amazon Neptune 的 Gremlin Java 客户端

亚马逊 Neptune 的 Gremlin 客户端是一款基于 Java 的开源 Gremlin 客户端，可以直接替代标准 Java 客户端。 TinkerPop

Neptune Gremlin 客户端针对 Neptune 集群进行了优化。它允许您管理集群中多个实例之间的流量分布，并在添加或移除副本时适应集群拓扑的变化。您甚至可以将客户端配置为根据角色、实例类型、可用区 (AZ) 或与实例关联的标签，将请求分发到集群中的实例子集。

最新版本的 Neptune Gremlin Java 客户端已在 Maven Central 上推出。

有关 Neptune Gremlin Java 客户端的更多信息，请参阅此博客文章。有关代码示例和演示，请查看客户的 GitHub 项目。

使用 Java 客户端连接到 Neptune 数据库实例

以下部分将引导您完成一个完整的 Java 示例的运行，该示例连接到 Neptune 数据库实例并使用 Apache Gremlin 客户端执行 Gremlin 遍历。 TinkerPop

必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

使用 Java 连接到 Neptune

1. 在您的 EC2 实例上安装 Apache Maven。首先，输入以下命令以添加具有 Maven 程序包的存储库：

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   输入以下命令以设置该程序包的版本号：

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

   然后，使用 yum 安装 Maven：

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

2. 安装 Java。Gremlin 库需要 Java 8 或 11。您可以按如下方式安装 Java 11：

   - 如果您使用的是 Amazon Linux 2 (AL2)：

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - 如果您使用的是 Amazon Linux 2023 (AL2023)：

```
sudo yum install java-11-amazon-corretto-devel
```

- 对于其它发行版，请使用以下适当的发行版：

```
sudo yum install java-11-openjdk-devel
```

或者：

```
sudo apt-get install openjdk-11-jdk
```

3. 将 Java 11 设置为 EC2 实例上的默认运行时系统：输入以下内容将 Java 8 设置为 EC2 实例上的默认运行时系统：

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

在系统提示时，输入 Java 11 的版本号。

4. 创建名为 **gremlinjava** 的新目录：

```
mkdir gremlinjava
cd gremlinjava
```

5. 在 gremlinjava 目录中，创建 pom.xml 文件，然后在文本编辑器中打开它：

```
nano pom.xml
```

6. 将以下内容复制到 pom.xml 文件中并保存它：

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://
maven.apache.org/maven-v4_0_0.xsd">
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.amazonaws</groupId>
  <artifactId>GremlinExample</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>GremlinExample</name>
```

```
  <url>https://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.apache.tinkerpop</groupId>
      <artifactId>gremlin-driver</artifactId>
      <version>3.6.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.tinkerpop/gremlin-groovy
      (Not needed for TinkerPop version 3.5.2 and up)
    <dependency>
      <groupId>org.apache.tinkerpop</groupId>
      <artifactId>gremlin-groovy</artifactId>
      <version>3.6.5</version>
    </dependency> -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-jdk14</artifactId>
      <version>1.7.25</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <source>11</source>
          <target>11</target>
        </configuration>
      </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.3</version>
          <configuration>
            <executable>java</executable>
            <arguments>
              <argument>-classpath</argument>
              <classpath/>
              <argument>com.amazonaws.App</argument>
            </arguments>
            <mainClass>com.amazonaws.App</mainClass>
            <complianceLevel>1.11</complianceLevel>
```

```
            <killAfter>-1</killAfter>
          </configuration>
        </plugin>
    </plugins>
  </build>
</project>
```

> **Note**
>
> 如果您要修改现有的 Maven 项目，则所需依赖项将在上述代码中突出显示。

7. 通过在命令行中键入以下命令来为示例源代码创建子目录 (`src/main/java/com/amazonaws/`)：

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

8. 在 `src/main/java/com/amazonaws/` 目录中，创建名为 `App.java` 的文件，然后在文本编辑器中打开它。

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

9. 将以下内容复制到 `App.java` 文件中。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的地址。请不要在 `addContactPoint` 方法中包括 `https://` 前缀。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点。

```
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 static
 org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.traversal;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.structure.T;
```

```
public class App
{
  public static void main( String[] args )
  {
    Cluster.Builder builder = Cluster.build();
    builder.addContactPoint("your-neptune-endpoint");
    builder.port(8182);
    builder.enableSsl(true);

    Cluster cluster = builder.create();

    GraphTraversalSource g =
 traversal().withRemote(DriverRemoteConnection.using(cluster));

    // Add a vertex.
    // Note that a Gremlin terminal step, e.g. iterate(), is required to make a
 request to the remote server.
    // The full list of Gremlin terminal steps is at https://tinkerpop.apache.org/
docs/current/reference/#terminal-steps
    g.addV("Person").property("Name", "Justin").iterate();

    // Add a vertex with a user-supplied ID.
    g.addV("Custom Label").property(T.id, "CustomId1").property("name", "Custom id
 vertex 1").iterate();
    g.addV("Custom Label").property(T.id, "CustomId2").property("name", "Custom id
 vertex 2").iterate();

    g.addE("Edge Label").from(__.V("CustomId1")).to(__.V("CustomId2")).iterate();

    // This gets the vertices, only.
    GraphTraversal t = g.V().limit(3).elementMap();

    t.forEachRemaining(
      e ->  System.out.println(t.toList())
    );

    cluster.close();
  }
}
```

有关使用 SSL/TLS（这是必需的）连接到 Neptune 的帮助，请参阅SSL/TLS 配置。

10. 使用以下 Maven 命令编译并运行示例：

```
mvn compile exec:exec
```

上述示例通过使用 `g.V().limit(3).elementMap()` 遍历返回图形中前两个顶点的每个属性的键和值的映射。要查询其他内容，请将其替换为具有其中一种适当的结尾方法的其他 Gremlin 遍历。

> **Note**
>
> 要将遍历提交到服务器进行评估，需要 Gremlin 查询的最后一部分 `.toList()`。如果您未包含该方法或其它等效方法，该查询将不会提交到 Neptune 数据库实例。
> 在添加顶点或边缘时，您还必须附加适当的结尾，例如当您使用 `addV( )` 步骤时。

以下方法将查询提交到 Neptune 数据库实例：

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

Gremlin Java 客户端的 SSL/TLS 配置

Neptune 要求默认情况下启用 SSL/TLS。通常，如果使用 `enableSsl(true)` 配置 Java 驱动程序，则它可以连接到 Neptune，而无需使用证书的本地副本设置 `trustStore()` 或 `keyStore()`。早期版本 TinkerPop 鼓励使用`keyCertChainFile()`来配置本地存储的`.pem`文件，但该版本已被弃用，在 3.5.x 之后不再可用。如果您通过 `SFSRootCAG2.pem` 将该设置与公有证书结合使用，则现在可以删除本地副本。

但是，如果您连接的实例没有用来验证公有证书的互联网连接，或者您使用的证书不是公有证书，则可以采取以下步骤来配置本地证书副本：

设置本地证书副本以启用 SSL/TLS

1. 从 Oracle 下载并安装 keytool。这将使设置本地密钥存储变得容易得多。
2. 下载 `SFSRootCAG2.pem` CA 证书（Gremlin Java SDK 需要证书来验证远程证书）：

```
wget https://www.amazontrust.com/repository/SFSRootCAG2.pem
```

3. 以 JKS 或 PKCS12 格式创建密钥存储。此示例使用 JKS。根据提示回答接下来的问题。稍后将需要您在此处创建的密码：

```
keytool -genkey -alias (host name) -keyalg RSA -keystore server.jks
```

4. 将您下载 SFSRootCAG2.pem 的文件导入到新创建的密钥存储中：

```
keytool -import -keystore server.jks -file .pem
```

5. 以编程方式配置 Cluster 对象：

```
Cluster cluster = Cluster.build("(your neptune endpoint)")
                         .port(8182)
                         .enableSSL(true)
                         .keyStore('server.jks')
                         .keyStorePassword("(the password from step 2)")
                         .create();
```

如果您愿意，您可以在配置文件中做同样的事情，就像您在 Gremlin 控制台上所做的那样：

```
hosts: [(your neptune endpoint)]
port: 8182
connectionPool: { enableSsl: true, keyStore: server.jks, keyStorePassword: (the
 password from step 2) }
serializer: { className:
 org.apache.tinkerpop.gremlin.driver.ser.GraphBinaryMessageSerializerV1, config:
 { serializeResultToString: true }}
```

使用重新连接逻辑连接到 Neptune 数据库实例的 Java 示例

以下 Java 示例演示如何使用重新连接逻辑连接到 Gremlin 客户端，以便从意外断开连接中恢复。

它具有以下依赖项：

```
<dependency>
    <groupId>org.apache.tinkerpop</groupId>
    <artifactId>gremlin-driver</artifactId>
```

```
    <version>${gremlin.version}</version>
</dependency>

<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>amazon-neptune-sigv4-signer</artifactId>
    <version>${sig4.signer.version}</version>
</dependency>

<dependency>
    <groupId>com.evanlennick</groupId>
    <artifactId>retry4j</artifactId>
    <version>0.15.0</version>
</dependency>
```

以下是示例代码：

```
public static void main(String args[]) {
  boolean useIam = true;

  // Create Gremlin cluster and traversal source
  Cluster.Builder builder = Cluster.build()
         .addContactPoint(System.getenv("neptuneEndpoint"))
         .port(Integer.parseInt(System.getenv("neptunePort")))
         .enableSsl(true)
         .minConnectionPoolSize(1)
         .maxConnectionPoolSize(1)
         .serializer(Serializers.GRAPHBINARY_V1D0)
         .reconnectInterval(2000);

  if (useIam) {
      builder.handshakeInterceptor( r -> {
         try {
            NeptuneNettyHttpSigV4Signer sigV4Signer =
                        new NeptuneNettyHttpSigV4Signer("(your region)", new
 DefaultAWSCredentialsProviderChain());
            sigV4Signer.signRequest(r);
         } catch (NeptuneSigV4SignerException e) {
            throw new RuntimeException("Exception occurred while signing the request",
 e);
         }
         return r;
      });
```

```
    }

  Cluster cluster = builder.create();

  GraphTraversalSource g = AnonymousTraversalSource
      .traversal()
      .withRemote(DriverRemoteConnection.using(cluster));

  // Configure retries
  RetryConfig retryConfig = new RetryConfigBuilder()
      .retryOnCustomExceptionLogic(getRetryLogic())
      .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
      .withMaxNumberOfTries(5)
      .withFixedBackoff()
      .build();

  @SuppressWarnings("unchecked")
  CallExecutor<Object> retryExecutor = new CallExecutorBuilder<Object>()
      .config(retryConfig)
      .build();

  // Do lots of queries
  for (int i = 0; i < 100; i++){
    String id = String.valueOf(i);

    @SuppressWarnings("unchecked")
    Callable<Object> query = () -> g.V(id)
        .fold()
        .coalesce(
            unfold(),
            addV("Person").property(T.id, id))
        .id().next();

    // Retry query
    // If there are connection failures, the Java Gremlin client will automatically
    // attempt to reconnect in the background, so all we have to do is wait and retry.
    Status<Object> status = retryExecutor.execute(query);

    System.out.println(status.getResult().toString());
  }

  cluster.close();
}
```

```
private static Function<Exception, Boolean> getRetryLogic() {

  return e -> {

    Class<? extends Exception> exceptionClass = e.getClass();

    StringWriter stringWriter = new StringWriter();
    String message = stringWriter.toString();


    if (RemoteConnectionException.class.isAssignableFrom(exceptionClass)){
      System.out.println("Retrying because RemoteConnectionException");
      return true;
    }

    // Check for connection issues
    if (message.contains("Timed out while waiting for an available host") ||
        message.contains("Timed-out") && message.contains("waiting for connection on
 Host") ||
        message.contains("Connection to server is no longer active") ||
        message.contains("Connection reset by peer") ||
        message.contains("SSLEngine closed already") ||
        message.contains("Pool is shutdown") ||
        message.contains("ExtendedClosedChannelException") ||
        message.contains("Broken pipe") ||
        message.contains(System.getenv("neptuneEndpoint")))
    {
      System.out.println("Retrying because connection issue");
      return true;
    };

    // Concurrent writes can sometimes trigger a ConcurrentModificationException.
    // In these circumstances you may want to backoff and retry.
    if (message.contains("ConcurrentModificationException")) {
      System.out.println("Retrying because ConcurrentModificationException");
      return true;
    }

    // If the primary fails over to a new instance, existing connections to the old
 primary will
    // throw a ReadOnlyViolationException. You may want to back and retry.
    if (message.contains("ReadOnlyViolationException")) {
      System.out.println("Retrying because ReadOnlyViolationException");
      return true;
```

```
    }

    System.out.println("Not a retriable error");
    return false;
  };
}
```

使用 Python 连接到 Neptune 数据库实例

如果可以的话，请始终使用你的引擎版本支持的最新版本的 Apac TinkerPop he Python Gremlin 客户端 gremlinpython。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。要使用的`gremlinpython`版本通常与 Java Gremlin 客户端表中描述的 TinkerPop版本保持一致。

> **Note**
>
> 只要你在编写的 `gremlinpython` Gremlin 查询中只使用 TinkerPop 3.4.x 功能，3.5.x 版本就与 3.4.x 版本兼容。

以下部分将指导您完成 Python 示例的运行，该示例连接到 Amazon Neptune 数据库实例并执行 Gremlin 遍历。

必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

开始之前，请执行以下操作：

- 从 Python.org 网站下载并安装 Python 3.6 或更高版本。
- 验证您是否安装了 pip。如果您没有 pip 或不确定是否有，请参阅 pip 文档中的是否需要安装 pip？。
- 如果您的 Python 安装尚未具有该命令，请按如下方式下载 `futures`：`pip install futures`

使用 Python 连接到 Neptune

1. 输入以下命令以安装 `gremlinpython` 程序包：

   ```
   pip install --user gremlinpython
   ```

2. 创建名为 `gremlinexample.py` 的文件，然后在文本编辑器中打开它。

3. 将以下内容复制到 `gremlinexample.py` 文件中。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的地址。

   有关查找 Neptune 数据库实例的地址的信息，请参阅连接到 Amazon Neptune 端点部分。

```
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()

remoteConn = DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin','g')
g = graph.traversal().withRemote(remoteConn)

print(g.V().limit(2).toList())
remoteConn.close()
```

4. 输入以下命令以运行示例：

```
python gremlinexample.py
```

   此示例结尾处的 Gremlin 查询将返回列表中的顶点 (`g.V().limit(2)`)。然后，此列表使用标准 Python `print` 函数输出。

> **Note**
>
> 要将遍历提交到服务器进行评估，需要 Gremlin 查询的最后一部分 `toList()`。如果您未包含该方法或其它等效方法，该查询将不会提交到 Neptune 数据库实例。

   以下方法将查询提交到 Neptune 数据库实例：

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

- `iterate()`

上述示例通过使用 `g.V().limit(2).toList()` 遍历返回图形中的前两个顶点。要查询其他内容，请将其替换为具有其中一种适当的结尾方法的其他 Gremlin 遍历。

使用 .NET 连接到 Neptune 数据库实例

如果可以的话，请始终使用您的引擎版本支持的最新版本的 Apache TinkerPop .NET Gremlin 客户端 Gremlin.Net。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。要使用的`Gremlin.Net`版本通常与 Java Gremlin 客户端表中描述的 TinkerPop版本保持一致。

以下部分包含采用 C# 编写的代码示例，该示例连接到 Neptune 数据库实例并执行 Gremlin 遍历。

到 Amazon Neptune 的连接必须来自与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例。此示例代码已在运行 Ubuntu 的 Amazon EC2 实例上进行测试。

开始之前，请执行以下操作：

- 在 Amazon EC2 实例上安装 .NET。要获取有关在多个操作系统 (包括 Windows、Linux 和 macOS) 上安装 .NET 的说明，请参阅 .NET 入门。
- 通过为您的程序包运行 `dotnet add package gremlin.net` 来安装 Gremlin.NET。有关更多信息，请参阅文档中的 Gremlin.net。 TinkerPop

使用 Gremlin.NET 连接到 Neptune

1. 创建新的 .NET 项目。

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

2. 将目录更改为新项目目录。

```
cd gremlinExample
```

3. 将以下内容复制到 `Program.cs` 文件中。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的地址。

   有关查找 Neptune 数据库实例的地址的信息，请参阅连接到 Amazon Neptune 端点部分。

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Gremlin.Net;
using Gremlin.Net.Driver;
using Gremlin.Net.Driver.Remote;
using Gremlin.Net.Structure;
using static Gremlin.Net.Process.Traversal.AnonymousTraversalSource;
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, 8182, enableSsl: true );
        var gremlinClient = new GremlinClient(gremlinServer);
        var remoteConnection = new DriverRemoteConnection(gremlinClient, "g");
        var g = Traversal().WithRemote(remoteConnection);
        g.AddV("Person").Property("Name", "Justin").Iterate();
        g.AddV("Custom Label").Property("name", "Custom id vertex 1").Iterate();
        g.AddV("Custom Label").Property("name", "Custom id vertex 2").Iterate();
        var output = g.V().Limit<Vertex>(3).ToList();
        foreach(var item in output) {
            Console.WriteLine(item);
        }
      }
      catch (Exception e)
      {
          Console.WriteLine("{0}", e);
      }
    }
  }
}
```

4. 输入以下命令以运行示例：

```
dotnet run
```

此示例结尾处的 Gremlin 查询将返回用于测试目的的单个顶点的计数。然后，它将输出到控制台。

> **Note**
>
> 要将遍历提交到服务器进行评估，需要 Gremlin 查询的最后一部分 `Next()`。如果您未包含该方法或其它等效方法，该查询将不会提交到 Neptune 数据库实例。

以下方法将查询提交到 Neptune 数据库实例：

- `ToList()`
- `ToSet()`
- `Next()`
- `NextTraverser()`
- `Iterate()`

如果您需要序列化并返回查询结果，请使用 `Next()`，否则请使用 `Iterate()`。

上述示例通过使用 `g.V().Limit(3).ToList()` 遍历返回一个列表。要查询其他内容，请将其替换为具有其中一种适当的结尾方法的其他 Gremlin 遍历。

使用 Node.js 连接到 Neptune 数据库实例

如果可以的话，请始终使用您的引擎版本支持的最新版本的 Apache TinkerPop JavaScript Gremlin 客户端 gremlin。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。`gremlin`要使用的版本通常与表中描述的 Java Gremlin 客户端 TinkerPop 版本一致。

以下部分将指导您完成 Node.js 示例的运行，该示例连接到 Amazon Neptune 数据库实例并执行 Gremlin 遍历。

必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

开始之前，请执行以下操作：

- 验证是否已安装 Node.js 版本 8.11 或更高版本。如果没有，请从 Nodejs.org 网站下载并安装 Node.js。

使用 Node.js 连接到 Neptune

1. 输入以下命令以安装 `gremlin-javascript` 程序包：

```
npm install gremlin
```

2. 创建名为 `gremlinexample.js` 的文件并在文本编辑器中打开它。
3. 将以下内容复制到 `gremlinexample.js` 文件中。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的地址。

   有关查找 Neptune 数据库实例的地址的信息，请参阅连接到 Amazon Neptune 端点部分。

```
const gremlin = require('gremlin');
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const Graph = gremlin.structure.Graph;

dc = new DriverRemoteConnection('wss://your-neptune-endpoint:8182/gremlin',{});

const graph = new Graph();
const g = graph.traversal().withRemote(dc);

g.V().limit(1).count().next().
    then(data => {
        console.log(data);
        dc.close();
    }).catch(error => {
        console.log('ERROR', error);
        dc.close();
    });
```

4. 输入以下命令以运行示例：

```
node gremlinexample.js
```

上述示例通过使用 `g.V().limit(1).count().next()` 遍历返回图形中的单个顶点的计数。要查询其他内容，请将其替换为具有其中一种适当的结尾方法的其他 Gremlin 遍历。

> **Note**
>
> 要将遍历提交到服务器进行评估，需要 Gremlin 查询的最后一部分 `next()`。如果您未包含该方法或其它等效方法，该查询将不会提交到 Neptune 数据库实例。

以下方法将查询提交到 Neptune 数据库实例：

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

如果您需要序列化并返回查询结果，请使用 `next()`，否则请使用 `iterate()`。

> **Important**
>
> 这是独立的 Node.js 示例。如果您计划在 AWS Lambda 函数中运行这样的代码，请参阅Lambda 函数示例，了解有关在 Neptune Lambda 函数中 JavaScript 高效使用的详细信息。

使用 Go 连接到 Neptune 数据库实例

如果可以的话，请始终使用您的引擎版本支持的最新版本的 Apache TinkerPop Go Gremlin 客户端 gremlingo。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。

要使用的`gremlingo`版本通常与 Java Gremlin 客户端表中描述的 TinkerPop版本保持一致。

> **Note**
>
> 只要你在编写的 Gremlin 查询中只使用 3. TinkerPop 4.x 功能，gremlingo 3.5.x 版本就向后兼容 3.4.x 版本。

以下部分将指导您完成 Go 示例的运行，该示例连接到 Neptune 数据库实例并执行 Gremlin 遍历。

必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

开始之前，请执行以下操作：

- 从 go.dev 网站下载并安装 Go 1.17 或更高版本。

使用 Go 连接到 Neptune

1. 从空目录开始，初始化一个新的 Go 模块：

```
go mod init example.com/gremlinExample
```

2. 添加 gremlin-go 作为新模块的依赖项：

```
go get github.com/apache/tinkerpop/gremlin-go/v3/driver
```

3. 创建一个名为 `gremlinExample.go` 的文件，然后在文本编辑器中打开它。
4. 将以下内容复制到 `gremlinExample.go` 文件中，同时将 *(your neptune endpoint)* 替换为 Neptune 数据库实例的地址：

```
package main

import (
  "fmt"
  gremlingo "github.com/apache/tinkerpop/gremlin-go/v3/driver"
)

func main() {
  // Creating the connection to the server.
  driverRemoteConnection, err := gremlingo.NewDriverRemoteConnection("wss://(your
 neptune endpoint):8182/gremlin",
    func(settings *gremlingo.DriverRemoteConnectionSettings) {
      settings.TraversalSource = "g"
    })
  if err != nil {
    fmt.Println(err)
    return
  }
  // Cleanup
  defer driverRemoteConnection.Close()
```

```
  // Creating graph traversal
  g := gremlingo.Traversal_().WithRemote(driverRemoteConnection)

  // Perform traversal
  results, err := g.V().Limit(2).ToList()
  if err != nil {
    fmt.Println(err)
    return
  }
  // Print results
  for _, r := range results {
    fmt.Println(r.GetString())
  }
}
```

> **Note**
>
> 搭载 macOS 的 Go 1.18+ 目前不支持 Neptune TLS 证书格式，尝试启动连接时可能会出现 509 错误。对于本地测试，可以通过在导入中添加“crypto/tls”并按如下方式修改 `DriverRemoteConnection` 设置来跳过此操作：
>
> ```
> // Creating the connection to the server.
> driverRemoteConnection, err := gremlingo.NewDriverRemoteConnection("wss://
> your-neptune-endpoint:8182/gremlin",
>   func(settings *gremlingo.DriverRemoteConnectionSettings) {
>       settings.TraversalSource = "g"
>       settings.TlsConfig = &tls.Config{InsecureSkipVerify: true}
>   })
> ```

5. 输入以下命令以运行示例：

```
go run gremlinExample.go
```

此示例结尾处的 Gremlin 查询将返回切片中的顶点 (`g.V().Limit(2)`)。然后，对该切片进行迭代并使用标准 `fmt.Println` 函数进行打印。

> **Note**
>
> 要将遍历提交到服务器进行评估，需要 Gremlin 查询的最后一部分 `ToList()`。如果您未包含该方法或其它等效方法，该查询将不会提交到 Neptune 数据库实例。

以下方法将查询提交到 Neptune 数据库实例：

- `ToList()`
- `ToSet()`
- `Next()`
- `GetResultSet()`
- `Iterate()`

上述示例通过使用 `g.V().Limit(2).ToList()` 遍历返回图形中的前两个顶点。要查询其他内容，请将其替换为具有其中一种适当的结尾方法的其他 Gremlin 遍历。

Gremlin 查询提示

您可使用查询提示为 Amazon Neptune 中的特定 Gremlin 查询指定优化和计算策略。

通过使用以下语法向查询添加 `withSideEffect` 步骤来指定查询提示。

```
g.withSideEffect(hint, value)
```

- hint – 确定要应用的提示的类型。
- value – 确定要考虑的系统方面的行为。

例如，以下示例为如何在 Gremlin 遍历中包含 `repeatMode` 提示。

> **Note**
>
> 所有 Gremlin 查询提示副作用的前缀为 `Neptune#`。

```
g.withSideEffect('Neptune#repeatMode',
 'DFS').V("3").repeat(out()).times(10).limit(1).path()
```

上述查询指示 Neptune 引擎遍历图形时要深度优先 (DFS)，而不是默认的 Neptune 广度优先 (BFS)。

以下各节提供了可用查询提示及其用法的更多信息。

主题

Gremlin repeatMode 查询提示

Neptune `repeatMode` 查询提示指定 Neptune 引擎计算 Gremlin 遍历中的 `repeat()` 步骤的方式：广度优先、深度优先或分块深度优先。

在使用 `repeat()` 步骤查找或跟踪路径时，该步骤的计算模式非常重要，而不只是将步骤重复有限次数。

语法

通过向查询添加 `withSideEffect` 步骤来指定 `repeatMode` 查询提示。

```
g.withSideEffect('Neptune#repeatMode', 'mode').gremlin-traversal
```

> **Note**
>
> 所有 Gremlin 查询提示副作用的前缀为 `Neptune#`。

可用模式

- BFS

广度优先搜索

`repeat()` 步骤的默认执行模式。这将在沿路径深入之前获取所有同级节点。

此版本为内存密集型的，边界可以非常大。查询将耗尽内存并被 Neptune 引擎取消的风险更高。这与其他 Gremlin 实施最匹配。

- DFS

  深度优先搜索

  在继续下一个解之前，跟踪每条路径至最大深度。

  这将使用更少的内存。在类似从多个跃点外的起点查找单一路径这样的情况下，它可提供更好的性能。

- CHUNKED_DFS

  分块深度优先搜索

  一种混合方式，它在 1000 个节点（既不是 1 个节点 (DFS)，也不是所有节点 (BFS)）组成的区块中，以深度优先方式搜索图形。

  Neptune 引擎将在每个级别获得多达 1000 个节点，然后更深入地跟踪该路径。

  这是一种速度和内存使用率之间的平衡方法。

  如果您要使用 BFS，但查询占用的内存太多，这也会很有用。

示例

以下各节介绍了重复模式对 Gremlin 遍历的影响。

在 Neptune 中，`repeat()` 步骤的默认模式是对所有遍历实施广度优先 (BFS) 执行策略。

在大多数情况下， TinkerGraph 实现使用相同的执行策略，但在某些情况下，它会改变遍历的执行。

例如， TinkerGraph 实现修改了以下查询。

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

此遍历中的 `repeat()` 步骤将“展开”到以下遍历，这将导致执行深度优先 (DFS) 策略。

```
g.V(<id>).out().out().out().out().out().out().out().out().out().out().limit(1).path()
```

> **Important**
>
> Neptune 查询引擎不会自动执行此操作。

广度优先 (BFS) 是默认的执行策略，与大多数 TinkerGraph 情况下的执行策略类似。但是，在特定情况下会优先选择深度优先 (DFS) 策略。

BFS（默认）

广度优先 (BFS) 是 `repeat()` 运算符的默认执行策略。

```
g.V("3").repeat(out()).times(10).limit(1).path()
```

Neptune 引擎将完全探索前九个跃点边界，然后再向外扩展十个跃点查找解。这在许多情况（如最短路径查询）下很有效率。

但是，对于上述示例，遍历使用 `repeat()` 运算符的深度优先 (DFS) 模式将快得多。

DFS

以下查询使用 `repeat()` 运算符的深度优先 (DFS) 模式。

```
g.withSideEffect("Neptune#repeatMode", "DFS").V("3").repeat(out()).times(10).limit(1)
```

在探索下一个解之前，这将跟踪每个单独的解，直至最大深度。

Gremlin noReordering 查询提示

在提交 Gremlin 遍历时，Neptune 查询引擎将调查遍历的结构并对查询的各个部分重新排序，尝试最大程度地减少计算所需的工作量和查询响应时间。例如，具有多个约束的遍历（如多个 `has()` 步骤）通常不会按给定顺序进行计算。相反，它在使用静态分析检查查询之后进行重新排序。

Neptune 查询引擎尝试确定哪个约束更适合选择并最先运行该约束。这通常会实现更高的性能，但 Neptune 选择计算查询的顺序可能并不总是最佳的。

如果您知道确切的数据特性并且想要手动指示查询执行顺序，请使用 Neptune noReordering 查询提示指定按给定顺序计算遍历。

语法

通过向查询添加 withSideEffect 步骤来指定 noReordering 查询提示。

```
g.withSideEffect('Neptune#noReordering', true or false).gremlin-traversal
```

> **Note**
>
> 所有 Gremlin 查询提示副作用的前缀为 Neptune#。

可用值

- true
- false

Gremlin typePromotion 查询提示

当您提交根据数值或范围进行筛选的 Gremlin 遍历时，Neptune 查询引擎在执行查询时通常必须使用类型提升。这意味着它必须检查每种类型的值，这些值可能包含您正在筛选的值。

例如，如果要筛选等于 55 的值，则引擎必须查找等于 55 的整数、等于 55L 的长整数、等于 55.0 的浮点数，依此类推。每个类型提升都需要在存储上进行额外查找，这可能会导致看似简单的查询需要意想不到的长时间才能完成。

假设您正在搜索所有客户年龄属性大于 5 的顶点：

```
g.V().has('customerAge', gt(5))
```

要彻底执行该遍历，Neptune 必须扩展查询，以检查您要查询的值可以提升到的所有数字类型。在这种情况下，必须对任何大于 5 的整数、任何超过 5L 的长整数、任何超过 5.0 的浮点数以及任何大于 5.0 的双精度数应用 gt 筛选条件。由于每种类型提升都需要在存储空间上进行额外查找，因此，当您针对此查询运行 Gremlin profile API 时，您会看到每个数字筛选条件都有多个筛选条件，而且完成所需的时间将比您预期的要长得多。

通常没有必要进行类型提升，因为您事先知道只需要查找一种特定类型的值即可。在这种情况下，您可以使用 typePromotion 查询提示来关闭类型提升，从而大大加快查询速度。

语法

通过向查询添加 withSideEffect 步骤来指定 typePromotion 查询提示。

```
g.withSideEffect('Neptune#typePromotion', true or false).gremlin-traversal
```

> **Note**
>
> 所有 Gremlin 查询提示副作用的前缀为 Neptune#。

可用值

- true
- false

要关闭上述查询的类型提升，您可以使用：

```
g.withSideEffect('Neptune#typePromotion', false).V().has('customerAge', gt(5))
```

Gremlin useDFE 查询提示

使用此查询提示可以启用 DFE 来执行查询。默认情况下，如果不将此查询提示设置为 true，Neptune 就不会使用 DFE，因为 neptune_dfe_query_engine 实例参数默认为 viaQueryHint。如果将该实例参数设置为 enabled，则除 useDFE 查询提示设置为 false 的查询外，所有查询都将使用 DFE 引擎。

为查询启用 DFE 的示例：

```
g.withSideEffect('Neptune#useDFE', true).V().out()
```

使用结果缓存的 Gremlin 查询提示

启用查询结果缓存后，可以使用以下查询提示。

Gremlin **enableResultCache** 查询提示

如果查询结果已被缓存，则值为 `true` 的 `enableResultCache` 查询提示会使查询结果从缓存中返回。否则，它将返回新的结果并缓存它们，直到从缓存中清除它们。例如：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

稍后，您可以通过再次发出完全相同的查询来访问缓存的结果。

如果此查询提示的值为 `false`，或者它不存在，则不会缓存查询结果。但是，将其设置为 `false` 不会清除现有的缓存结果。要清除缓存的结果，请使用 `invalidateResultCache` 或 `invalidateResultCachekey` 提示。

Gremlin **enableResultCacheWithTTL** 查询提示

`enableResultCacheWithTTL` 查询提示还会返回缓存结果（如果有），而不影响缓存中已有结果的 TTL。如果当前没有缓存结果，则查询会返回新结果并将其缓存由 `enableResultCacheWithTTL` 查询提示指定的生存时间 (TTL)。该生存时间以秒为单位指定。例如，以下查询将生存时间指定为六十秒：

```
g.with('Neptune#enableResultCacheWithTTL', 60)
 .V().has('genre','drama').in('likes')
```

在 60 秒 time-to-live 结束之前，您可以使用带有`enableResultCache`或`enableResultCacheWithTTL`查询提示的相同查询（此处`g.V().has('genre','drama').in('likes')`）来访问缓存的结果。

> **Note**
>
> 用 `enableResultCacheWithTTL` 指定的生存时间不会影响已经缓存的结果。
>
> - 如果之前使用 `enableResultCache` 缓存结果，则必须先显式清除缓存，然后 `enableResultCacheWithTTL` 才能生成新结果并将结果缓存达它指定的 TTL。
> - 如果之前使用 `enableResultCachewithTTL` 缓存结果，则该先前的 TTL 必须先过期，然后 `enableResultCacheWithTTL` 才能生成新结果并将结果缓存达它指定的 TTL。

生存时间过后，查询的缓存结果将被清除，同一查询的后续实例随后会返回新的结果。如果 `enableResultCacheWithTTL` 附加到该后续查询，则新结果将使用它指定的 TTL 进行缓存。

Gremlin **invalidateResultCacheKey** 查询提示

invalidateResultCacheKey 查询提示可以取 true 或 false 值。true 值会导致 invalidateResultCacheKey 附加到的查询的缓存结果被清除。例如，以下示例会导致为查询键 g.V().has('genre','drama').in('likes') 缓存的结果被清除：

```
g.with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

上面的示例查询不会导致其新结果被缓存。如果要在清除现有缓存结果后缓存新结果，则可以在同一个查询中包含 enableResultCache（或 enableResultCacheWithTTL）：

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **invalidateResultCache** 查询提示

invalidateResultCache 查询提示可以取 true 或 false 值。true 值会导致结果缓存中的所有结果都被清除。例如：

```
g.with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

上面的示例查询不会导致其结果被缓存。如果要在完全清除现有缓存后缓存新结果，则可以在同一个查询中包含 enableResultCache（或 enableResultCacheWithTTL）：

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

Gremlin **numResultsCached** 查询提示

numResultsCached 查询提示只能用于包含 iterate() 的查询，它指定要为它附加到的查询缓存的最大结果数。请注意，不会返回当 numResultsCached 存在时缓存的结果，而只缓存这些结果。

例如，以下查询指定最多应缓存 100 个结果，但不返回其中任何缓存结果：

```
g.with('Neptune#enableResultCache', true)
```

```
  .with('Neptune#numResultsCached', 100)
  .V().has('genre','drama').in('likes').iterate()
```

然后，您可以使用如下所示的查询来检索一定范围的缓存结果（此处为前十个）：

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#numResultsCached', 100)
  .V().has('genre','drama').in('likes').range(0, 10)
```

Gremlin **noCacheExceptions** 查询提示

noCacheExceptions 查询提示可以取 true 或 false 值。true 值会导致抑制与结果缓存相关的所有异常。例如：

```
g.with('Neptune#enableResultCache', true)
  .with('Neptune#noCacheExceptions', true)
  .V().has('genre','drama').in('likes')
```

特别是，这会抑制 QueryLimitExceededException，如果查询的结果太大而无法容纳在结果缓存中，则会引发该异常。

Gremlin 查询状态 API

要获取 Gremlin 查询的状态，请使用 HTTP GET 或 POST 来向 https://*your-neptune-endpoint*:*port*/gremlin/status 终端节点提出请求。

Gremlin 查询状态请求参数

- queryId（可选）- 正在运行的 Gremlin 查询的 ID。仅显示指定查询的状态。
- includeWaiting（可选）– 返回所有正在等待的查询的状态。

  通常，响应中只包含正在运行的查询，但是当指定 includeWaiting 参数时，还会返回所有等待查询的状态。

Gremlin 查询状态响应语法

```
{
  "acceptedQueryCount": integer,
  "runningQueryCount": integer,
```

```
  "queries": [
    {
      "queryId":"guid",
      "queryEvalStats":
        {
          "waited": integer,
          "elapsed": integer,
          "cancelled": boolean
        },
      "queryString": "string"
    }
  ]
}
```

Gremlin 查询状态响应值

- ac cepted QueryCount — 已接受但尚未完成的查询数量，包括队列中的查询。
- 正在运行 QueryCount — 当前正在运行的 Gremlin 查询的数量。
- queries – 当前 Gremlin 查询的列表。
- queryId – 查询的 GUID id。Neptune 为每个查询自动分配该 ID 值，或者您也可以分配自己的 ID（请参阅将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中）。
- query EvalStats-此查询的统计信息。
- subqueries – 此查询中子查询的数量。
- elapsed – 到目前为止，查询已运行的毫秒数。
- cancelled – True 指示查询已取消。
- queryString – 提交的查询。在超过 1024 个字符时，将截断为此长度。
- waited – 表示查询等待了多长时间，以毫秒为单位。

Gremlin 查询状态示例

以下是使用 `curl` 和 HTTP `GET` 的状态命令的示例。

```
curl https://your-neptune-endpoint:port/gremlin/status
```

此输出显示了一个正在运行的查询。

```
{
```

```
  "acceptedQueryCount":9,
  "runningQueryCount":1,
  "queries": [
    {
      "queryId":"fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
      "queryEvalStats":
        {
          "waited": 0,
          "elapsed": 23,
          "cancelled": false
        },
      "queryString": "g.V().out().count()"
    }
  ]
}
```

Gremlin 查询取消

要获取 Gremlin 查询的状态，请使用 HTTP GET 或 POST 来向 `https://`*your-neptune-endpoint*`:`*port*`/gremlin/status` 终端节点提出请求。

Gremlin 查询取消请求参数

- cancelQuery – 取消时必需。此参数没有相应的值。
- queryId – 要取消的正在运行的 Gremlin 查询的 ID。

Gremlin 查询取消示例

以下是使用 `curl` 命令取消查询的示例。

```
curl https://your-neptune-endpoint:port/gremlin/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=fb34cd3e-f37c-4d12-9cf2-03bb741bf54f"
```

成功取消返回 HTTP `200` OK。

对基于 Gremlin 脚本的会话的支持

您可以将 Gremlin 会话与 Amazon Neptune 中的隐式事务结合使用。有关 Gremlin 会话的信息，请参阅 Ap TinkerPop ache 文档中的考虑会话。以下各节描述了如何在 Java 中使用 Gremlin 会话。

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.1.0.200463.0 开始推出。
> 从 Nep tune 引擎版本 1.1.0 和 3.5.2 TinkerPop 版本开始，你也可以使用。Gremlin 事务

> **Important**
>
> 目前，Neptune 可以保持基于脚本的会话打开的最长时间为 10 分钟。如果您未在此时间之前关闭会话，会话超时，其中的所有内容将回滚。

主题

- Gremlin 控制台上的 Gremlin 会话
- Gremlin 语言变体中的 Gremlin 会话

Gremlin 控制台上的 Gremlin 会话

如果在 Gremlin 控制台上创建不带 `session` 参数的远程连接，则将以无会话模式创建远程连接。在这种模式下，提交到服务器的每个请求本身都被视为完整事务，并且请求之间不保存状态。如果某个请求失败，只回滚该请求。

如果您的确使用 `session` 参数创建远程连接，则您创建的基于脚本的会话会持续，直到您关闭远程连接。每个会话由控制台生成并返回给您的唯一 UUID 标识。

以下是创建会话的控制台调用的示例。提交查询后，新的调用将关闭该会话并提交查询。

> **Note**
>
> 必须始终关闭 Gremlin 客户端才能释放服务器端资源。

```
gremlin> :remote connect tinkerpop.server conf/neptune-remote.yaml session
  . . .
  . . .
gremlin> :remote close
```

有关更多信息和示例，请参阅 TinkerPop 文档中的会话。

您在会话期间运行的所有查询构成一个事务，直到所有查询成功并且您关闭远程连接后，该事务才会提交。如果某个查询失败，或者如果您未在 Neptune 支持的最大会话生命周期内关闭连接，则不会提交会话事务，并回滚其中的所有查询。

Gremlin 语言变体中的 Gremlin 会话

在 Gremlin 语言变体 (GLV) 中，您需要创建 `SessionedClient` 对象，以便在单个事务中发出多个查询，如下面的示例所示。

```
try {                               // line 1
  Cluster cluster = Cluster.open();                    // line 2
  Client client = cluster.connect("sessionName");      // line 3
   ...
   ...
} finally {
  // Always close. If there are no errors, the transaction is committed; otherwise,
 it's rolled back.
  client.close();
}
```

上一示例中的第 3 行根据为所讨论的集群设置的配置选项创建 `SessionedClient` 对象。您传递到连接方法的 *`sessionName`* 字符串成为会话的唯一名称。为避免冲突，请为名称使用 UUID。

客户端在初始化时启动会话事务。仅当您调用 `client.close( )` 时，才会提交在构成该会话时运行的所有查询。同样，如果单个查询失败，或者如果您未在 Neptune 支持的最大会话生命周期内关闭连接，则会话事务失败，并回滚其中的所有查询。

> **Note**
>
> 必须始终关闭 Gremlin 客户端才能释放服务器端资源。

```
GraphTraversalSource g = traversal().withRemote(conn);

Transaction tx = g.tx();

// Spawn a GraphTraversalSource from the Transaction.
// Traversals spawned from gtx are executed within a single transaction.
GraphTraversalSource gtx = tx.begin();
try {
```

```
  gtx.addV('person').iterate();
  gtx.addV('software').iterate();

  tx.commit();
} finally {
    if (tx.isOpen()) {
        tx.rollback();
    }
}
```

Neptune 中的 Gremlin 事务

Gremlin 事务的执行上下文有多种。在与 Gremlin 结合使用时，务必了解您所处的上下文及其影响：

- **`Script-based`** – 使用基于文本的 Gremlin 字符串发出请求，如下所示：
  - 使用 Java 驱动程序和 `Client.submit(`*`string`*`)`。
  - 使用 Gremlin 控制台和 `:remote connect`。
  - 使用 HTTP API。
- **`Bytecode-based`** – 请求是使用 Gremlin 语言变体 (GLV) 中典型的序列化的 Gremlin 字节码发出的。

  例如，使用 Java 驱动程序 `g = traversal().withRemote(...)`。

对于上述任一上下文，都有额外的上下文，即请求以无会话或绑定到会话的形式发送。

> **Note**
>
> Gremlin 事务必须始终提交或回滚，这样才能释放服务器端资源。

无会话请求

无会话时，请求等同于单个事务。

对于脚本来说，这意味着在单个请求中发送的一条或多条 Gremlin 语句将作为单个事务提交或回滚。例如：

```
Cluster cluster = Cluster.open();
Client client = cluster.connect(); // sessionless
```

```
// 3 vertex additions in one request/transaction:
client.submit("g.addV();g.addV();g.addV()").all().get();
```

对于字节码，会对从 g 生成和执行的每个遍历发出无会话请求：

```
GraphTraversalSource g = traversal().withRemote(...);

// 3 vertex additions in three individual requests/transactions:
g.addV().iterate();
g.addV().iterate();
g.addV().iterate();

// 3 vertex additions in one single request/transaction:
g.addV().addV().addV().iterate();
```

绑定到会话的请求

绑定到会话时，可以在单个事务的上下文中应用多个请求。

对于脚本来说，这意味着没有必要将所有图形操作串联成单个嵌入式字符串值：

```
Cluster cluster = Cluster.open();
Client client = cluster.connect(sessionName); // session
try {
    // 3 vertex additions in one request/transaction:
    client.submit("g.addV();g.addV();g.addV()").all().get();
} finally {
    client.close();
}

try {
    // 3 vertex additions in three requests, but one transaction:
    client.submit("g.addV()").all().get(); // starts a new transaction with the same
 sessionName
    client.submit("g.addV()").all().get();
    client.submit("g.addV()").all().get();
} finally {
    client.close();
}
```

对于字节码，之后 TinkerPop `3.5.x`，可以显式控制事务并透明地管理会话。Gremlin 语言变体 (GLV) 支持 Gremlin 对事务执行 `commit()` 或 `rollback()` 的 `tx()` 语法，如下所示：

```
GraphTraversalSource g = traversal().withRemote(conn);

Transaction tx = g.tx();

// Spawn a GraphTraversalSource from the Transaction.
// Traversals spawned from gtx are executed within a single transaction.
GraphTraversalSource gtx = tx.begin();
try {
    gtx.addV('person').iterate();
    gtx.addV('software').iterate();

    tx.commit();
} finally {
    if (tx.isOpen()) {
        tx.rollback();
    }
}
```

尽管上面的示例是用 Java 编写的，但您也可以在 Python、Javascript 和 .NET 中使用这种 `tx()` 语法。

> **Warning**
>
> 无会话只读查询在 SNAPSHOT 隔离下执行，但在显式事务中运行的只读查询则在 SERIALIZABLE 隔离下执行。与在 SNAPSHOT 隔离下运行的只读查询不同，在 SERIALIZABLE 隔离下执行的只读查询会产生更高的开销，并且可能阻塞并发写入或被并发写入阻塞。

将 Gremlin API 与 Amazon Neptune 结合使用

> **Note**
>
> Amazon Neptune 不支持 `bindings` 属性。

所有 Gremlin HTTPS 请求均使用单个终端节点：`https://`*`your-neptune-endpoint`*`:`*`port`*`/gremlin`。所有 Neptune 连接都必须使用 HTTPS。

你可以直接通过 Gremlin 控制台连接到 Neptune 图表。 WebSockets

有关连接到 Gremlin 终端节点的更多信息，请参阅使用 Gremlin 访问 Neptune 图形。

Gremlin 的 Amazon Neptune 实施具有您需要考虑的特定详细信息和差异。有关更多信息，请参阅 Amazon Neptune 中的 Gremlin 标准合规性。

有关 Gremlin 语言和遍历的信息，请参阅 Apache 文档中的遍历。 TinkerPop

在 Amazon Neptune Gremlin 中缓存查询结果

从引擎版本 1.0.5.1 开始，Amazon Neptune 支持 Gremlin 查询的结果缓存。

您可以启用查询结果缓存，然后使用查询提示来缓存 Gremlin 只读查询的结果。

然后，只要查询仍在缓存中，查询的任何重新运行都会以低延迟且没有 I/O 成本的方式检索缓存的结果。这适用于在 HTTP 端点和使用 Websocket 提交的查询，无论是以字节码还是字符串形式提交。

> **Note**
>
> 即使启用了查询缓存，也不会缓存发送到配置文件端点的查询。

您可以通过多种方式控制 Neptune 查询结果缓存的行为。例如：

- 您可以将缓存的结果以块为单位进行分页。
- 您可以为指定的查询指定 time-to-live (TTL)。
- 您可以清除指定查询的缓存。
- 您可以清除整个缓存。
- 您可以设置为在结果超过缓存大小时收到通知。

缓存是使用 least-recently-used (LRU) 策略维护的，这意味着一旦分配给缓存的空间已满，就会删除 least-recently-used 结果，以便在缓存新结果时腾出空间。

> **Important**
>
> 查询结果缓存不适用于 `t3.medium` 或 `t4.medium` 实例类型。

在 Neptune 中启用查询结果缓存

要在 Neptune 中启用查询结果缓存，请使用控制台将 `neptune_result_cache` 数据库实例参数设置为 1（已启用）。

启用结果缓存后，Neptune 会预留一部分当前内存用于缓存查询结果。您使用的实例类型越大，可用内存越多，Neptune 为缓存预留的内存就越多。

如果结果缓存内存已满，Neptune 会自动丢弃 least-recently-used (LRU) 缓存的结果，以便为新的结果腾出空间。

您可使用 实例状态 命令检查结果缓存的当前状态。

使用提示缓存查询结果

启用查询结果缓存后，您可以使用查询提示来控制查询缓存。以下所有示例都适用于相同的查询遍历，即：

```
g.V().has('genre','drama').in('likes')
```

使用 `enableResultCache`

启用查询结果缓存后，您可以使用 `enableResultCache` 查询提示缓存 Gremlin 查询的结果，如下所示：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

然后，Neptune 会将查询结果返回给您，并对其进行缓存。稍后，您可以通过再次发出完全相同的查询来访问缓存的结果：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

标识缓存结果的缓存键是查询字符串本身，即：

```
g.V().has('genre','drama').in('likes')
```

使用 enableResultCacheWithTTL

您可以使用 enableResultCacheWithTTL 查询提示来指定查询结果应缓存多长时间。例如，以下查询指定查询结果应在 120 秒后过期：

```
g.with('Neptune#enableResultCacheWithTTL', 120)
 .V().has('genre','drama').in('likes')
```

同样，标识缓存结果的缓存键是基本查询字符串：

```
g.V().has('genre','drama').in('likes')
```

再一次，您可以使用带有 enableResultCache 查询提示的查询字符串来访问缓存的结果：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

如果自缓存结果以来已过去 120 秒或更长时间，则该查询将返回新的结果，并对其进行缓存，但没有任何结果 time-to-live。

您也可以通过再次发出带有 enableResultCacheWithTTL 查询提示的相同查询来访问缓存的结果。例如：

```
g.with('Neptune#enableResultCacheWithTTL', 140)
 .V().has('genre','drama').in('likes')
```

在 120 秒（即当前有效的 TTL）过去之前，这个使用 enableResultCacheWithTTL 查询提示的新查询将返回缓存的结果。120 秒后，它将返回新结果并将其缓存 140 秒。 time-to-live

> **Note**
>
> 如果查询密钥的结果已被缓存，则与的相同查询键enableResultCacheWithTTL不会生成新结果，也不会影响当前缓存 time-to-live 的结果。
>
> - 如果之前使用 enableResultCache 缓存结果，则必须先清除缓存，然后 enableResultCacheWithTTL 才能生成新结果并将结果缓存达它指定的 TTL。
> - 如果之前使用 enableResultCachewithTTL 缓存结果，则该先前的 TTL 必须先过期，然后 enableResultCacheWithTTL 才能生成新结果并将结果缓存达它指定的 TTL。

使用 **invalidateResultCacheKey**

您可以使用 invalidateResultCacheKey 查询提示清除一个特定查询的缓存结果。例如：

```
g.with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

该查询会清除查询键 g.V().has('genre','drama').in('likes') 的缓存，并返回该查询的新结果。

也可以将 invalidateResultCacheKey 与 enableResultCache 或 enableResultCacheWithTTL 结合使用。例如，以下查询清除当前缓存的结果，缓存新结果并返回它们：

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCacheKey', true)
 .V().has('genre','drama').in('likes')
```

使用 **invalidateResultCache**

您可以使用 invalidateResultCache 查询提示清除查询结果缓存中所有缓存的结果。例如：

```
g.with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

该查询会清除整个结果缓存并返回查询的新结果。

也可以将 invalidateResultCache 与 enableResultCache 或 enableResultCacheWithTTL 结合使用。例如，以下查询会清除整个结果缓存，缓存该查询的新结果，然后返回这些结果：

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#invalidateResultCache', true)
 .V().has('genre','drama').in('likes')
```

对缓存查询结果进行分页

假设您已经缓存了大量这样的结果：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes')
```

现在假设您发出以下范围查询：

```
g.with('Neptune#enableResultCache', true)
 .V().has('genre','drama').in('likes').range(0,10)
```

Neptune 首先寻找完整的缓存键，即 `g.V().has('genre','drama').in('likes').range(0,10)`。如果该键不存在，Neptune 接下来会查看该查询字符串中是否有不包含范围的键（即 `g.V().has('genre','drama').in('likes')`）。当它找到该键时，Neptune 会按照该范围所指定，从其缓存中获取前十个结果。

> **Note**
>
> 如果您将 `invalidateResultCacheKey` 查询提示用于末尾有某个范围的查询，当 Neptune 找不到与具有该范围的查询完全匹配的查询时，它会清除没有该范围的查询的缓存。

配合使用 **numResultsCached**和 **.iterate()**

使用 numResultsCached 查询提示，可以在不返回所有正在缓存的结果的情况下填充结果缓存，这在您更喜欢对大量结果进行分页时很有用。

numResultsCached 查询提示仅适用于以 `iterate()` 结尾的查询。

例如，如果要缓存示例查询的前 50 个结果：

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').iterate()
```

在本例中，缓存中的查询键为：`g.with("Neptune#numResultsCached", 50).V().has('genre','drama').in('likes')`。现在，您可以使用此查询检索前十个缓存的结果：

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').range(0, 10)
```

而且，您可以按如下方式从查询中检索接下来的十个结果：

```
g.with("Neptune#enableResultCache", true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').range(10, 20)
```

请勿忘记加入 numResultsCached 提示！它是查询键的重要组成部分，因此必须存在才能访问缓存的结果。

使用 **numResultsCached** 时，请记住以下事项：

- 使用 **numResultsCached** 提供的数字将应用于查询的末尾。 例如，这意味着以下查询实际上缓存了范围 (1000, 1500) 内的结果：

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 500)
   .V().range(1000, 2000).iterate()
  ```

- 使用 **numResultsCached** 提供的数字指定要缓存的最大结果数。 例如，这意味着以下查询实际上缓存了范围 (1000, 2000) 内的结果：

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 100000)
   .V().range(1000, 2000).iterate()
  ```

- 以 **.range().iterate()** 结尾的查询缓存的结果有其自己的范围。 例如，假设您使用如下查询来缓存结果：

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 500)
   .V().range(1000, 2000).iterate()
  ```

  要从缓存中检索前 100 个结果，您可以编写如下查询：

  ```
  g.with("Neptune#enableResultCache", true)
   .with("Neptune#numResultsCached", 500)
   .V().range(1000, 2000).range(0, 100)
  ```

  这一百个结果将等同于范围 (1000, 1100) 中基本查询的结果。

用于查找缓存结果的查询缓存键

缓存查询的结果后，使用相同查询缓存键的后续查询将从缓存中检索结果，而不是生成新的结果。查询的查询缓存键的计算方式如下：

1. 除 numResultsCached 外，所有与缓存相关的查询提示都将被忽略。
2. 最终 iterate() 步骤被忽略。
3. 查询的其余部分根据其字节码表示进行排序。

将生成的字符串与缓存中已有的查询结果索引进行匹配，以确定查询是否存在缓存命中。

例如，使用以下查询：

```
g.withSideEffect('Neptune#typePromotion', false).with("Neptune#enableResultCache",
 true)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes').iterate()
```

它将存储为以下内容的字节码版本：

```
g.withSideEffect('Neptune#typePromotion', false)
 .with("Neptune#numResultsCached", 50)
 .V().has('genre','drama').in('likes')
```

与结果缓存相关的异常

如果您尝试缓存的查询结果太大而无法容纳在缓存内存中，即使删除了先前缓存的所有内容也是如此，那么 Neptune 就会引发 QueryLimitExceededException 故障。不返回任何结果，并且异常会生成以下错误消息：

```
The result size is larger than the allocated cache,
      please refer to results cache best practices for options to rerun the query.
```

您可以使用 noCacheExceptions 查询提示抑制此消息，如下所示：

```
g.with('Neptune#enableResultCache', true)
 .with('Neptune#noCacheExceptions', true)
 .V().has('genre','drama').in('likes')
```

使用 Gremlin `mergeV()` 和 `mergeE()` 步骤进行高效的更新插入

如果顶点或边缘已经存在，更新插入（或条件插入）会重复使用顶点或边缘，如果不存在，则创建它。高效的更新插入可以显著改变 Gremlin 查询的性能。

更新插入允许您编写幂等插入操作：无论您运行多少次这样的操作，总体结果都是一样的。这在高度并发的写入场景中很有用，在这种情况下，对图形的同一部分进行并发修改可能会强制一个或多个事务回滚并引发 `ConcurrentModificationException`，从而需要重试。

例如，以下查询使用提供的 `Map` 来更新插入顶点，以首先尝试查找 `T.id` 为 `"v-1"` 的顶点。如果找到该顶点，则将其返回。如果找不到，则通过 `onCreate` 子句创建具有 `id` 和属性的顶点。

```
g.mergeV([(id):'v-1']).
  option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org'])
```

批处理更新插入以提高吞吐量

对于高吞吐量的写入场景，您可以将 `mergeV()` 和 `mergeE()` 步骤串联在一起，以批量更新插入顶点和边缘。批处理减少了更新插入大量顶点和边缘的事务开销。然后，您可以通过使用多个客户端并行更新插入批量请求来进一步提高吞吐量。

根据经验，我们建议对于每个批量请求，更新插入大约 200 条记录。记录是单个顶点或边缘标签或属性。例如，具有单个标签和 4 个属性的顶点会创建 5 条记录。带有标签和单个属性的边缘会创建 2 条记录。如果您想更新插入批量顶点，每个顶点都有一个标签和 4 个属性，那么您应该从批量大小为 40 开始，因为 `200 / (1 + 4) = 40`。

您可以试验批量大小。每批 200 条记录是一个不错的起点，但理想的批量大小可能会更高或更低，具体取决于您的工作负载。但请注意，Neptune 可能会限制每个请求的 Gremlin 步骤总数。此限制未记录在案，但为了安全起见，请尽量确保您的请求包含不超过 1500 个 Gremlin 步骤。Neptune 可能会拒绝超过 1500 个步骤的大批量请求。

要提高吞吐量，您可以使用多个客户端并行批量更新插入（请参阅[创建高效的多线程 Gremlin 写入]）。客户端数量应与 Neptune 写入器实例上的工作线程数相同，通常是服务器上 vCPU 数量的 2 倍。例如，一个 `r5.8xlarge` 实例有 32 个 vCPU 和 64 个工作线程。对于使用 `r5.8xlarge` 的高吞吐量写入场景，您将使用 64 个客户端并行向 Neptune 写入批量更新插入。

每个客户端都应提交批量请求，等待请求完成后，再提交另一个请求。尽管多个客户端并行运行，但每个客户端都以串行方式提交请求。这样可以确保服务器获得稳定的请求流，这些请求会占用所有工作线程，而不会淹没服务器端的请求队列（请参阅[调整 Neptune 数据库集群中数据库实例的大小]）。

尽量避免生成多个遍历器的步骤

当 Gremlin 步骤执行时，它会接受一个传入的遍历器，并发出一个或多个输出遍历器。一个步骤发出的遍历器的数量决定了执行下一步的次数。

通常，在执行批量操作时，您希望每个操作（例如更新插入顶点 A）执行一次，以便操作序列如下所示：更新插入顶点 A，接着更新插入顶点 B，然后更新插入顶点 C，依此类推。只要一个步骤只创建或修改一个元素，它就会只发出一个遍历器，而代表下一个操作的步骤只执行一次。另一方面，如果一个操作创建或修改了多个元素，则它会发出多个遍历器，这反过来又会导致后续步骤多次执行，每个发出的遍历器执行一次。这可能会导致数据库执行不必要的额外工作，在某些情况下，还可能导致创建不想要的额外顶点、边缘或属性值。

诸如 `g.V().addV()` 的查询就是一个例子，说明事情如何出错。这个简单的查询会为在图形中找到的每个顶点添加一个顶点，因为 `V()` 会为图形中的每个顶点发出一个遍历器，而每个遍历器都会触发对 `addV()` 的调用。

有关处理可能发出多个遍历器的操作的方法，请参阅混用更新插入和插入。

更新插入顶点

`mergeV()` 步骤是专门为更新插入顶点而设计的。它将 `Map` 作为参数来表示要匹配图形中现有顶点的元素，如果找不到元素，则使用该 `Map` 来创建新的顶点。该步骤还允许您在创建或匹配时更改行为，其中 `option()` 调制器可以与 `Merge.onCreate` 和 `Merge.onMatch` 令牌一起应用来控制相应的行为。有关如何使用此步骤的更多信息，请参阅 TinkerPop 参考文档。

您可以使用顶点 ID 来确定特定顶点是否存在。这是首选方法，因为 Neptune 针对围绕 ID 的高度并发用例优化了更新插入。例如，以下查询会创建一个具有给定顶点 ID 的顶点（如果该顶点尚不存在），如果已存在，则重用它：

```
g.mergeV([(T.id): 'v-1']).
    option(onCreate, [(T.label): 'PERSON', email: 'person-1@example.org', age: 21]).
    option(onMatch, [age: 22]).
  id()
```

请注意，此查询以 `id()` 步骤结尾。虽然对于更新插入顶点来说并不是绝对必要的，但在更新插入查询结尾采取 `id()` 步骤，可以确保服务器不会将所有顶点属性序列化回客户端，这有助于降低查询的锁定成本。

或者，您可以使用顶点属性来标识顶点：

```
g.mergeV([email: 'person-1@example.org']).
```

```
    option(onCreate, [(T.label): 'PERSON', age: 21]).
    option(onMatch, [age: 22]).
  id()
```

如果可能，请使用您自己的用户提供的 ID 来创建顶点，并使用这些 ID 来确定在更新插入操作期间是否存在顶点。这让 Neptune 可以优化更新插入。当并发修改很常见时，基于 ID 的更新插入可能比基于属性的更新插入效率要高得多。

串联顶点更新插入

您可以将顶点更新插入串联在一起以批量插入它们：

```
g.V('v-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-1')
                         .property('email', 'person-1@example.org'))
 .V('v-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-2')
                         .property('email', 'person-2@example.org'))
 .V('v-3')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-3')
                         .property('email', 'person-3@example.org'))
 .id()
```

或者，您也可以使用以下 `mergeV()` 语法：

```
g.mergeV([(T.id): 'v-1', (T.label): 'PERSON', email: 'person-1@example.org']).
  mergeV([(T.id): 'v-2', (T.label): 'PERSON', email: 'person-2@example.org']).
  mergeV([(T.id): 'v-3', (T.label): 'PERSON', email: 'person-3@example.org'])
```

然而，由于这种形式的查询在搜索条件中包含的元素对于按 `id` 执行的基本查找来说是多余的，所以它不如以前的查询效率高。

更新插入边缘

`mergeE()` 步骤是专门为更新插入边缘而设计的。它将 `Map` 作为参数来表示要匹配图形中现有边缘的元素，如果找不到元素，则使用该 `Map` 来创建新的边缘。该步骤还允许您在创建或匹配时更改行为，

其中 option() 调制器可以与 Merge.onCreate 和 Merge.onMatch 令牌一起应用来控制相应的行为。有关如何使用此步骤的更多信息，请参阅 TinkerPop 参考文档。

您可以使用边缘 ID 来更新插入边缘，就像使用自定义顶点 ID 更新插入顶点一样。同样，这是首选方法，因为它允许 Neptune 优化查询。例如，如果边缘尚不存在，则以下查询会根据其边缘 ID 创建该边缘，如果存在，则重用该边缘。如果该查询需要创建新边缘，它还会使用 Direction.from 和 Direction.to 顶点的 ID：

```
g.mergeE([(T.id): 'e-1']).
    option(onCreate, [(from): 'v-1', (to): 'v-2', weight: 1.0]).
    option(onMatch, [weight: 0.5]).
  id()
```

请注意，此查询以 id() 步骤结尾。虽然对于更新插入边缘来说并不是绝对必要的，但在更新插入查询结尾添加 id() 步骤，可以确保服务器不会将所有边缘属性序列化回客户端，这有助于降低查询的锁定成本。

许多应用程序使用自定义顶点 ID，但让 Neptune 来生成边缘 ID。如果您不知道边缘的 ID，但您知道 from 和 to 顶点 ID，您可以使用这种查询来更新插入边缘：

```
g.mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  id()
```

对于创建边缘的步骤，mergeE() 引用的所有顶点都必须存在。

串联边缘更新插入

与顶点更新插入一样，将批量请求的 mergeE() 步骤串联在一起很简单：

```
g.mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  mergeE([(from): 'v-2', (to): 'v-3', (T.label): 'KNOWS']).
  mergeE([(from): 'v-3', (to): 'v-4', (T.label): 'KNOWS']).
  id()
```

组合使用顶点和边缘更新插入

有时，您可能想要同时更新插入两个顶点和连接它们的边缘。您可以混用此处介绍的批量示例。以下示例更新插入 3 个顶点和 2 个边缘：

```
g.mergeV([(id):'v-1']).
```

```
    option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org']).
  mergeV([(id):'v-2']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-2@example.org']).
  mergeV([(id):'v-3']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-3@example.org']).
  mergeE([(from): 'v-1', (to): 'v-2', (T.label): 'KNOWS']).
  mergeE([(from): 'v-2', (to): 'v-3', (T.label): 'KNOWS']).
 id()
```

混用更新插入和插入

有时，您可能想要同时更新插入两个顶点和连接它们的边缘。您可以混用此处介绍的批量示例。以下示例更新插入 3 个顶点和 2 个边缘：

更新插入通常一次处理一个元素。如果您坚持此处介绍的更新插入模式，则每个更新插入操作都会发出单个遍历器，这会导致后续操作仅执行一次。

但是，有时您可能想混用更新插入和插入。例如，如果您使用边缘来表示操作或事件的实例，则可能出现这种情况。请求可能会使用更新插入来确保所有必要的顶点都存在，然后使用插入来添加边缘。对于此类请求，请注意可能从每个操作中发出的遍历器数量。

考虑以下示例，它混用了更新插入和插入，以将代表事件的边缘添加到图形中：

```
// Fully optimized, but inserts too many edges
g.mergeV([(id):'v-1']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-1@example.org']).
  mergeV([(id):'v-2']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-2@example.org']).
  mergeV([(id):'v-3']).
    option(onCreate, [(label): 'PERSON', 'email': 'person-3@example.org']).
  mergeV([(T.id): 'c-1', (T.label): 'CITY', name: 'city-1']).
  V('p-1', 'p-2').
  addE('FOLLOWED').to(V('p-1')).
  V('p-1', 'p-2', 'p-3').
  addE('VISITED').to(V('c-1')).
  id()
```

该查询应插入 5 个边缘：2 个 FOLLOWED 边缘和 3 个 VISITED 边缘。但是，写入的查询插入 8 个边缘：2 个 FOLLOWED 边缘和 6 个 VISITED 边缘。其原因是插入 2 个 FOLLOWED 边缘的操作会发出 2 个遍历器，从而导致随后的插入操作（插入 3 个边缘）执行两次。

修复方法是在每个可能发出多个遍历器的操作之后添加一个 `fold()` 步骤：

```
g.mergeV([(T.id): 'v-1', (T.label): 'PERSON', email: 'person-1@example.org']).
  mergeV([(T.id): 'v-2', (T.label): 'PERSON', email: 'person-2@example.org']).
  mergeV([(T.id): 'v-3', (T.label): 'PERSON', email: 'person-3@example.org']).
  mergeV([(T.id): 'c-1', (T.label): 'CITY', name: 'city-1']).
  V('p-1', 'p-2').
  addE('FOLLOWED').
    to(V('p-1')).
  fold().
  V('p-1', 'p-2', 'p-3').
  addE('VISITED').
    to(V('c-1')).
  id()
```

在这里，我们在插入 FOLLOWED 边缘的操作之后插入了一个 `fold()` 步骤。这将生成单个遍历器，从而导致后续操作仅执行一次。

这种方法的缺点是，由于 `fold()` 未进行优化，因此查询现在尚未完全优化。`fold()` 后面的插入操作现在也不会得到优化。

如果您需要使用 `fold()` 来代表后续步骤减少遍历器的数量，请尝试对操作进行排序，以便成本最低的操作占据查询中未优化的部分。

设置基数

Neptune 中顶点属性的默认基数已设置，这意味着在使用 mergeV () 时，地图中提供的值都将赋予该基数。要使用单基数，必须明确其用法。从 TinkerPop 3.7.0 开始，有一种新的语法允许将基数作为映射的一部分提供，如以下示例所示：

```
g.mergeV([(T.id): 1234]).
  option(onMatch, ['age': single(20), 'name': single('alice'), 'city': set('miami')])
```

或者，你可以将基数设置为默认值，如下`option`所示：

```
// age and name are set to single cardinality by default
g.mergeV([(T.id): 1234]).
  option(onMatch, ['age': 22, 'name': 'alice', 'city': set('boston')], single)
```

在 3.7.0 `mergeV()` 之前的版本中，用于设置基数的选项较少。一般的方法是回退到该`property()`步骤，如下所示：

```
g.mergeV([(T.id): '1234']).
  option(onMatch, sideEffect(property(single,'age', 20).
  property(set,'city','miami')).constant([:]))
```

> **Note**
>
> 这种方法只有mergeV()在与起始步骤一起使用时才有效。因此，您将无法mergeV()在单个遍历中进行链接，因为如果传入的遍历器是图形元素，则在开始步骤mergeV()之后使用此语法的第一个遍历会产生错误。在这种情况下，你需要将mergeV()呼叫分成多个请求，每个请求都可以作为起始步骤。

使用 `fold()/coalesce()/unfold()` 进行高效的 Gremlin 更新插入

如果顶点或边缘已经存在，更新插入（或条件插入）会重复使用顶点或边缘，如果不存在，则创建它。高效的更新插入可以显著改变 Gremlin 查询的性能。

本页展示了如何使用 `fold()/coalesce()/unfold()` Gremlin 模式来进行高效的更新插入。但是，随着Neptune在引擎 TinkerPop 版本 1.2.1.0中引入的3.6.x版本的发布，在大多数情况下，新的mergeV()和mergeE()步骤更可取。此处描述的 `fold()/coalesce()/unfold()` 模式在某些复杂情况下可能仍然有用，但如果可以的话，一般使用 `mergeV()` 和 `mergeE()`，如使用 Gremlin `mergeV()` 和 `mergeE()` 步骤进行高效的更新插入中所述。

更新插入允许您编写幂等插入操作：无论您运行多少次这样的操作，总体结果都是一样的。这在高度并发的写入场景中很有用，在这种情况下，对图形的同一部分进行并发修改可能会强制一个或多个事务回滚并引发 `ConcurrentModificationException`，从而需要重试。

例如，以下查询通过首先在数据集中查找指定的顶点，然后将结果折叠到列表中来更新插入顶点。在为 `coalesce()` 步骤提供的第一次遍历中，查询随后展开此列表。如果展开的列表不为空，将从 `coalesce()` 中发出结果。但是，如果由于顶点当前不存在而 `unfold()` 返回一个空集合，则 `coalesce()` 继续计算向其提供的第二个遍历，在第二次遍历中，查询将创建缺失的顶点。

```
g.V('v-1').fold()
          .coalesce(
             unfold(),
             addV('Person').property(id, 'v-1')
                           .property('email', 'person-1@example.org')
           )
```

使用 coalesce() 的优化形式执行更新插入

Neptune 可以优化 `fold().coalesce(unfold(), ...)` 习惯用法以进行高吞吐量更新，但这种优化只有在 `coalesce()` 的两个部分都返回顶点或边缘而不返回其它部分时才有效。如果您尝试从 `coalesce()` 的任何部分返回不同的内容（例如属性），则不会进行 Neptune 优化。查询可能会成功，但其性能不如优化的版本，尤其是在处理大型数据集时。

由于未优化的更新插入查询会增加执行时间并降低吞吐量，因此值得使用 Gremlin `explain` 端点来确定更新插入查询是否经过完全优化。审核 `explain` 计划时，请查找以 `+ not converted into Neptune steps` 和 `WARNING: >>` 开头的行。例如：

```
+ not converted into Neptune steps: [FoldStep, CoalesceStep([[UnfoldStep],
 [AddEdgeSte...
WARNING: >> FoldStep << is not supported natively yet
```

这些警告可以帮助您识别查询中阻碍其完全优化的部分。

有时无法完全优化查询。在这些情况下，您应该尝试将无法优化的步骤放在查询的末尾，从而允许引擎优化尽可能多的步骤。在一些批量更新插入示例中使用了这种技术，其中对一组顶点或边缘执行了所有优化的更新插入，然后再对相同的顶点或边缘进行任何额外的、可能未优化的修改。

批处理更新插入以提高吞吐量

对于高吞吐量的写入场景，您可以将更新插入步骤串联在一起，以批量更新插入顶点和边缘。批处理减少了更新插入大量顶点和边缘的事务开销。然后，您可以通过使用多个客户端并行更新插入批量请求来进一步提高吞吐量。

根据经验，我们建议对于每个批量请求，更新插入大约 200 条记录。记录是单个顶点或边缘标签或属性。例如，具有单个标签和 4 个属性的顶点会创建 5 条记录。带有标签和单个属性的边缘会创建 2 条记录。如果您想更新插入批量顶点，每个顶点都有一个标签和 4 个属性，那么您应该从批量大小为 40 开始，因为 `200 / (1 + 4) = 40`。

您可以试验批量大小。每批 200 条记录是一个不错的起点，但理想的批量大小可能会更高或更低，具体取决于您的工作负载。但请注意，Neptune 可能会限制每个请求的 Gremlin 步骤总数。此限制未记录在案，但为了安全起见，请尽量确保您的请求包含不超过 1500 个 Gremlin 步骤。Neptune 可能会拒绝超过 1500 个步骤的大批量请求。

要提高吞吐量，您可以使用多个客户端并行批量更新插入（请参阅创建高效的多线程 Gremlin 写入）。客户端数量应与 Neptune 写入器实例上的工作线程数相同，通常是服务器上 vCPU 数量的 2

倍。例如，一个 r5.8xlarge 实例有 32 个 vCPU 和 64 个工作线程。对于使用 r5.8xlarge 的高吞吐量写入场景，您将使用 64 个客户端并行向 Neptune 写入批量更新插入。

每个客户端都应提交批量请求，等待请求完成后，再提交另一个请求。尽管多个客户端并行运行，但每个客户端都以串行方式提交请求。这样可以确保服务器获得稳定的请求流，这些请求会占用所有工作线程，而不会淹没服务器端的请求队列（请参阅调整 Neptune 数据库集群中数据库实例的大小）。

尽量避免生成多个遍历器的步骤

当 Gremlin 步骤执行时，它会接受一个传入的遍历器，并发出一个或多个输出遍历器。一个步骤发出的遍历器的数量决定了执行下一步的次数。

通常，在执行批量操作时，您希望每个操作（例如更新插入顶点 A）执行一次，以便操作序列如下所示：更新插入顶点 A，接着更新插入顶点 B，然后更新插入顶点 C，依此类推。只要一个步骤只创建或修改一个元素，它就会只发出一个遍历器，而代表下一个操作的步骤只执行一次。另一方面，如果一个操作创建或修改了多个元素，则它会发出多个遍历器，这反过来又会导致后续步骤多次执行，每个发出的遍历器执行一次。这可能会导致数据库执行不必要的额外工作，在某些情况下，还可能导致创建不想要的额外顶点、边缘或属性值。

诸如 g.V().addV() 的查询就是一个例子，说明事情如何出错。这个简单的查询会为在图形中找到的每个顶点添加一个顶点，因为 V() 会为图形中的每个顶点发出一个遍历器，而每个遍历器都会触发对 addV() 的调用。

有关处理可能发出多个遍历器的操作的方法，请参阅混用更新插入和插入。

更新插入顶点

您可以使用顶点 ID 来确定相应的顶点是否存在。这是首选方法，因为 Neptune 针对围绕 ID 的高度并发用例优化了更新插入。例如，以下查询会创建一个具有给定顶点 ID 的顶点（如果该顶点尚不存在），如果已存在，则重用它：

```
g.V('v-1')
  .fold()
   .coalesce(unfold(),
             addV('Person').property(id, 'v-1')
                           .property('email', 'person-1@example.org'))
   .id()
```

请注意，此查询以 id() 步骤结尾。虽然对于更新插入顶点来说并不是绝对必要的，但在更新插入查询结尾添加 id() 步骤，可以确保服务器不会将所有顶点属性序列化回客户端，这有助于降低查询的锁定成本。

或者，您可以使用顶点属性来确定顶点是否存在：

```
g.V()
 .hasLabel('Person')
 .has('email', 'person-1@example.org')
 .fold()
 .coalesce(unfold(),
           addV('Person').property('email', 'person-1@example.org'))
 .id()
```

如果可能，请使用您自己的用户提供的 ID 来创建顶点，并使用这些 ID 来确定在更新插入操作期间是否存在顶点。这使得 Neptune 可以围绕 ID 优化更新插入。在高度并发的修改场景中，基于 ID 的更新插入可能比基于属性的更新插入效率要高得多。

串联顶点更新插入

您可以将顶点更新插入串联在一起以批量插入它们：

```
g.V('v-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-1')
                         .property('email', 'person-1@example.org'))
 .V('v-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-2')
                         .property('email', 'person-2@example.org'))
 .V('v-3')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-3')
                         .property('email', 'person-3@example.org'))
 .id()
```

更新插入边缘

您可以使用边缘 ID 来更新插入边缘，就像使用自定义顶点 ID 更新插入顶点一样。同样，这是首选方法，因为它允许 Neptune 优化查询。例如，如果边缘尚不存在，则以下查询会根据其边缘 ID 创建该边缘，如果存在，则重用该边缘。如果该查询需要创建新边缘，它还会使用 `from` 和 `to` 顶点的 ID。

```
g.E('e-1')
```

```
  .fold()
  .coalesce(unfold(),
            addE('KNOWS').from(V('v-1'))
                         .to(V('v-2'))
                         .property(id, 'e-1'))
  .id()
```

许多应用程序使用自定义顶点 ID，但让 Neptune 来生成边缘 ID。如果您不知道边缘的 ID，但您知道 `from` 和 `to` 顶点 ID，您可以使用此公式来更新插入边缘：

```
g.V('v-1')
  .outE('KNOWS')
  .where(inV().hasId('v-2'))
  .fold()
  .coalesce(unfold(),
            addE('KNOWS').from(V('v-1'))
                         .to(V('v-2')))
  .id()
```

请注意，`where()` 子句中的顶点步骤应该是 `inV()`（或者，如果您曾经使用 `inE()` 来查找边缘，则为 `outV()`），而不是 `otherV()`。请勿在此处使用 `otherV()`，否则查询将无法优化，性能会受到影响。例如，Neptune 不会优化以下查询：

```
// Unoptimized upsert, because of otherV()
g.V('v-1')
  .outE('KNOWS')
  .where(otherV().hasId('v-2'))
  .fold()
  .coalesce(unfold(),
            addE('KNOWS').from(V('v-1'))
                         .to(V('v-2')))
  .id()
```

如果您不知道前面的边缘或顶点 ID，可以使用顶点属性更新插入：

```
g.V()
  .hasLabel('Person')
  .has('name', 'person-1')
  .outE('LIVES_IN')
  .where(inV().hasLabel('City').has('name', 'city-1'))
  .fold()
  .coalesce(unfold(),
```

```
          addE('LIVES_IN').from(V().hasLabel('Person')
                                   .has('name', 'person-1'))
                          .to(V().hasLabel('City')
                                 .has('name', 'city-1')))
 .id()
```

与顶点更新插入一样，最好通过边缘 ID 或 from 和 to 顶点 ID 使用基于 ID 的边缘更新插入，而不是基于属性的更新插入，这样 Neptune 就可以完全优化更新插入。

检查 **from** 和 **to** 顶点是否存在

请注意创建新边缘的步骤的构造：addE().from().to()。这种构造可确保查询检查 from 和 to 顶点是否存在。如果其中任何一个顶点不存在，则查询将返回如下错误：

```
{
  "detailedMessage": "Encountered a traverser that does not map to a value for child...
  "code": "IllegalArgumentException",
  "requestId": "..."
}
```

如果 from 或 to 顶点可能不存在，则应尝试在它们之间更新插入边缘之前，更新插入它们。请参阅 组合使用顶点和边缘更新插入。

还有另一种构造可用来创建您不应使用的边缘：V().addE().to()。只有当 from 顶点存在时，它才会添加边缘。如果 to 顶点不存在，查询会生成错误，如前所述，但如果 from 顶点不存在，则它将无法插入边缘且不发出提示，而不会生成任何错误。例如，如果 from 顶点不存在，则以下更新插入会在不更新插入边缘的情况下完成：

```
// Will not insert edge if from vertex does not exist
g.V('v-1')
 .outE('KNOWS')
 .where(inV().hasId('v-2'))
 .fold()
 .coalesce(unfold(),
          V('v-1').addE('KNOWS')
                  .to(V('v-2')))
 .id()
```

串联边缘更新插入

如果要将边缘更新插入串联在一起以创建批量请求，则即使您已经知道边缘 ID，也必须通过顶点查找来开始每个更新插入。

如果您已经知道要更新插入的边缘的 ID 以及 from 和 to 顶点的 ID，则可以使用以下公式：

```
g.V('v-1')
 .outE('KNOWS')
 .hasId('e-1')
 .fold()
 .coalesce(unfold(),
           V('v-1').addE('KNOWS')
                   .to(V('v-2'))
                   .property(id, 'e-1'))
 .V('v-3')
 .outE('KNOWS')
 .hasId('e-2').fold()
 .coalesce(unfold(),
           V('v-3').addE('KNOWS')
                   .to(V('v-4'))
                   .property(id, 'e-2'))
 .V('v-5')
 .outE('KNOWS')
 .hasId('e-3')
 .fold()
 .coalesce(unfold(),
           V('v-5').addE('KNOWS')
                   .to(V('v-6'))
                   .property(id, 'e-3'))
 .id()
```

也许最常见的批量边缘更新插入场景是您知道 from 和 to 顶点 ID，但不知道要更新插入的边缘的 ID。在这种情况下，请使用以下公式：

```
g.V('v-1')
 .outE('KNOWS')
 .where(inV().hasId('v-2'))
 .fold()
 .coalesce(unfold(),
           V('v-1').addE('KNOWS')
                   .to(V('v-2')))

 .V('v-3')
 .outE('KNOWS')
 .where(inV().hasId('v-4'))
 .fold()
 .coalesce(unfold(),
```

```
          V('v-3').addE('KNOWS')
                  .to(V('v-4')))
 .V('v-5')
 .outE('KNOWS')
 .where(inV().hasId('v-6'))
 .fold()
 .coalesce(unfold(),
          V('v-5').addE('KNOWS').to(V('v-6')))
 .id()
```

如果您知道要更新插入的边缘的 ID，但不知道 `from` 和 `to` 顶点的 ID（这很少见），您可以使用以下公式：

```
g.V()
 .hasLabel('Person')
 .has('email', 'person-1@example.org')
 .outE('KNOWS')
 .hasId('e-1')
 .fold()
 .coalesce(unfold(),
          V().hasLabel('Person')
             .has('email', 'person-1@example.org')
             .addE('KNOWS')
             .to(V().hasLabel('Person')
                    .has('email', 'person-2@example.org'))
                    .property(id, 'e-1'))
 .V()
 .hasLabel('Person')
 .has('email', 'person-3@example.org')
 .outE('KNOWS')
 .hasId('e-2')
 .fold()
 .coalesce(unfold(),
          V().hasLabel('Person')
             .has('email', 'person-3@example.org')
             .addE('KNOWS')
             .to(V().hasLabel('Person')
                    .has('email', 'person-4@example.org'))
             .property(id, 'e-2'))
 .V()
 .hasLabel('Person')
 .has('email', 'person-5@example.org')
 .outE('KNOWS')
```

```
  .hasId('e-1')
  .fold()
  .coalesce(unfold(),
            V().hasLabel('Person')
               .has('email', 'person-5@example.org')
               .addE('KNOWS')
               .to(V().hasLabel('Person')
                      .has('email', 'person-6@example.org'))
                      .property(id, 'e-3'))
  .id()
```

组合使用顶点和边缘更新插入

有时，您可能想要同时更新插入两个顶点和连接它们的边缘。您可以混用此处介绍的批量示例。以下示例更新插入 3 个顶点和 2 个边缘：

```
g.V('p-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-1')
                          .property('email', 'person-1@example.org'))
  .V('p-2')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'p-2')
                          .property('name', 'person-2@example.org'))
  .V('c-1')
  .fold()
  .coalesce(unfold(),
            addV('City').property(id, 'c-1')
                        .property('name', 'city-1'))
  .V('p-1')
  .outE('LIVES_IN')
  .where(inV().hasId('c-1'))
  .fold()
  .coalesce(unfold(),
            V('p-1').addE('LIVES_IN')
                    .to(V('c-1')))
  .V('p-2')
  .outE('LIVES_IN')
  .where(inV().hasId('c-1'))
  .fold()
  .coalesce(unfold(),
```

```
          V('p-2').addE('LIVES_IN')
                  .to(V('c-1')))
 .id()
```

混用更新插入和插入

有时，您可能想要同时更新插入两个顶点和连接它们的边缘。您可以混用此处介绍的批量示例。以下示例更新插入 3 个顶点和 2 个边缘：

更新插入通常一次处理一个元素。如果您坚持此处介绍的更新插入模式，则每个更新插入操作都会发出单个遍历器，这会导致后续操作仅执行一次。

但是，有时您可能想混用更新插入和插入。例如，如果您使用边缘来表示操作或事件的实例，则可能出现这种情况。请求可能会使用更新插入来确保所有必要的顶点都存在，然后使用插入来添加边缘。对于此类请求，请注意可能从每个操作中发出的遍历器数量。

考虑以下示例，它混用了更新插入和插入，以将代表事件的边缘添加到图形中：

```
// Fully optimized, but inserts too many edges
g.V('p-1')
 .fold()
 .coalesce(unfold(),
          addV('Person').property(id, 'p-1')
                        .property('email', 'person-1@example.org'))
 .V('p-2')
 .fold()
 .coalesce(unfold(),
          addV('Person').property(id, 'p-2')
                        .property('name', 'person-2@example.org'))
 .V('p-3')
 .fold()
 .coalesce(unfold(),
          addV('Person').property(id, 'p-3')
                        .property('name', 'person-3@example.org'))
 .V('c-1')
 .fold()
 .coalesce(unfold(),
          addV('City').property(id, 'c-1')
                      .property('name', 'city-1'))
 .V('p-1', 'p-2')
 .addE('FOLLOWED')
 .to(V('p-1'))
```

```
 .V('p-1', 'p-2', 'p-3')
 .addE('VISITED')
 .to(V('c-1'))
 .id()
```

该查询应插入 5 个边缘：2 个 FOLLOWED 边缘和 3 个 VISITED 边缘。但是，写入的查询插入 8 个边缘：2 个 FOLLOWED 边缘和 6 个 VISITED 边缘。其原因是插入 2 个 FOLLOWED 边缘的操作会发出 2 个遍历器，从而导致随后的插入操作（插入 3 个边缘）执行两次。

修复方法是在每个可能发出多个遍历器的操作之后添加一个 `fold()` 步骤：

```
g.V('p-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'p-1')
                         .property('email', 'person-1@example.org'))
 .V('p-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'p-2').
                         .property('name', 'person-2@example.org'))
 .V('p-3')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'p-3').
                         .property('name', 'person-3@example.org'))
 .V('c-1')
 .fold().
 .coalesce(unfold(),
            addV('City').property(id, 'c-1').
                        .property('name', 'city-1'))
 .V('p-1', 'p-2')
 .addE('FOLLOWED')
 .to(V('p-1'))
 .fold()
 .V('p-1', 'p-2', 'p-3')
 .addE('VISITED')
 .to(V('c-1')).
 .id()
```

在这里，我们在插入 FOLLOWED 边缘的操作之后插入了一个 `fold()` 步骤。这将生成单个遍历器，从而导致后续操作仅执行一次。

这种方法的缺点是，由于 fold() 未进行优化，因此查询现在尚未完全优化。fold() 后面的插入操作现在不会得到优化。

如果您需要使用 fold() 来代表后续步骤减少遍历器的数量，请尝试对操作进行排序，以便成本最低的操作占据查询中未优化的部分。

修改现有顶点和边缘的更新插入

有时，如果顶点或边缘不存在，则需要创建顶点或边缘，然后向其添加或更新属性，无论它是新的还是现有的顶点或边缘。

要添加或修改属性，请使用 property() 步骤。在 coalesce() 步骤之外使用此步骤。如果您尝试修改 coalesce() 步骤内现有顶点或边缘的属性，Neptune 查询引擎可能不会对查询进行优化。

以下查询在每个更新插入的顶点上添加或更新计数器属性。每个 property() 步骤都具有单个基数，以确保新值替换任何现有值，而不是添加到一组现有值中。

```
g.V('v-1')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-1')
                         .property('email', 'person-1@example.org'))
 .property(single, 'counter', 1)
 .V('v-2')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-2')
                         .property('email', 'person-2@example.org'))
 .property(single, 'counter', 2)
 .V('v-3')
 .fold()
 .coalesce(unfold(),
           addV('Person').property(id, 'v-3')
                         .property('email', 'person-3@example.org'))
 .property(single, 'counter', 3)
 .id()
```

如果您有一个适用于所有已更新插入的元素的属性值（例如 lastUpdated 时间戳值），则可以在查询结尾添加或更新它：

```
g.V('v-1')
```

```
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org'))
  .V('v-2').
  .fold().
  .coalesce(unfold(),
            addV('Person').property(id, 'v-2')
                          .property('email', 'person-2@example.org'))
  .V('v-3')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-3')
                          .property('email', 'person-3@example.org'))
  .V('v-1', 'v-2', 'v-3')
  .property(single, 'lastUpdated', datetime('2020-02-08'))
  .id()
```

如果还有其它条件可以确定是否应进一步修改顶点或边缘，则可以使用 has() 步骤来筛选要对其应用修改的元素。以下示例使用 has() 步骤根据其 version 属性的值筛选已更新插入的顶点。然后，对于 version 小于 3 的任何顶点，该查询将该顶点的 version 更新为 3：

```
g.V('v-1')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-1')
                          .property('email', 'person-1@example.org')
                          .property('version', 3))
  .V('v-2')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-2')
                          .property('email', 'person-2@example.org')
                          .property('version', 3))
  .V('v-3')
  .fold()
  .coalesce(unfold(),
            addV('Person').property(id, 'v-3')
                          .property('email', 'person-3@example.org')
                          .property('version', 3))
  .V('v-1', 'v-2', 'v-3')
  .has('version', lt(3))
```

```
  .property(single, 'version', 3)
  .id()
```

使用 Gremlin **explain** 分析 Neptune 查询执行

Amazon Neptune 添加了名为 explain 的 Gremlin 特征。此特征是一种自助式工具，用于了解 Neptune 引擎所采用的执行方法。您可以通过将 `explain` 参数添加到提交 Gremlin 查询的 HTTP 调用来调用它。

`explain` 功能提供了有关查询执行计划的逻辑结构的信息。您可以使用此信息来识别潜在的评估和执行瓶颈并调整查询，如调整 Gremlin 查询中所述。您还可以使用查询提示来改进查询执行计划。

> **Note**
>
> 从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始，此特征可用。

主题

- 了解 Gremlin 查询在 Neptune 中的工作方式
- 在 Neptune 中使用 Gremlin explain API
- Neptune 中的 Gremlin profile API
- 使用 explain 和 profile 调整 Gremlin 查询
- Amazon Neptune 中的原生 Gremlin 步骤支持

了解 Gremlin 查询在 Neptune 中的工作方式

要充分利用 Amazon Neptune 中的 `explain` 和 `profile` 报告，了解一些有关 Gremlin 查询的背景信息将很有帮助。

主题

- Neptune 中的 Gremlin 语句
- Neptune 如何使用语句索引处理 Gremlin 查询
- 在 Neptune 中如何处理 Gremlin 查询

Neptune 中的 Gremlin 语句

Amazon Neptune 中的属性图数据由四个位置（四元组）语句组成。这些语句中的每一个都代表属性图数据的单个原子单元。有关更多信息，请参阅 Neptune 图形数据模型。与资源描述框架 (RDF) 数据模型类似，这四个位置如下所述：

- `subject (S)`
- `predicate (P)`
- `object (O)`
- `graph (G)`

每个语句都是对一个或多个资源的断言。例如，一个语句可以断言两个资源之间是否存在关系，或者可以将一个属性（键/值对）附加到某个资源。

您可以将谓词视为语句的动词，以描述关系或属性的类型。对象是关系的目标，或者是属性的值。图形位置可选，可通过多种不同方式使用。对于 Neptune 属性图 (PG) 数据，此项可以不使用（空图），或是用于表示边缘的标识符。一组具有共享资源标识符的语句创建一个图形。

Neptune 属性图数据模型中有三类语句：

主题

- Gremlin 顶点标签语句
- Gremlin 边缘语句
- Gremlin 属性语句

Gremlin 顶点标签语句

Neptune 中的顶点标签语句有两个作用：

- 跟踪顶点的标签。
- 只要存在一条此类语句，即暗示图中存在特定顶点。

这些语句的主语是顶点标识符，宾语是标签，两者均由用户指定。您对这些语句使用特殊的固定谓词（显示为 `<~label>`）和默认图标识符（空图，显示为 `<~>`）。

例如，请考虑以下 addV 遍历。

```
g.addV("Person").property(id, "v1")
```

这种遍历导致将以下语句添加到图中。

```
StatementEvent[Added(<v1> <~label> <Person> <~>) .]
```

Gremlin 边缘语句

Gremlin 边缘语句用于暗示 Neptune 图形中两个顶点之间存在边缘。边缘语句的主语 (S) 是源 `from` 顶点。谓词 (P) 是用户提供的边缘标签。宾语 (O) 是目标 `to` 顶点。图 (G) 是用户提供的边缘标识符。

例如，请考虑以下 `addE` 遍历。

```
g.addE("knows").from(V("v1")).to(V("v2")).property(id, "e1")
```

这种遍历导致将以下语句添加到图中。

```
StatementEvent[Added(<v1> <knows> <v2> <e1>) .]
```

Gremlin 属性语句

Neptune 中的 Gremlin 属性语句对顶点或边缘的单个属性值进行断言。主语是用户提供的顶点或边缘标识符。谓词是属性名称（键），宾语是单个属性值。图 (G) 还是默认的图标识符，即空图，显示为 `<~>`。

考虑以下示例。

```
g.V("v1").property("name", "John")
```

该语句导致以下结果。

```
StatementEvent[Added(<v1> <name> "John" <~>) .]
```

属性语句与其他语句的不同之处在于，属性语句的宾语是基元值（`string`、`date`、`byte`、`short`、`int`、`long`、`float` 或 `double`）。它们的宾语不是可用作其他断言的主语的资源标识符。

对于多属性，集合中的每个单属性值接收各自的语句。

```
g.V("v1").property(set, "phone", "956-424-2563").property(set, "phone", "956-354-3692
 (tel:9563543692)")
```

这将产生以下结果。

```
StatementEvent[Added(<v1> <phone> "956-424-2563" <~>) .]
StatementEvent[Added(<v1> <phone> "956-354-3692" <~>) .]
```

Neptune 如何使用语句索引处理 Gremlin 查询

在 Amazon Neptune 中，可以通过三种语句索引访问语句，如[如何在 Neptune 中为语句编制索引]中所述。Neptune 从 Gremlin 查询中提取语句模式，其中一些位置是已知的，其余的则留待索引搜索发现。

Neptune 假定属性图架构的大小不大。这意味着不同边缘标签和属性名称的数量相当少，导致不同谓词的总数较少。Neptune 在单独的索引中跟踪不同的谓词。它使用该谓词缓存来进行 `{ all P x POGS }` 的联合扫描，而不是使用 OSGP 索引。避免使用反向遍历 OSGP 索引可节省存储空间和加载吞吐量。

Neptune Gremlin Explain/Profile API 让您能够获取图形中的谓词数。然后，您可以确定您的应用程序是否使 Neptune 的“属性图架构较小”的假设失效。

以下示例有助于说明 Neptune 如何使用索引来处理 Gremlin 查询。

问题：顶点 **v1** 有哪些标签？

```
Gremlin code:      g.V('v1').label()
Pattern:           (<v1>, <~label>, ?, ?)
Known positions:   SP
Lookup positions:  OG
Index:             SPOG
Key range:         <v1>:<~label>:*
```

问题：顶点 **v1** 有哪些“已知”的出边？

```
Gremlin code:      g.V('v1').out('knows')
Pattern:           (<v1>, <knows>, ?, ?)
Known positions:   SP
Lookup positions:  OG
Index:             SPOG
Key range:         <v1>:<knows>:*
```

问题：哪些顶点具有 **Person** 顶点标签？

```
Gremlin code:      g.V().hasLabel('Person')
Pattern:           (?, <~label>, <Person>, <~>)
Known positions:   POG
Lookup positions:  S
Index:             POGS
Key range:         <~label>:<Person>:<~>:*
```

问题：给定边缘 **e1** 有哪些源/目标顶点？

```
Gremlin code:      g.E('e1').bothV()
Pattern:           (?, ?, ?, <e1>)
Known positions:   G
Lookup positions:  SPO
Index:             GPSO
Key range:         <e1>:*
```

Neptune 所没有 的一个声明索引是反向遍历 OSGP 索引。该索引可用于收集所有边缘标签的所有入边，如以下示例所示。

问题：什么是传入的相邻顶点 **v1**？

```
Gremlin code:      g.V('v1').in()
Pattern:           (?, ?, <v1>, ?)
Known positions:   O
Lookup positions:  SPG
Index:             OSGP  // <-- Index does not exist
```

在 Neptune 中如何处理 Gremlin 查询

在 Amazon Neptune 中，可以通过一系列模式来表示更复杂的遍历，这些模式基于命名变量的定义创建关系，而命名变量可以在模式之间共享以创建联接。如以下示例所示。

问题：顶点 **v1** 有哪些两跳邻域？

```
Gremlin code:      g.V(‘v1’).out('knows').out('knows').path()
Pattern:           (?1=<v1>, <knows>, ?2, ?) X Pattern(?2, <knows>, ?3, ?)

The pattern produces a three-column relation (?1, ?2, ?3) like this:
                   ?1     ?2     ?3
                   ================
```

```
v1      v2      v3
v1      v2      v4
v1      v5      v6
```

通过在两个模式之间共享 ?2 变量（在第一个模式的 O 位置和第二个模式的 S 位置），可以创建从一跳邻域到二跳邻域的联接。每个 Neptune 解都有三个命名变量的绑定，这些变量可用于重新创建 TinkerPopTraverser（包括路径信息）。

Gremlin 查询处理的第一步是将查询解析为 TinkerPop Traversal 对象，该对象由一系列步骤组成。TinkerPop 这些步骤是开源 Apache TinkerPop 项目的一部分，既是逻辑运算符，也是物理运算符，在参考实现中构成 Gremlin 遍历。它们都用于表示查询模型。它们是可执行的运算符，可以根据其代表的运算符的语义生成解决方案。例如，`.V()`既由表示又由执行 TinkerPop GraphStep。

由于这些 off-the-shelf TinkerPop 步骤是可执行的，因此这样的 TinkerPop Traversal 可以执行任何 Gremlin 查询并生成正确的答案。但是，当针对大型图表执行时， TinkerPop 步骤有时效率低下且速度很慢。Neptune 不使用这些步骤，而是尝试将该遍历转换为由模式组组成的声明形式（如前所述）。

当前，Neptune 的原生查询引擎仅支持部分 Gremlin 运算符（步骤）。因此，它尝试将尽可能多的步骤折叠为一个 `NeptuneGraphQueryStep`，其中包含已转换的所有步骤的声明性逻辑查询计划。理想情况下，所有步骤都将转换。但是，当遇到无法转换的步骤时，Neptune 会脱离原生执行，并将所有查询执行从该点推迟到这些步骤。 TinkerPop 它不会尝试穿插进行本机执行。

将步骤转换成逻辑查询计划后，Neptune 运行一系列查询优化器，以根据静态分析和估计基数来重写查询计划。优化器执行多种操作，例如根据范围计数对运算符进行重新排序、删除不必要或多余的运算符、重新排列筛选条件、将运算符推入不同的组等。

生成优化的查询计划后，Neptune 创建物理运算符的管道来执行查询。这包括从语句索引中读取数据、执行各种类型的联接、筛选、排序等。管道生成解流，然后将其转换回 TinkerPop Traverser 对象流。

查询结果的序列化

Amazon Neptune 目前依靠 TinkerPop 响应消息序列化器将查询结果（TinkerPop Traversers）转换为序列化数据，然后通过电线发送回客户端。这些序列化格式往往很冗长。

例如，要序列化 `g.V().limit(1)` 等顶点查询的结果，Neptune 查询引擎必须执行一次搜索来生成查询结果。但是，`GraphSON` 序列化程序将执行大量额外的搜索，才能将顶点打包为序列化格式。它必须执行一次搜索来获取标签，执行一次搜索来获取属性键，并对顶点的每个属性键执行一次搜索来获取每个键的所有值。

某些序列化格式效率更高，但是所有序列化格式都需要进行额外的搜索。此外， TinkerPop 序列化程序不会尽量避免重复搜索，这通常会导致不必要地重复许多搜索。

因此，在编写查询时，只询问所需的信息极为重要。例如，`g.V().limit(1).id()` 将仅返回顶点 ID，并消除所有其他序列化程序搜索。Neptune 中的 Gremlin `profile` API 允许您查看在查询执行和序列化期间进行了多少次搜索调用。

在 Neptune 中使用 Gremlin **`explain`** API

Amazon Neptune Gremlin `explain` API 返回将在运行指定查询时执行的查询计划。由于该 API 实际上并未运行查询，因此几乎可即时返回计划。

它与 TinkerPop .explain () 步骤的不同之处在于能够报告特定于 Neptune 引擎的信息。

Gremlin **`explain`** 报告中包含的信息

`explain` 报告包含以下信息：

- 要求的查询字符串。
- 原始遍历。这是通过将查询字符串解析为步骤而生成的 TinkerPop Traversal 对象。 TinkerPop 它等同于在针对的查询`.explain()`上运行所产生的原始查询 TinkerPop TinkerGraph。
- 转换的遍历。这是通过将遍历转换为海王星逻辑查询计划表示形式而生成的 Neptune Trav TinkerPop ersal。在许多情况下，整个 TinkerPop 遍历会转换为两个 Neptune 步骤：一个执行整个查询 (`NeptuneGraphQueryStep`)，另一个将海王星查询引擎的输出转换回 Traversers ()。TinkerPop `NeptuneTraverserConverterStep`
- 优化的遍历。这是 Neptune 查询计划的优化版本，已经通过一系列静态减负优化器运行它，这些优化器基于静态分析和估计基数重写查询。优化器执行多种操作，例如根据范围计数对运算符进行重新排序、删除不必要或多余的运算符、重新排列筛选条件、将运算符推入不同的组等。
- 谓词计数。如前所述，出于 Neptune 索引策略的原因，具有大量不同谓词可能会导致性能问题。对于使用没有边缘标签（`.in` 或 `.both`）的反向遍历运算符的查询，这一点尤为明显。如果使用了此类运算符并且谓词数量足够多，`explain` 报告将显示警告消息。
- DFE 信息。启用 DFE 替代引擎后，以下遍历组件可能会出现在优化的遍历中：
  - **`DFEStep`** – 在包含子 `DFENode` 的遍历中，Neptune 优化的 DFE 步骤。`DFEStep` 表示查询计划中在 DFE 引擎中执行的部分。
  - **`DFENode`** – 包含中间表示（作为一个或多个子 `DFEJoinGroupNodes`）。
  - **`DFEJoinGroupNode`** – 表示一个或多个 `DFENode` 或 `DFEJoinGroupNode` 元素的联接。
  - **`NeptuneInterleavingStep`** – 在包含子 `DFEStep` 的遍历中，Neptune 优化的 DFE 步骤。

还包含一个 stepInfo 元素，其中包含有关遍历的信息，例如边界元素、使用的路径元素等。此信息用于处理子 DFEStep。

要查明 DFE 是否正在计算您的查询，一种简单的方法是检查 explain 输出中是否包含 DFEStep。任何不属于的遍历部分都不会由 DFE 执行，而是由引擎执行。DFEStep TinkerPop

有关示例报告，请参阅启用 DFE 的示例。

Gremlin **explain** 语法

explain API 的语法与查询的 HTTP API 相同，不同之处在于，它使用 /gremlin/explain 作为终端节点而不是 /gremlin，如以下示例所示。

```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d
 '{"gremlin":"g.V().limit(1)"}'
```

前面的查询将产生以下输出。

```
=======================================================
                Neptune Gremlin Explain
=======================================================

Query String
============
g.V().limit(1)

Original Traversal
==================
[GraphStep(vertex,[]), RangeGlobalStep(0,1)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
        }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]
```

```
Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, finishers=[limit(1)], annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 18
```

未转换的步骤 TinkerPop

理想情况下，遍历中的所有 TinkerPop 步骤都具有原生 Neptune 运算符覆盖范围。如果不是这种情况，Neptune会因为操作员覆盖范围存在差距而退回 TinkerPop 分步执行。如果遍历使用 Neptune 尚无原生运算符的步骤，`explain` 报告将显示一条警告，指出缺失运算符的位置。

当遇到没有相应原生 Neptune 运算符的步骤时，即使后续步骤确实有原生 Neptune 运算符，也将使用 TinkerPop 步骤运行从该点开始的整个遍历。

这种情况的例外是调用 Neptune 全文搜索时。 NeptuneSearchStep 实现了没有原生等效项的步骤作为全文搜索步骤。

查询中的所有步骤都具有原生等效运算符的 **`explain`** 输出示例

下面是一个示例查询 `explain` 报告，其中所有步骤都有原生等效运算符：

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============
g.V().out()

Original Traversal
```

```
==================
[GraphStep(vertex,[]), VertexStep(OUT,vertex)]

Converted Traversal
==================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
==================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 18
```

查询中的部分步骤没有本机等效运算符的示例

Neptune 原生处理 `GraphStep` 和 `VertexStep`，但如果引入 `FoldStep` 和 `UnfoldStep`，则结果 `explain` 输出会有所不同：

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************
```

```
Query String
============
g.V().fold().unfold().out()

Original Traversal
==================
[GraphStep(vertex,[]), FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=3}
    },
    NeptuneTraverserConverterStep,
    NeptuneMemoryTrackerStep
]
+ not converted into Neptune steps: [FoldStep, UnfoldStep, VertexStep(OUT,vertex)]

WARNING: >> FoldStep << is not supported natively yet
```

在这种情况下，FoldStep 将使遍历跳出本机执行。同时，后续的 VertexStep 也不再在本机处理，因为它出现在 Fold/Unfold 步骤之后。

为了提高性能和节省成本，请务必尝试制定遍历公式，以便在 Neptune 查询引擎中以本机方式完成尽可能多的工作，而不是通过分步实现。 TinkerPop

使用 Neptune 的查询示例 full-text-search

以下查询使用 Neptune 全文搜索：

```
g.withSideEffect("Neptune#fts.endpoint", "some_endpoint")
  .V()
  .tail(100)
  .has("Neptune#fts mark*")
  -------
  .has("name", "Neptune#fts mark*")
  .has("Person", "name", "Neptune#fts mark*")
```

`.has("name", "Neptune#fts mark*")` 部分将搜索限制为带有 name 的顶点，而 `.has("Person", "name", "Neptune#fts mark*")` 将搜索限制为带有 name 和标签 Person 的顶点。这将导致 explain 报告中出现以下遍历：

```
Final Traversal
[NeptuneGraphQueryStep(Vertex) {
    JoinGroupNode {
        PatternNode[(?1, termid(1,URI), ?2, termid(0,URI)) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
    }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
}, NeptuneTraverserConverterStep, NeptuneTailGlobalStep(10),
 NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=some_endpoint}
    }
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=some_endpoint}
    }
}]
```

启用 DFE 时使用 **explain** 的示例

以下是启用 DFE 备用查询引擎时 explain 报告的示例：

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
```

```
============

g.V().as("a").out().has("name", "josh").out().in().where(eq("a"))


Original Traversal
==================
[GraphStep(vertex,[])@[a], VertexStep(OUT,vertex), HasStep([name.eq(josh)]),
 VertexStep(OUT,vertex), VertexStep(IN,vertex), WherePredicateStep(eq(a))]

Converted Traversal
===================
Neptune steps:
[
    DFEStep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
          DFEPatternNode[(?1, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?2,
 <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>) . project DISTINCT[?1]
 {rangeCountEstimate=unknown}],
          DFEPatternNode[(?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters=(!
= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}]
        }, {rangeCountEstimate=unknown}
        ]
      } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]
    } ,
    NeptuneTraverserConverterDFEStep
]
+ not converted into Neptune steps: HasStep([name.eq(josh)]),
Neptune steps:
[
    NeptuneInterleavingStep {
      StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
 pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=0ms],
      DFEStep(Vertex) {
        DFENode {
          DFEJoinGroupNode[ children={
            DFEPatternNode[(?7, ?8, ?9, ?10) . project ALL[?7, ?9]
 graphFilters=(!= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}],
            DFEPatternNode[(?12, ?11, ?9, ?13) . project ALL[?9, ?12]
 graphFilters=(!= <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph> . ),
 {rangeCountEstimate=unknown}]
```

```
          }, {rangeCountEstimate=unknown}
          ]
        } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]
      }
    }
]
+ not converted into Neptune steps: WherePredicateStep(eq(a)),
Neptune steps:
[
    DFECleanupStep
]


Optimized Traversal
===================
Neptune steps:
[
    DFEStep(Vertex) {
      DFENode {
        DFEJoinGroupNode[ children={
          DFEPatternNode[(?1, ?3, ?4, ?5) . project ALL[?1, ?4] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]
        }, {rangeCountEstimate=unknown}
        ]
      } [Vertex(?1):GraphStep@[a], Vertex(?4):VertexStep]
    } ,
    NeptuneTraverserConverterDFEStep
]
+ not converted into Neptune steps: NeptuneHasStep([name.eq(josh)]),
Neptune steps:
[
    NeptuneMemoryTrackerStep,
    NeptuneInterleavingStep {
      StepInfo[joinVars=[?7, ?1], frontierElement=Vertex(?7):HasStep,
 pathElements={a=(last,Vertex(?1):GraphStep@[a])}, listPathElement={}, indexTime=0ms],
      DFEStep(Vertex) {
        DFENode {
          DFEJoinGroupNode[ children={
            DFEPatternNode[(?7, ?8, ?9, ?10) . project ALL[?7, ?9] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}],
            DFEPatternNode[(?12, ?11, ?9, ?13) . project ALL[?9, ?12] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=9223372036854775807}]
          }, {rangeCountEstimate=unknown}
          ]
```

```
        } [Vertex(?9):VertexStep, Vertex(?12):VertexStep]
      }
    }
]
+ not converted into Neptune steps: WherePredicateStep(eq(a)),
Neptune steps:
[
    DFECleanupStep
]


WARNING: >> [NeptuneHasStep([name.eq(josh)]), WherePredicateStep(eq(a))] << (or one of
 the children for each step) is not supported natively yet

Predicates
==========
# of predicates: 8
```

有关报告中特定于 DFE 的部分的说明，请参阅explain 中的信息。

Neptune 中的 Gremlin **profile** API

Neptune Gremlin `profile` API 可运行指定的 Gremlin 遍历、收集有关运行的各种指标并生成分析报告作为输出。

> **Note**
>
> 从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始，此特征可用。

它与 TinkerPop .profile () 步骤的不同之处在于能够报告特定于 Neptune 引擎的信息。

分析报告包括有关查询计划的以下信息：

- 物理运算符管道
- 查询执行和序列化的索引操作
- 结果大小

`profile` API 使用的是 HTTP API 查询语法的扩展版本，以 `/gremlin/profile` 而不是 `/gremlin` 作为终端节点。

Neptune Gremlin **profile** 特定的参数

- profile.results – boolean，允许的值：TRUE 和 FALSE，默认值：TRUE。

  如果为 true，则收集查询结果并显示为 profile 报告的一部分。如果为 false，则仅显示结果计数。

- profile.chop – int，默认值：250。

  如果不为零，结果字符串将在该字符数处截断。这不影响捕获所有结果。它只是限制了分析报告中字符串的大小。如果设置为零，字符串将包含所有结果。

- profile.serializer – string，默认值：<null>。

  如果不为空，则在以该参数指定的格式序列化的响应消息中返回收集的结果。将报告生成该响应消息所需的索引操作数以及要发送到客户端的字节大小。

  允许的值是<null>或任何有效的 MIME 类型或 TinkerPop 驱动程序 “序列化器” 枚举值。

  ```
  "application/json" or "GRAPHSON"
  "application/vnd.gremlin-v1.0+json" or "GRAPHSON_V1"
  "application/vnd.gremlin-v1.0+json;types=false" or "GRAPHSON_V1_UNTYPED"
  "application/vnd.gremlin-v2.0+json" or "GRAPHSON_V2"
  "application/vnd.gremlin-v2.0+json;types=false" or "GRAPHSON_V2_UNTYPED"
  "application/vnd.gremlin-v3.0+json" or "GRAPHSON_V3"
  "application/vnd.gremlin-v3.0+json;types=false" or "GRAPHSON_V3_UNTYPED"
  "application/vnd.graphbinary-v1.0" or "GRAPHBINARY_V1"
  ```

- profile.indexOps – boolean，允许的值：TRUE 和 FALSE，默认值：FALSE。

  如果为 true，则显示查询执行和序列化期间发生的所有索引操作的详细报告。警告：该报告可能会很长。

Neptune Gremlin **profile** 的示例输出

下面是一个示例 profile 查询。

```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile \
     -d '{"gremlin":"g.V().hasLabel(\"airport\")
                         .has(\"code\", \"AUS\")
                         .emit()
```

```
                        .repeat(in().simplePath())
                        .times(2)
                        .limit(100)",
          "profile.serializer":"application/vnd.gremlin-v3.0+gryo"}'
```

在博客文章[我来为您创建该图形 - 第 1 部分 – 航线](#)中的航线示例图形上执行此查询时，此查询会生成以下 `profile` 报告。

```
*******************************************************
                Neptune Gremlin Profile
*******************************************************

Query String
==================
g.V().hasLabel("airport").has("code",
 "AUS").emit().repeat(in().simplePath()).times(2).limit(100)

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([~label.eq(airport), code.eq(AUS)]),
 RepeatStep(emit(true),[VertexStep(IN,vertex), PathFilterStep(simple),
 RepeatEndStep],until(loops(2))), RangeGlobalStep(0,100)]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "AUS", ?) . project ?1 .],
 {estimatedCardinality=1, indexTime=84, hashJoin=true, joinTime=3, actualTotalOutput=1}
            PatternNode[(?1, <~label>, ?2=<airport>, <~>) . project ask .],
 {estimatedCardinality=3374, indexTime=29, hashJoin=true, joinTime=0,
 actualTotalOutput=61}
            RepeatNode {
                Repeat {
                    PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
 SimplePathFilter(?1, ?3)) .], {hashJoin=true, estimatedCardinality=50148, indexTime=0,
 joinTime=3}
                }
                Emit {
                    Filter(true)
                }
```

```
                LoopsCondition {
                    LoopsFilter([?1, ?3],eq(2))
                }
            }, annotations={repeatMode=BFS, emitFirst=true, untilFirst=false, leftVar=?
1, rightVar=?3}
        }, finishers=[limit(100)], annotations={path=[Vertex(?1):GraphStep,
 Repeat[Vertex(?3):VertexStep]], joinStats=true, optimizationTime=495, maxVarId=7,
 executionTime=323}
    },
    NeptuneTraverserConverterStep
]

Physical Pipeline
=================
NeptuneGraphQueryStep
    |-- StartOp
    |-- JoinGroupOp
        |-- SpoolerOp(100)
        |-- DynamicJoinOp(PatternNode[(?1, <code>, "AUS", ?) . project ?1 .],
 {estimatedCardinality=1, indexTime=84, hashJoin=true})
        |-- SpoolerOp(100)
        |-- DynamicJoinOp(PatternNode[(?1, <~label>, ?2=<airport>, <~>) . project
 ask .], {estimatedCardinality=3374, indexTime=29, hashJoin=true})
        |-- RepeatOp
            |-- <upstream input> (Iteration 0) [visited=1, output=1 (until=0, emit=1),
 next=1]
            |-- BindingSetQueue (Iteration 1) [visited=61, output=61 (until=0,
 emit=61), next=61]
                |-- SpoolerOp(100)
                |-- DynamicJoinOp(PatternNode[(?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) .], {hashJoin=true,
 estimatedCardinality=50148, indexTime=0})
            |-- BindingSetQueue (Iteration 2) [visited=38, output=38 (until=38,
 emit=0), next=0]
                |-- SpoolerOp(100)
                |-- DynamicJoinOp(PatternNode[(?3, ?5, ?1, ?6) . project ?
1,?3 . IsEdgeIdFilter(?6) . SimplePathFilter(?1, ?3)) .], {hashJoin=true,
 estimatedCardinality=50148, indexTime=0})
        |-- LimitOp(100)

Runtime (ms)
============
Query Execution:  392.686
Serialization:   2636.380
```

```
Traversal Metrics
=================
Step                                                               Count  Traversers
    Time (ms)    % Dur
-------------------------------------------------------------------------------------------------------
NeptuneGraphQueryStep(Vertex)                                        100         100
      314.162    82.78
NeptuneTraverserConverterStep                                        100         100
       65.333    17.22
                                            >TOTAL                     -           -
      379.495        -

Repeat Metrics
==============
Iteration  Visited   Output    Until     Emit     Next
----------------------------------------------------
        0        1        1        0        1        1
        1       61       61        0       61       61
        2       38       38       38        0        0
----------------------------------------------------
              100      100       38       62       62

Predicates
==========
# of predicates: 16

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
 performance

Results
=======
Count: 100
Output: [v[3], v[3600], v[3614], v[4], v[5], v[6], v[7], v[8], v[9], v[10], v[11],
 v[12], v[47], v[49], v[136], v[13], v[15], v[16], v[17], v[18], v[389], v[20], v[21],
 v[22], v[23], v[24], v[25], v[26], v[27], v[28], v[416], v[29], v[30], v[430], v[31],
 v[9...
Response serializer: GRYO_V3D0
Response size (bytes): 23566

Index Operations
================
Query execution:
    # of statement index ops: 3
```

```
    # of unique statement index ops: 3
    Duplication ratio: 1.0
    # of terms materialized: 0
Serialization:
    # of statement index ops: 200
    # of unique statement index ops: 140
    Duplication ratio: 1.43
    # of terms materialized: 393
```

除了通过调用 Neptune `explain` 返回的查询计划之外，`profile` 结果还包括有关查询执行的运行时系统统计数据。每个 Join 操作都标记了执行其联接所花费的时间以及进行该操作的解决方案的实际数量。

`profile` 输出包括在核心查询执行阶段以及序列化阶段（如果指定了 `profile.serializer` 选项）所花费的时间。

`profile` 输出的底部还包括在每个阶段执行的索引操作的明细。

请注意，出于缓存原因，同一查询的连续运行在运行时和索引操作方面可能会显示不同的结果。

对于使用 `repeat()` 步骤的查询，如果将 `repeat()` 步骤作为 `NeptuneGraphQueryStep` 的一部分下推，还将提供每次迭代边界的明细。

启用 DFE 时 **`profile`** 报告的差异

启用 Neptune DFE 替代查询引擎后，`profile` 输出会有所不同：

优化的遍历：本部分与 `explain` 输出中的一个部分类似，但包含其它信息。这包括在规划中考虑的 DFE 运算符类型，以及相关的最坏情况和最佳情况的成本估算。

物理管道：本部分捕获了用于执行查询的运算符。`DFESubQuery` 元素抽象了 DFE 用来执行计划中其负责的部分的物理计划。这些 `DFESubQuery` 元素将在下一部分中展开，其中列出了 DFE 统计数据。

DFE QueryEngine 统计信息：只有当 DFE 至少执行了部分查询时，才会显示此部分。这一部分概述了特定于 DFE 的各种运行时系统统计数据，并包含查询执行按 `DFESubQuery` 划分的各个部分所花费时间的详细细分。

在此部分中，不同 `DFESubQuery` 元素中的嵌套子查询是展平的，唯一标识符用以 `subQuery=` 开头的标头标记。

遍历指标：此部分显示阶梯级遍历指标，当 DFE 引擎运行查询的全部或部分内容时，显示 `DFEStep` 和/或 `NeptuneInterleavingStep` 的指标。请参阅 [使用 explain 和 profile 调整 Gremlin 查询](#)。

> **Note**
>
> DFE 是在实验室模式下发布的一项实验特征，因此 `profile` 输出的确切格式仍可能发生变化。

启用 Neptune 数据流引擎 (DFE) 时的 **`profile`** 输出示例

使用 DFE 引擎运行 Gremlin 查询时，Gremlin `profile` API 的输出格式如下例所示。

查询：

```
curl https://localhost:8182/gremlin/profile \
  -d "{\"gremlin\": \"g.withSideEffect('Neptune#useDFE', true).V().has('code',
 'ATL').out()\"}"
```

```
*******************************************************
                    Neptune Gremlin Profile
    *****************************************************

    Query String
    ==================
    g.withSideEffect('Neptune#useDFE', true).V().has('code', 'ATL').out()

    Original Traversal
    ==================
    [GraphStep(vertex,[]), HasStep([code.eq(ATL)]), VertexStep(OUT,vertex)]

    Optimized Traversal
    ===================
    Neptune steps:
    [
        DFEStep(Vertex) {
          DFENode {
            DFEJoinGroupNode[null](
              children=[
                DFEPatternNode((?1, vp://code[419430926], ?4, defaultGraph[526]) .
 project DISTINCT[?1] objectFilters=(in(ATL[452987149]) . ), {rangeCountEstimate=1},
                    opInfo=(type=PipelineJoin,
 cost=(exp=(in=1.00,out=1.00,io=0.00,comp=0.00,mem=0.00),wc=(in=1.00,out=1.00,io=0.00,comp=0.00
                    disc=(type=PipelineScan,
 cost=(exp=(in=1.00,out=1.00,io=0.00,comp=0.00,mem=34.00),wc=(in=1.00,out=1.00,io=0.00,comp=0.0
```

```
                DFEPatternNode((?1, ?5, ?6, ?7) . project ALL[?1, ?6] graphFilters=(!=
 defaultGraph[526] . ), {rangeCountEstimate=92233720368547758O7})],
             opInfo=[
               OperatorInfoWithAlternative[
                 rec=(type=PipelineJoin,
 cost=(exp=(in=1.00,out=27.76,io=0.00,comp=0.00,mem=0.00),wc=(in=1.00,out=27.76,io=0.00,comp=0.
                   disc=(type=PipelineScan,
 cost=(exp=(in=1.00,out=27.76,io=Infinity,comp=0.00,mem=295147905179352830000.00),wc=(in=1.00,c
                 alt=(type=PipelineScan,
 cost=(exp=(in=1.00,out=27.76,io=Infinity,comp=0.00,mem=295147905179352830000.00),wc=(in=1.00,c
          } [Vertex(?1):GraphStep, Vertex(?6):VertexStep]
       } ,
       NeptuneTraverserConverterDFEStep,
       DFECleanupStep
   ]


   Physical Pipeline
   =================
   DFEStep
       |-- DFESubQuery1

   DFEQueryEngine Statistics
   =================
   DFESubQuery1

 ########################################################################################
   # ID # Out #1 # Out #2 # Name                  # Arguments
                                                                          # Mode #
 Units In # Units Out # Ratio  # Time (ms) #

 ########################################################################################
   # 0  # 1      # -      # DFESolutionInjection  # solutions=[]
                                                                        # -    # 0
     # 1         # 0.00   # 0.01      #
   #    #        #        #                      # outSchema=[]
                                                                        #      #
     #           #        #           #

 ########################################################################################
   # 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_1 # -    #
 1        # 1         # 1.00   # 0.02      #
```

```
 ##########################################################################################
    # 2  # 3      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_2 # -    #
 1        # 242       # 242.00 # 0.02      #

 ##########################################################################################
    # 3  # 4      # -      # DFEMergeChunks        # -
                                                                          # -    # 242
      # 242       # 1.00   # 0.01      #

 ##########################################################################################
    # 4  # -      # -      # DFEDrain              # -
                                                                          # -    # 242
      # 0         # 0.00   # 0.01      #

 ##########################################################################################


    subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/
graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_1

 ##########################################################################################
    # ID # Out #1 # Out #2 # Name                 # Arguments
                          # Mode # Units In # Units Out # Ratio # Time (ms) #

 ##########################################################################################
    # 0  # 1      # -      # DFEPipelineScan      # pattern=Node(?1) with property
 'code' as ?4 and label 'ALL' # -    # 0        # 1         # 0.00  # 0.22      #
    #    #        #        #                     # inlineFilters=[(?4 IN ["ATL"])]
                          #      #          #           #       #           #
    #    #        #        #                     # patternEstimate=1
                          #      #          #           #       #           #

 ##########################################################################################
    # 1  # 2      # -      # DFEMergeChunks       # -
                          # -    # 1        # 1         # 1.00  # 0.02      #

 ##########################################################################################
    # 2  # 4      # -      # DFERelationalJoin    # joinVars=[]
                          # -    # 2        # 1         # 0.50  # 0.09      #

 ##########################################################################################
```

```
   # 3  # 2      # -      # DFESolutionInjection # solutions=[]
                        # -    # 0        # 1         # 0.00  # 0.01      #
   #    #        #        #                      # outSchema=[]
                        #      #          #           #       #           #

 ##########################################################################################
   # 4  # -      # -      # DFEDrain             # -
                        # -    # 1        # 0         # 0.00  # 0.01      #

 ##########################################################################################


   subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/
graph#089f43e3-4d71-4259-8d19-254ff63cee04/graph_2

 ##########################################################################################
   # ID # Out #1 # Out #2 # Name                # Arguments
 # Mode # Units In # Units Out # Ratio  # Time (ms) #

 ##########################################################################################
   # 0  # 1      # -      # DFESolutionInjection # solutions=[]
 # -    # 0        # 1         # 0.00   # 0.01      #
   #    #        #        #                      # outSchema=[?1]
 #      #          #           #        #           #

 ##########################################################################################
   # 1  # 2      # 3      # DFETee               # -
 # -    # 1        # 2         # 2.00   # 0.01      #

 ##########################################################################################
   # 2  # 4      # -      # DFEDistinctColumn    # column=?1
 # -    # 1        # 1         # 1.00   # 0.21      #
   #    #        #        #                      # ordered=false
 #      #          #           #        #           #

 ##########################################################################################
   # 3  # 5      # -      # DFEHashIndexBuild    # vars=[?1]
 # -    # 1        # 1         # 1.00   # 0.03      #

 ##########################################################################################
   # 4  # 5      # -      # DFEPipelineJoin      # pattern=Edge((?1)-[?7:?5]->(?6))
 # -    # 1        # 242       # 242.00 # 0.51      #
   #    #        #        #                      # constraints=[]
 #      #          #           #        #           #
```

```
   #    #        #        #                    # patternEstimate=9223372036854775807
#      #          #          #        #          #

#########################################################################################
   # 5  # 6      # 7      # DFESync              # -
# -    # 243      # 243      # 1.00   # 0.02      #

#########################################################################################
   # 6  # 8      # -      # DFEForwardValue      # -
# -    # 1        # 1        # 1.00   # 0.01      #

#########################################################################################
   # 7  # 8      # -      # DFEForwardValue      # -
# -    # 242      # 242      # 1.00   # 0.02      #

#########################################################################################
   # 8  # 9      # -      # DFEHashIndexJoin     # -
# -    # 243      # 242      # 1.00   # 0.31      #

#########################################################################################
   # 9  # -      # -      # DFEDrain             # -
# -    # 242      # 0        # 0.00   # 0.01      #

#########################################################################################


   Runtime (ms)
   ============
   Query Execution: 11.744

   Traversal Metrics
   =================
   Step                                                               Count
Traversers       Time (ms)    % Dur

-----------------------------------------------------------------------------------------
   DFEStep(Vertex)                                                      242
242          10.849    95.48
   NeptuneTraverserConverterDFEStep                                     242
242           0.514     4.52
                                                >TOTAL                     -
-          11.363        -

   Predicates
```

```
==========
# of predicates: 18

Results
=======
Count: 242


Index Operations
================
Query execution:
    # of statement index ops: 0
    # of terms materialized: 0
```

> **Note**
>
> 由于 DFE 引擎是在实验室模式下发布的实验特征，因此 `profile` 输出的确切格式可能会发生变化。

使用 `explain` 和 `profile` 调整 Gremlin 查询

您通常可以在 Amazon Neptune 中使用从 Neptune Explain 和 Profile API 获得的报告中提供的信息，调整您的 Gremlin 查询以获得更好的性能。为此，了解 Neptune 如何处理 Gremlin 遍历会有所帮助。

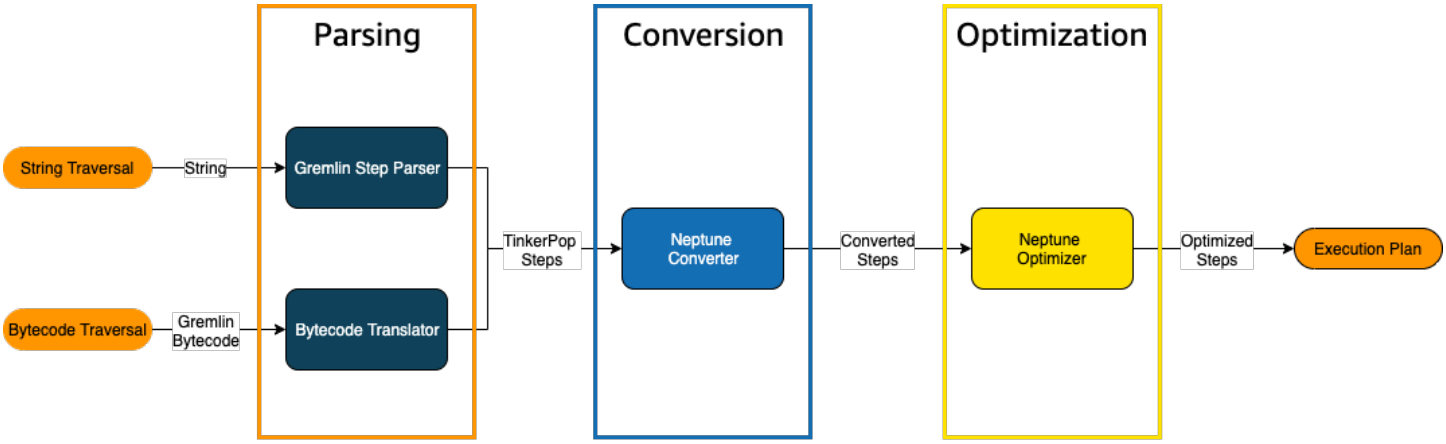
> **Important**
>
> 3.4.11 TinkerPop 版本中进行了更改，提高了查询处理方式的正确性，但目前有时会严重影响查询性能。
> 例如，这种查询的运行速度可能会慢得多：
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   out()
> ```
>
> 由于 3.4.11 的更改，极限步骤之后的顶点现在以非最佳方式获取。 TinkerPop 为避免这种情况，您可以通过在 `order().by()` 之后的任何点添加 barrier() 步骤来修改查询。例如：
>
> ```
> g.V().hasLabel('airport').
> ```

```
  order().
    by(out().count(),desc).
  limit(10).
  barrier().
  out()
```

TinkerPop 3.4.11 已在 Neptune 引擎版本 1.0.5.0 中启用。

了解 Neptune 中的 Gremlin 遍历处理

当 Gremlin 遍历发送到 Neptune 时，有三个主要过程可将遍历转换为底层执行计划供引擎执行。它们是解析、转换和优化：



遍历解析过程

处理遍历的第一步是将其解析为通用语言。在 Neptune 中，通用语言是 API 中的一组 TinkerPop 步骤。TinkerPop这些步骤中的每一个都代表遍历中的一个计算单位。

您可以将 Gremlin 遍历以字符串或字节码的形式发送给 Neptune。REST 端点和 Java 客户端驱动程序 `submit()` 方法以字符串形式发送遍历，如下例所示：

```
client.submit("g.V()")
```

使用 Gremlin 语言变体 (GLV) 的应用程序和语言驱动程序以字节码发送遍历。

遍历转换过程

处理遍历的第二步是将其步长转换为一组已转换和未转换的 Neptune TinkerPop 步长。Apache TinkerPop Gremlin 查询语言中的大多数步骤都转换为特定于海王星的步骤，这些步骤经过优化，可在

底层 Neptune 引擎上运行。当在遍历中遇到没有 Neptune 等效项的 TinkerPop 步骤时，查询引擎会处理该步骤和遍历中的所有后续步骤。 TinkerPop

有关在什么情况下可以转换哪些步骤的更多信息，请参阅Gremlin 步骤支持。

遍历优化过程

遍历处理的最后一步是通过优化器运行一系列已转换和未转换的步骤，以尝试确定最佳的执行计划。此优化的输出是 Neptune 引擎处理的执行计划。

使用 Neptune Gremlin **explain** API 调整查询

Neptune Explain API 与 Gremlin `explain()` 步骤不同。它返回 Neptune 引擎在执行查询时将处理的最终执行计划。由于它不执行任何处理，因此无论使用什么参数，它都会返回相同的计划，并且其输出不包含有关实际执行的统计数据。

考虑以下简单的遍历，它可以找到安克雷奇的所有机场顶点：

```
g.V().has('code','ANC')
```

有两种方法可以通过 Neptune `explain` API 运行此遍历。第一种方法是对 Explain 端点进行 REST 调用，如下所示：

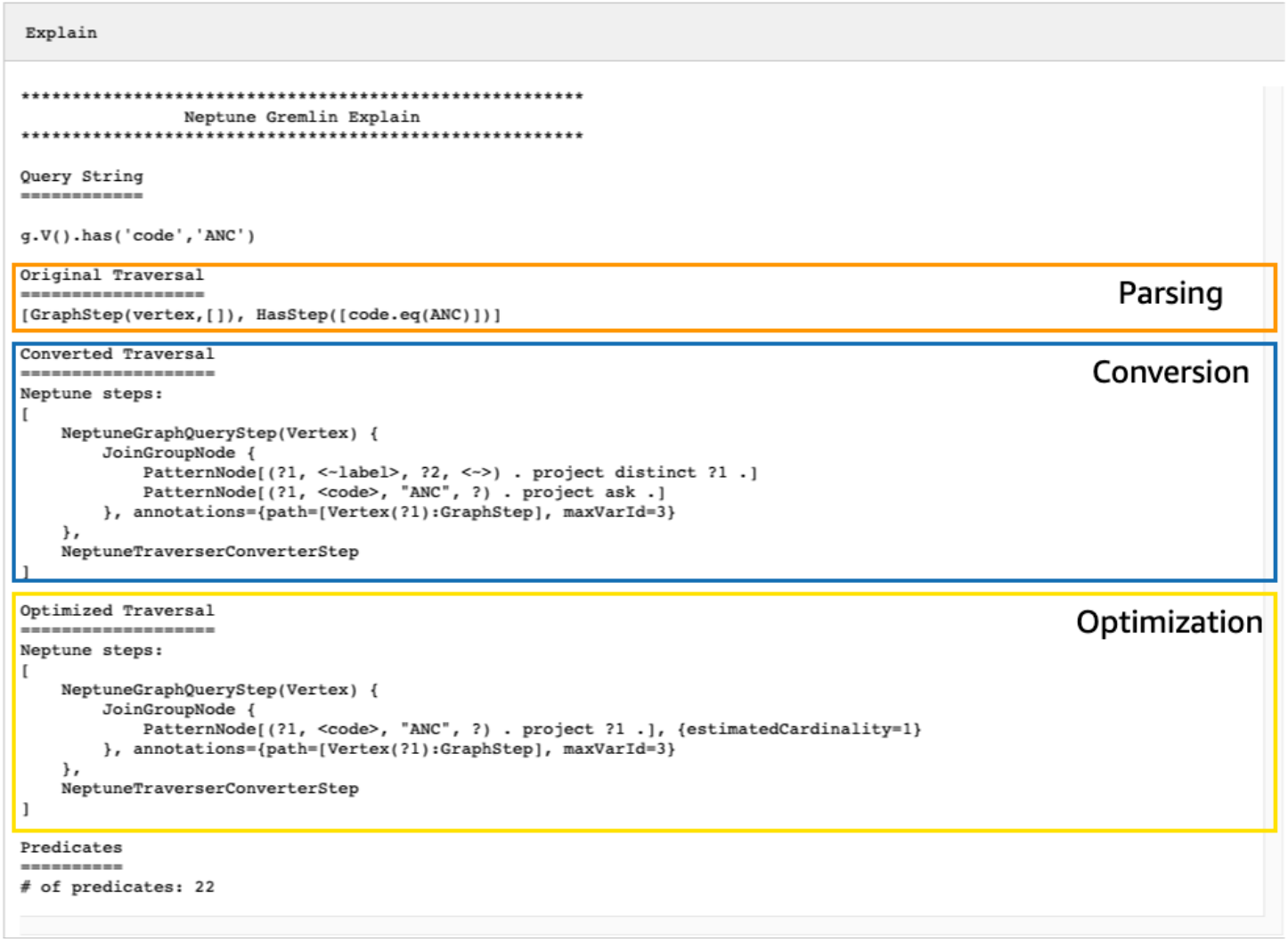
```
curl -X POST https://your-neptune-endpoint:port/gremlin/explain -d
 '{"gremlin":"g.V().has('code','ANC')"}'
```

第二种方法是将 Neptune Workbench 的 %%gremlin 单元格魔术命令与 `explain` 参数结合使用。这会将单元格正文中包含的遍历传递给 Neptune `explain` API，然后在运行单元格时显示结果输出：

```
%%gremlin explain

g.V().has('code','ANC')
```

生成的 `explain` API 输出描述了 Neptune 的遍历执行计划。如下图所示，该计划包括处理管道中 3 个步骤的每一个步骤：



通过查看未转换的步骤来调整遍历

在 Neptune `explain` API 输出中首先要查找的内容之一是未转换为 Neptune 原生步骤的 Gremlin 步骤。在查询计划中，当遇到无法转换为 Neptune 原生步骤的步骤时，该步骤和计划中的所有后续步骤都将由 Gremlin 服务器处理。

在上述示例中，遍历中的所有步骤均已转换。让我们来看看这个遍历的 `explain` API 输出：

```
g.V().has('code','ANC').out().choose(hasLabel('airport'), values('code'), constant('Not
 an airport'))
```

如下图所示，Neptune 无法转换 `choose()` 步骤：

Explain

```
****************************************************
                Neptune Gremlin Explain
****************************************************

Query String
============

g.V().has('code','ANC').out().choose(hasLabel('airport'), values('code'), constant('Not an airport'))

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(OUT,vertex), ChooseStep([HasStep([~label.eq(airport)]), HasNex

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [ChooseStep([HasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [PropertiesSte

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1}
            PatternNode[(?1, ?5, ?3, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .], {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep], maxVarId=7}
    },
    NeptuneTraverserConverterStep
]
+ not converted into Neptune steps: [ChooseStep([NeptuneHasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [Proper

WARNING: >> ChooseStep([NeptuneHasStep([~label.eq(airport)]), HasNextStep],[[(eq(true)), [PropertiesStep([code],value), E

Predicates
==========
# of predicates: 26
```

您可以执行几项操作以优化遍历的性能。第一种方法是重写它，以消除无法转换的步骤。另一种方法是将该步骤移到遍历的末尾，这样所有其它步骤都可以转换为原生步骤。

包含未转换的步骤的查询计划并不总是需要调整。如果无法转换的步骤位于遍历的末尾，并且与输出的格式化方式有关，而不是与图型的遍历方式有关，那么它们可能对性能影响不大。

在检查 Neptune `explain` API 的输出时，要注意的另一件事是不使用索引的步骤。以下遍历查找航班降落在安克雷奇的所有机场：

```
g.V().has('code','ANC').in().values('code')
```

此遍历的 Explain API 的输出是：

```
****************************************************
                Neptune Gremlin Explain
****************************************************

Query String
============

g.V().has('code','ANC').in().values('code')

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,vertex),
 PropertiesStep([code],value)]

Converted Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
            PatternNode[(?3, ?7, ?8, <~>) . project ?3,?8 . ContainsFilter(?7 in
 (<code>)) .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .],
 {estimatedCardinality=1}
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .],
 {estimatedCardinality=INFINITY}
```

```
            PatternNode[(?3, ?7=<code>, ?8, <~>) . project ?3,?8 .],
 {estimatedCardinality=7564}
       }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 26

WARNING: reverse traversal with no edge label(s) - .in() / .both() may impact query
 performance
```

输出底部的 WARNING 消息之所以出现，是因为无法使用 Neptune 维护的 3 个索引之一来处理遍历中的 `in()` 步骤（请参阅如何在 Neptune 中为语句编制索引和Neptune 中的 Gremlin 语句）。由于 `in()` 步骤不包含边缘筛选条件，因此它无法使用 SPOG、POGS 或 GPSO 索引对其进行解析。相反，Neptune 必须执行联合扫描才能找到请求的顶点，但效率要低得多。

在这种情况下，可通过两种方法来调整遍历。第一种方法是在 `in()` 步骤中添加一个或多个筛选条件，以便可以使用索引查找来解析查询。对于上面的示例，这可能是：

```
g.V().has('code','ANC').in('route').values('code')
```

修改后的遍历的 Neptune `explain` API 的输出不再包含 WARNING 消息：

```
*******************************************************
                Neptune Gremlin Explain
*******************************************************

Query String
============

g.V().has('code','ANC').in('route').values('code')

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)]), VertexStep(IN,[route],vertex),
 PropertiesStep([code],value)]

Converted Traversal
```

```
==================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .]
            PatternNode[(?1, <code>, "ANC", ?) . project ask .]
            PatternNode[(?3, ?5, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?6) .
 ContainsFilter(?5 in (<route>)) .]
            PatternNode[(?3, <~label>, ?4, <~>) . project ask .]
            PatternNode[(?3, ?7, ?8, <~>) . project ?3,?8 . ContainsFilter(?7 in
 (<code>)) .]
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Optimized Traversal
==================
Neptune steps:
[
    NeptuneGraphQueryStep(PropertyValue) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .],
 {estimatedCardinality=1}
            PatternNode[(?3, ?5=<route>, ?1, ?6) . project ?1,?3 . IsEdgeIdFilter(?
6) .], {estimatedCardinality=32042}
            PatternNode[(?3, ?7=<code>, ?8, <~>) . project ?3,?8 .],
 {estimatedCardinality=7564}
        }, annotations={path=[Vertex(?1):GraphStep, Vertex(?3):VertexStep,
 PropertyValue(?8):PropertiesStep], maxVarId=9}
    },
    NeptuneTraverserConverterStep
]

Predicates
==========
# of predicates: 26
```

如果您正在运行许多此类遍历，另一种选择是在启用了可选 OSGP 索引的 Neptune 数据库集群中运行它们（请参阅启用 OSGP 索引）。启用 OSGP 索引具有缺点：

- 在加载任何数据之前，必须在数据库集群中启用它。
- 顶点和边缘的插入速率可减慢多达 23%。
- 存储使用量将增加大约 20%。
- 将请求分散到所有索引上的读取查询可能会增加了延迟。

对于一组受限的查询模式来说，拥有 OSGP 索引很有意义，但除非您经常运行这些模式，否则通常最好尽量确保可以使用三个主索引解析您编写的遍历。

使用大量谓词

Neptune 将图形中的每个边缘标签和每个不同的顶点或边缘属性名称视为谓词，并且默认设计为使用相对较少的不同谓词。当您的图形数据中有超过几千个谓词时，性能可能会降低。

如果是这样的话，Neptune `explain` 输出会警告您：

```
Predicates
==========
# of predicates: 9549
WARNING: high predicate count (# of distinct property names and edge labels)
```

如果不方便重新设计数据模型以减少标签和属性的数量，从而减少谓词的数量，那么调整遍历的最佳方法是在启用了 OSGP 索引的数据库集群中运行遍历，如上所述。

使用 Neptune Gremlin **`profile`** API 调整遍历

Neptune `profile` API 与 Gremlin `profile()` 步骤有很大不同。与 `explain` API 一样，它的输出包括 Neptune 引擎在执行遍历时使用的查询计划。此外，根据设置遍历的参数的方式，`profile` 输出还包括遍历的实际执行统计数据。

再次以找到安克雷奇所有机场顶点的简单遍历为例：

```
g.V().has('code','ANC')
```

与 `explain` API 一样，您可以使用 REST 调用来调用 `profile` API：

```
curl -X POST https://your-neptune-endpoint:port/gremlin/profile -d
 '{"gremlin":"g.V().has('code','ANC')"}'
```

还可以将 Neptune Workbench 的 %%gremlin 单元格魔术命令与 `profile` 参数结合使用。这会将单元格正文中包含的遍历传递给 Neptune `profile` API，然后在运行单元格时显示结果输出：

```
%%gremlin profile

g.V().has('code','ANC')
```

生成的 `profile` API 输出包含 Neptune 的遍历执行计划和有关计划执行的统计数据，如下图所示：

```
Profile

************************************************************
                Neptune Gremlin Profile
************************************************************

Query String                                                                                                    Execution Plan
==================

g.V().has('code','ANC')

Original Traversal
==================
[GraphStep(vertex,[]), HasStep([code.eq(ANC)])]

Optimized Traversal
===================
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1, indexTime=0, joinTime=0, numSear
        }, annotations={path=[Vertex(?1):GraphStep], joinStats=true, optimizationTime=1, maxVarId=3, executionTime=3}
    },
    NeptuneTraverserConverterStep
]

Physical Pipeline                                                                                               Pipeline
=================
NeptuneGraphQueryStep
    |-- StartOp
    |-- JoinGroupOp
        |-- SpoolerOp(1000)
        |-- DynamicJoinOp(PatternNode[(?1, <code>, "ANC", ?) . project ?1 .], {estimatedCardinality=1})

Runtime (ms)
============
Query Execution: 5.096

Traversal Metrics                                                                                               Statistics and Results
=================
Step                                                               Count  Traversers       Time (ms)    % Dur
-------------------------------------------------------------------------------------------------------------
NeptuneGraphQueryStep(Vertex)                                          1           1           0.956    90.62
NeptuneTraverserConverterStep                                          1           1           0.099     9.38
                                            >TOTAL                     -           -           1.055        -

Predicates
==========
# of predicates: 26

Results
=======
Count: 1
Output: [v[2]]


Index Operations
================
Query execution:
    # of statement index ops: 1
    # of unique statement index ops: 1
    Duplication ratio: 1.0
    # of terms materialized: 0
```

在 `profile` 输出中，执行计划部分仅包含遍历的最终执行计划，不包含中间步骤。管道部分包含已执行的物理管道操作以及遍历执行所花费的实际时间（以毫秒为单位）。运行时系统指标对于比较两个不同版本的遍历在优化时所花费的时间非常有用。

> **Note**
>
> 遍历的初始运行时间通常比后续运行时更长，因为第一个遍历会导致相关数据被缓存。

`profile` 输出的第三部分包含执行统计数据和遍历的结果。要了解这些信息在调整遍历时有何用处，可以考虑以下遍历，它可以找到名称以“Anchora”开头的每个机场，以及从这些机场转乘两次即可到达的所有机场，同时返回机场代码、航班线路和距离：

```
%%gremlin profile

g.withSideEffect("Neptune#fts.endpoint", "{your-OpenSearch-endpoint-URL").
    V().has("city", "Neptune#fts Anchora~").
    repeat(outE('route').inV().simplePath()).times(2).
    project('Destination', 'Route').
        by('code').
        by(path().by('code').by('dist'))
```

Neptune **`profile`** API 输出中的遍历指标

所有 `profile` 输出中可用的第一组指标是遍历指标。这些指标与 Gremlin `profile()` 步骤指标类似，但有一些区别：

```
Traversal Metrics
=================
Step                                                               Count  Traversers
    Time (ms)    % Dur
-------------------------------------------------------------------------------------------------------------
NeptuneGraphQueryStep(Vertex)                                       3856        3856
       91.701     9.09
NeptuneTraverserConverterStep                                       3856        3856
       38.787     3.84
ProjectStep([Destination, Route],[value(code), ...                  3856        3856
      878.786    87.07
  PathStep([value(code), value(dist)])                              3856        3856
      601.359
```

```
                                        >TOTAL                     -          -
    1009.274        -
```

遍历指标表的第一列列出了遍历执行的步骤。前两个步骤通常是 Neptune 特定的步骤 `NeptuneGraphQueryStep` 和 `NeptuneTraverserConverterStep`。

`NeptuneGraphQueryStep` 表示可以由 Neptune 引擎在原生环境中转换和执行的整个遍历部分的执行时间。

`NeptuneTraverserConverterStep`表示将这些已转换步骤的输出转换为 TinkerPop 遍历器的过程，这些遍历器允许处理无法转换的步骤（如果有），或者以兼容的格式返回结果。 TinkerPop

在上面的示例中，我们有几个未转换的步骤，因此我们看到每个 TinkerPop 步骤 (`ProjectStep`,`PathStep`) 随后都作为一行出现在表中。

表中的第二列报告通过该步骤的表示遍历器的数量，而第三列报告通过该步骤的遍历器数量，如配置步骤文档中所TinkerPop述。`CountTraversers`

在我们的示例中，`NeptuneGraphQueryStep` 返回了 3856 个顶点和 3856 个遍历器，在剩下的处理过程中，这些数字保持不变，因为 `ProjectStep` 和 `PathStep` 正在格式化结果，而不是筛选结果。

> **Note**
>
> 与之不同的是 TinkerPop，Neptune 引擎不会通过增加和步数来优化性能。`NeptuneGraphQueryStep` `NeptuneTraverserConverterStepBulking` 是一种将遍历器组合在同一个顶点上以减少操作开销的 TinkerPop操作，这就是导致和数字出现差异的原因。`Count Traversers`由于批量仅发生在 Neptune 委托 TinkerPop给的步骤中，而不发生在 Neptune 本机处理的步骤中，因此和列很少有区别。`Count Traverser`

“时间”列报告该步骤所花费的毫秒数，而 `%` `Dur` 列报告该步骤占总处理时间的百分比。这些指标通过显示花费时间最多的步骤来告诉您调整工作要集中在哪里。

Neptune **`profile`** API 输出中的索引操作指标

Neptune Profile API 输出中的另一组指标是索引操作：

```
Index Operations
================
Query execution:
```

```
    # of statement index ops: 23191
    # of unique statement index ops: 5960
    Duplication ratio: 3.89
    # of terms materialized: 0
```

它们报告：

- 索引查找的总数。
- 执行的唯一索引查找的次数。
- 索引查找总数与唯一索引查找次数的比率。比率越低，表示冗余越少。
- 从术语词典中具体化的术语数量。

Neptune **profile** API 输出中的重复指标

如果您的遍历使用如上例所示的 `repeat()` 步骤，则 `profile` 输出中将显示包含重复指标的部分：

```
Repeat Metrics
==============
Iteration  Visited   Output    Until     Emit      Next
-------------------------------------------------------
        0        2        0        0        0        2
        1       53        0        0        0       53
        2     3856     3856     3856        0        0
-------------------------------------------------------
              3911     3856     3856        0       55
```

它们报告：

- 一行的循环次数（`Iteration` 列）。
- 循环访问的元素数量（`Visited` 列）。
- 循环输出的元素数量（`Output` 列）。
- 循环输出的最后一个元素（`Until` 列）。
- 循环发出的元素数量（`Emit` 列）。
- 从循环传递到后续循环的元素数量（`Next` 列）。

这些重复指标对于了解遍历的分支因子非常有帮助，以了解数据库完成了多少工作。您可以使用这些数字来诊断性能问题，尤其是当同一个遍历在不同的参数下性能大相径庭时。

Neptune **profile** API 输出中的全文搜索指标

当遍历使用全文搜索查找时（如上例所示），profile 输出中会出现一个包含全文搜索 (FTS) 指标的部分：

```
FTS Metrics
==============
SearchNode[(idVar=?1, query=Anchora~, field=city) . project ?1 .],
    {endpoint=your-OpenSearch-endpoint-URL, incomingSolutionsThreshold=1000,
 estimatedCardinality=INFINITY,
    remoteCallTimeSummary=[total=65, avg=32.500000, max=37, min=28],
    remoteCallTime=65, remoteCalls=2, joinTime=0, indexTime=0, remoteResults=2}

    2 result(s) produced from SearchNode above
```

这显示了发送到 ElasticSearch (ES) 集群的查询，并报告了与之交互的几个指标 ElasticSearch ，这些指标可以帮助您查明与全文搜索相关的性能问题：

- 有关 ElasticSearch 索引调用的摘要信息：
  - 所有 remoteCall 满足查询所需的总毫秒数 (total)。
  - 在一个 remoteCall 中花费的平均毫秒数 (avg)。
  - 在一个 remoteCall 中花费的最小毫秒数 (min)。
  - 在一个 remoteCall 中花费的最大毫秒数 (max)。
- 远程调用 ElasticSearch () remoteCallTime 所消耗的总时间。
- 向 ElasticSearch () remoteCalls 发出的远程呼叫次数。
- ElasticSearch 结果联接所花费的毫秒数 ()。joinTime
- 在索引查找中花费的毫秒数 (indexTime)。
- ElasticSearch (remoteResults) 返回的结果总数。

Amazon Neptune 中的原生 Gremlin 步骤支持

如调整 Gremlin 查询中所述，Amazon Neptune 引擎目前并不完全原生支持所有 Gremlin 步骤。当前支持分为四类：

- 始终可以转换为原生 Neptune 引擎操作的 Gremlin 步骤
- 在某些情况下可以转换为原生 Neptune 引擎操作的 Gremlin 步骤
- 从不转换为原生 Neptune 引擎操作的 Gremlin 步骤

- Neptune 中根本不支持的 Gremlin 步骤

始终可以转换为原生 Neptune 引擎操作的 Gremlin 步骤

只要满足以下条件，许多 Gremlin 步骤可以转换为原生 Neptune 引擎操作：

- 在查询中，它们前面没有无法转换的步骤。
- 它们的父步骤（如果有）可以转换。
- 它们的所有子遍历（如果有）都可以转换。

如果满足这些条件，以下 Gremlin 步骤将始终转换为原生 Neptune 引擎操作：

- and( )
- as( )
- count( )
- E( )
- emit( )
- explain( )
- group( )
- groupCount( )
- has( )
- identity( )
- is( )
- key( )
- label( )
- limit( )
- local( )
- loops( )
- not( )
- or( )
- profile( )
- properties( )
- subgraph( )

- until( )
- V( )
- value( )
- valueMap( )
- values( )

在某些情况下可以转换为原生 Neptune 引擎操作的 Gremlin 步骤

在某些情况下，一些 Gremlin 步骤可以转换为原生 Neptune 引擎操作，但在其它情况下则不可以：

- addE( ) – `addE()` 步骤通常可以转换为原生 Neptune 引擎操作，除非紧随其后的是包含遍历作为键的 `property()` 步骤。
- addV( ) – `addV()` 步骤通常可以转换为原生 Neptune 引擎操作，除非紧随其后的是包含遍历作为键的 `property()` 步骤，或者除非分配了多个标签。
- aggregate( ) – `aggregate()` 步骤通常可以转换为原生 Neptune 引擎操作，除非该步骤用于子遍历，或者除非存储的值不是顶点、边缘、id、标签或属性值。

  在下面的示例中，未转换 `aggregate()`，因为它正在子遍历中使用：

  ```
  g.V().has('code','ANC').as('a')
       .project('flights').by(select('a')
       .outE().aggregate('x'))
  ```

  在此示例中，未转换 aggregate()，因为存储的内容是值的 `min()`：

  ```
  g.V().has('code','ANC').outE().aggregate('x').by(values('dist').min())
  ```

- barrier( ) – `barrier()` 步骤通常可以转换为原生 Neptune 引擎操作，除非其后的步骤未被转换。
- cap( ) – 转换 `cap()` 步骤的唯一情况是当将它与 `unfold()` 步骤组合以返回顶点、边缘、id 或属性值聚合的展开版本时。在本例中，`cap()` 将进行转换，因为它后跟 `.unfold()`：

  ```
  g.V().has('airport','country','IE').aggregate('airport').limit(2)
       .cap('airport').unfold()
  ```

  但是，如果您移除 `.unfold()`，则不会转换 `cap()`：

  ```
  g.V().has('airport','country','IE').aggregate('airport').limit(2)
  ```

```
    .cap('airport')
```

- coalesce () — 转换`coalesce()`步骤的唯一情况是它遵循食谱页面上推荐的 Upsert 模式。TinkerPop 不允许使用其它 coalesce() 模式。转换仅限于以下情况：所有子遍历都可以转换，它们都生成与输出相同的类型（顶点、边缘、id、值、键或标签），它们都遍历到一个新元素，并且它们不包含步骤 `repeat()`。
- constant( ) – 当前，只有在遍历的 `sack().by()` 部分中使用 constant() 步骤来分配常量值时，才会转换此步骤，如下所示：

```
g.V().has('code','ANC').sack(assign).by(constant(10)).out().limit(2)
```

- cyclicPath( ) – `cyclicPath()` 步骤通常可以转换为原生 Neptune 引擎操作，除非该步骤与 `by()`、`from()` 或 `to()` 调制器结合使用。例如，在以下查询中，未转换 `cyclicPath()`：

```
g.V().has('code','ANC').as('a').out().out().cyclicPath().by('code')
g.V().has('code','ANC').as('a').out().out().cyclicPath().from('a')
g.V().has('code','ANC').as('a').out().out().cyclicPath().to('a')
```

- drop( ) – `drop()` 步骤通常可以转换为原生 Neptune 引擎操作，除非在 `sideEffect()` 或 `optional()` 步骤中使用该步骤。
- fold () — 只有两种情况可以转换 fold () 步骤，即在TinkerPop 食谱页面推荐的 Upsert 模式中使用它时，以及`group().by()`在这样的上下文中使用它时：

```
g.V().has('code','ANC').out().group().by().by(values('code', 'city').fold())
```

- id( ) – 除非在属性上使用 `id()` 步骤，否则将转换该步骤，如下所示：

```
g.V().has('code','ANC').properties('code').id()
```

- order( ) – `order()` 步骤通常可以转换为原生 Neptune 引擎操作，除非满足以下条件之一：
  - `order()` 步骤位于嵌套的子遍历中，如下所示：

```
g.V().has('code','ANC').where(V().out().order().by(id))
```

  - 正在使用本地排序，例如使用 `order(local)`。
  - 正在 `by()` 调制中使用自定义比较器进行排序。一个例子是这样使用 `sack()`：

```
g.withSack(0).
  V().has('code','ANC').
```

```
repeat(outE().sack(sum).by('dist').inV()).times(2).limit(10).
order().by(sack())
```

  - 同一个元素上有多个排序。
- project( ) – `project()` 步骤通常可以转换为原生 Neptune 引擎操作，除非 `project()` 后面的 `by()` 语句数量与指定的标签数量不匹配，如下所示：

```
g.V().has('code','ANC').project('x', 'y').by(id)
```

- range( ) - 只有当相关范围的下限为零（例如 `range(0,3)`）时，才会转换 `range()` 步骤。
- repeat( ) – `repeat()` 步骤通常可以转换为原生 Neptune 引擎操作，除非它嵌套在另一个 `repeat()` 步骤中，如下所示：

```
g.V().has('code','ANC').repeat(out().repeat(out()).times(2)).times(2)
```

- sack( ) – `sack()` 步骤通常可以转换为原生 Neptune 引擎操作，但以下情况除外：
  - 如果使用的是非数字 sack 运算符。
  - 如果使用的是 `+`、`-`、`mult`、`div`、`min` 和 `max` 以外的数字 sack 运算符。
  - 如果在 `where()` 步骤中使用 `sack()` 来根据 sack 值进行筛选，如下所示：

```
g.V().has('code','ANC').sack(assign).by(values('code')).where(sack().is('ANC'))
```

- sum( ) – `sum()` 步骤通常可以转换为原生 Neptune 引擎操作，但在用于计算全局求和时则不可转换，如下所示：

```
g.V().has('code','ANC').outE('routes').values('dist').sum()
```

- union( ) – `union()` 步骤可以转换为原生 Neptune 引擎操作，只要它是查询中除最终步骤之外的最后一步。
- unfold () — 只有在食谱页面推荐的 Upsert 模式中使用该`unfold()`步骤时，以及将其与以下内容一起`cap()`使用时，才能将其转换为原生 Neptune 引擎操作：TinkerPop

```
g.V().has('airport','country','IE').aggregate('airport').limit(2)
     .cap('airport').unfold()
```

- where( ) – `where()` 步骤通常可以转换为原生 Neptune 引擎操作，但以下情况除外：
  - 当使用 by() 调制时，如下所示：

```
g.V().hasLabel('airport').as('a')
     .where(gt('a')).by('runways')
```

- 当使用 `eq`、`neq`、`within` 和 `without` 以外的比较运算符时。
- 当使用用户提供的聚合时。

从不转换为原生 Neptune 引擎操作的 Gremlin 步骤

Neptune 支持以下 Gremlin 步骤，但它们从不转换为原生 Neptune 引擎操作。相反，它们由 Gremlin 服务器执行。

- choose( )
- coin( )
- inject( )
- match( )
- math( )
- max( )
- mean( )
- min( )
- option( )
- optional( )
- path( )
- propertyMap( )
- sample( )
- skip( )
- tail( )
- timeLimit( )
- tree( )

Neptune 中根本不支持的 Gremlin 步骤

Neptune 根本不支持以下 Gremlin 步骤。在大多数情况下，这是因为它们需要 `GraphComputer`，而 Neptune 目前不支持。

- connectedComponent( )
- io( )
- shortestPath( )
- withComputer( )
- pageRank( )
- peerPressure( )
- program( )

实际上，`io()` 步骤得到了部分支持，因为它可以用来从 URL 进行 `read()`，但不用来 `write()`。

在 Neptune DFE 查询引擎中使用 Gremlin

如果您在实验室模式下完全启用已知为 DFE 的 Neptune 替代查询引擎（通过将 `neptune_lab_mode` 数据库集群参数设置为 `DFEQueryEngine=enabled`），那么 Neptune 会将只读的 Gremlin 查询/遍历转换为中间逻辑表示形式，并尽可能在 DFE 引擎上运行它们。

但是，DFE 尚不支持所有 Gremlin 步骤。当无法在 DFE 上原生运行某个步骤时，Neptune 会退回来运行该步骤。 TinkerPop `explain` 和 `profile` 报告包含发生这种情况时的警告。

> **Note**
>
> 从引擎版本 1.0.5.0 开始，在没有原生支持的情况下处理 Gremlin 步骤的默认 DFE 行为发生了变化。以前 DFE 发动机落在海王星 Gremlin 发动机上，现在它又回到了普通发动机上。TinkerPop

DFE 引擎原生支持的 Gremlin 步骤

- **`GraphStep`**
- **`VertexStep`**
- **`EdgeVertexStep`**
- **`IdStep`**
- **`TraversalFilterStep`**
- **`PropertiesStep`**

- **HasStep** 支持对属性、id 和标签上的顶点和边缘进行筛选，但文本和 Without 谓词除外。
- **WherePredicateStep**，使用 Path 范围的筛选条件，但不支持 ByModulation、SideEffect 或 Map 查找
- **DedupGlobalStep**，但 ByModulation、SideEffect 和 Map 查找支持除外。

查询计划交错

当转换过程遇到没有相应原生 DFE 运算符的 Gremlin 步骤时，在回退到使用 Tinkerpop 之前，它会尝试查找其它可以在 DFE 引擎上原生运行的中间查询部分。它通过将交错逻辑应用于顶级遍历来实现这一点。结果是尽可能使用支持的步骤。

任何此类中间、非前缀的查询转换在 explain 和 profile 输出中都使用 NeptuneInterleavingStep 来表示。

为比较性能，您可能需要关闭查询中的交错功能，同时仍使用 DFE 引擎来运行前缀部分。或者，您可能只想使用 TinkerPop 引擎执行非前缀查询。您可以通过使用 disableInterleaving 查询提示执行该操作。

正如值为 false 的 useDFE 查询提示完全阻止在 DFE 上运行查询一样，值为 true 的 disableInterleaving 查询提示会关闭查询转换的 DFE 交错。例如：

```
g.with('Neptune#disableInterleaving', true)
 .V().has('genre','drama').in('likes')
```

更新了 Gremlin **explain** 和 **profile** 输出

Gremlin explain 提供有关 Neptune 用来运行查询的优化遍历的详细信息。有关启用 DFE 引擎时 explain 输出内容的示例，请参阅 DFE explain 输出示例。

Gremlin profile API 运行指定的 Gremlin 遍历，收集有关运行的各种指标，并生成 Profile 报告，其中包含有关优化查询计划的详细信息以及各种运算符的运行时系统统计数据。有关启用 DFE 引擎时 profile 输出内容的示例，请参阅 DFE profile 输出示例。

> **Note**
>
> 由于 DFE 引擎是在实验室模式下发布的实验特征，因此 explain 和 profile 输出的确切格式可能会发生变化。

使用 openCypher 访问 Neptune 图形

Neptune 支持使用 openCypher 构建图形应用程序，openCypher 是目前使用图形数据库的开发人员最常用的查询语言之一。开发人员、业务分析师和数据科学家都喜欢 openCypher 的受 SQL 启发的语法，因为它提供了一种熟悉的结构来为图形应用程序编写查询。

openCypher 是一种用于属性图的声明式查询语言，最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。其语法在 Cypher 查询语言参考版本 9 中介绍。

有关 openCypher 规范在 Neptune 支持方面的限制和差异，请参阅亚马逊 Neptune 中符合 OpenCypher 规范。

> **Note**
>
> Cypher 查询语言的当前 Neo4j 实现在某些方面与 openCypher 规范有所不同。如果您要将当前的 Neo4j Cypher 代码迁移到 Neptune，请参阅Neptune 与 Neo4j 的兼容性和重写 Cypher 查询以在 Neptune 上的 openCypher 中运行以获取帮助。

从引擎版本 1.1.1.0 开始，openCypher 可在 Neptune 中用于生产用途。

Gremlin 与 openCypher：相似之处和不同之处

Gremlin 和 openCypher 都是属性图查询语言，它们在很多方面互补。

Gremlin 旨在吸引程序员并无缝融入代码。因此，从设计上讲，Gremlin 势在必行，而对于具有 SQL 或 SPARQL 经验的人来说，可能会更熟悉 openCypher 的声明式语法。对于在 Jupyter 笔记本中使用 Python 的数据科学家来说，Gremlin 似乎更自然，而对于具有一些 SQL 背景的企业用户来说，openCypher 可能看起来更直观。

好消息是，在 Neptune 中，您不必在 Gremlin 和 openCypher 之间进行选择。无论使用这两种语言中的哪一种来输入数据，任何一种语言的查询都可以在同一个图形上运行。您可能会发现在某些情况下使用 Gremlin 更方便，而在其它情况下使用 openCypher 更方便，这取决于您正在做的事情。

Gremlin 使用命令式语法，这允许您通过一系列步骤控制如何在图形中移动，每个步骤都接收数据流，对其执行某种操作（使用筛选条件、映射等），然后将结果输出到下一个步骤。Gremlin 查询通常采用 `g.V()` 形式，后跟其它步骤。

在 openCypher 中，您使用一种受 SQL 启发的声明式语法，此语法使用主旨语法（比如 ()-[]->() ）指定要在图形中查找的节点和关系的模式。openCypher 查询通常以 MATCH 子句开头，后跟其它子句，如 WHERE、WITH 和 RETURN。

开始使用 openCypher

无论如何加载，您都可以使用 openCypher 在 Neptune 中查询属性图数据，但不能使用 openCypher 来查询以 RDF 形式加载的数据。

Neptune 批量加载程序接受采用 Gremlin 的 CSV 格式和 openCypher 的 CSV 格式的属性图数据。当然，您也可以使用 Gremlin 和/或 openCypher 查询将属性数据添加到图形中。

有许多在线教程可用于学习 Cypher 查询语言。在这里，一些 openCypher 查询的简短示例可以帮助您了解这种语言，但是到目前为止，开始使用 openCypher 查询 Neptune 图形的最好、最简单的方法是在 Neptune Workbench 中使用 openCypher 笔记本。该工作台是开源的，托管在 GitHub https://github.com/aws-samples/amazon-neptune-samples。

你可以在 Neptune GitHub 图形笔记本存储库中找到 OpenCypher 笔记本。特别是，请查看 openCypher 的 Air-routes visualization 和 English Premier Teams 笔记本。

openCypher 处理的数据采用一系列无序的键/值映射的形式。完善、操作和增强这些映射的主要方法是使用子句对键/值对执行模式匹配、插入、更新和删除等任务。

openCypher 中有几个子句用于在图形中查找数据模式，其中 MATCH 最常用。MATCH 允许您指定要在图形中查找的节点、关系和筛选条件的模式。例如：

- 获取所有节点

```
MATCH (n) RETURN n
```

- 查找连接的节点

```
MATCH (n)-[r]->(d) RETURN n, r, d
```

- 查找路径

```
MATCH p=(n)-[r]->(d) RETURN p
```

- 获取所有带有标签的节点

```
MATCH (n:airport) RETURN n
```

请注意，上面的第一个查询返回图形中的每单个节点，接下来的两个查询返回每个具有关系的节点，但通常不建议这样做！几乎在所有情况下，您都希望缩小返回的数据的范围，这可以通过指定节点或关系标签和属性来实现，如第四个示例所示。

您可以在 Neptune github 示例存储库中找到 openCypher 语法的便捷备忘单。

Neptune openCypher 状态 servlet 和状态端点

openCypher 状态端点提供对服务器上当前正在运行或等待运行的查询的相关信息的访问。它还允许您取消这些查询。端点为：

```
https://(the server):(the port number)/openCypher/status
```

您可以使用 HTTP GET 和 POST 方法从服务器获取当前状态或取消查询。您也可以使用 DELETE 方法取消正在运行或等待的查询。

状态请求的参数

状态查询参数

- **includeWaiting**（true 或 false）- 当设置为 true 且其它参数不存在时，会导致返回等待的查询和正在运行的查询的状态信息。
- **cancelQuery** – 仅与 GET 和 POST 方法结合使用，以表示这是取消请求。DELETE 方法不需要此参数。

  不使用 cancelQuery 参数的值，但如果 cancelQuery 存在，则需要使用 queryId 参数来确定要取消哪个查询。
- **queryId** – 包含特定查询的 ID。

  与 GET 或 POST 方法结合使用且 cancelQuery 参数不存在时，queryId 会导致返回其标识的特定查询的状态信息。如果 cancelQuery 参数存在，则会取消 queryId 标识的特定查询。

  与 DELETE 方法结合使用时，queryId 始终表示要取消的特定查询。
- **silent** – 仅在取消查询时使用。如果设置为 true，则导致取消以静默方式进行。

状态请求响应字段

如果未提供特定查询的 ID，则为状态响应字段

- ac cepted QueryCount — 已接受但尚未完成的查询数量，包括队列中的查询。

- 正在运行 QueryCount — 当前正在运行的 OpenCypher 查询的数量。
- queries – 当前 openCypher 查询的列表。

特定查询的状态响应字段

- queryId – 查询的 GUID id。Neptune 为每个查询自动分配该 ID 值，或者您也可以分配自己的 ID（请参阅将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中）。
- queryString – 提交的查询。在超过 1024 个字符时，将截断为此长度。
- query EvalStats-此查询的统计信息：
  - waited – 表示查询等待了多长时间，以毫秒为单位。
  - elapsed – 到目前为止，查询已运行的毫秒数。
  - cancelled – `True` 表示查询已取消，或 `False` 表示尚未取消。

状态请求和响应示例

- 请求所有查询的状态，包括正在等待的查询：

```
curl https://server:port/openCypher/status \
  --data-urlencode "includeWaiting=true"
```

响应：

```
{
  "acceptedQueryCount" : 0,
  "runningQueryCount" : 0,
  "queries" : [ ]
}
```

- 请求正在运行的查询的状态，不 包括正在等待的查询：

```
curl https://server:port/openCypher/status
```

响应：

```
{
  "acceptedQueryCount" : 0,
  "runningQueryCount" : 0,
```

```
  "queries" : [ ]
}
```

- 请求单个查询的状态：

```
curl https://server:port/openCypher/status \
 --data-urlencode "queryId=eadc6eea-698b-4a2f-8554-5270ab17ebee"
```

响应：

```
{
  "queryId" : "eadc6eea-698b-4a2f-8554-5270ab17ebee",
  "queryString" : "MATCH (n1)-[:knows]->(n2), (n2)-[:knows]->(n3), (n3)-[:knows]-
>(n4), (n4)-[:knows]->(n5), (n5)-[:knows]->(n6), (n6)-[:knows]->(n7), (n7)-[:knows]-
>(n8), (n8)-[:knows]->(n9), (n9)-[:knows]->(n10) RETURN COUNT(n1);",
  "queryEvalStats" : {
    "waited" : 0,
    "elapsed" : 23463,
    "cancelled" : false
  }
}
```

- 请求取消查询

1. 使用 POST：

```
curl -X POST https://server:port/openCypher/status \
  --data-urlencode "cancelQuery" \
  --data-urlencode "queryId=f43ce17b-db01-4d37-a074-c76d1c26d7a9"
```

响应：

```
{
  "status" : "200 OK",
  "payload" : true
}
```

2. 使用 GET：

```
curl -X GET https://server:port/openCypher/status \
  --data-urlencode "cancelQuery" \
```

```
  --data-urlencode "queryId=588af350-cfde-4222-bee6-b9cedc87180d"
```

响应：

```
{
  "status" : "200 OK",
  "payload" : true
}
```

3. 使用 DELETE：

```
curl -X DELETE \
  -s "https://server:port/openCypher/status?queryId=b9a516d1-d25c-4301-
bb80-10b2743ecf0e"
```

响应：

```
{
  "status" : "200 OK",
  "payload" : true
}
```

Amazon Neptune openCypher HTTPS 端点

主题

- HTTPS 端点上的 openCypher 读取和写入查询
- 默认 openCypher JSON 结果格式

HTTPS 端点上的 openCypher 读取和写入查询

openCypher HTTPS 端点支持同时使用 GET 和 POST 方法执行读取和更新查询。不支持 DELETE 和 PUT 方法。

以下说明将带您演练使用 curl 命令和 HTTPS 连接到 openCypher 端点。必须从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

语法如下：

```
HTTPS://(the server):(the port number)/openCypher
```

以下是示例读取查询，一个使用 POST，一个使用 GET：

1. 使用 POST：

```
curl HTTPS://server:port/openCypher \
  -d "query=MATCH (n1) RETURN n1;"
```

2. 使用 GET（查询字符串采用 URL 编码）：

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=MATCH%20(n1)%20RETURN%20n1"
```

以下是示例写入/更新查询，一个使用 POST，一个使用 GET：

1. 使用 POST：

```
curl HTTPS://server:port/openCypher \
  -d "query=CREATE (n:Person { age: 25 })"
```

2. 使用 GET（查询字符串采用 URL 编码）：

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=CREATE%20(n%3APerson%20%7B%20age%3A
%2025%20%7D)"
```

默认 openCypher JSON 结果格式

默认情况下或通过将请求标头显式设置为 `Accept: application/json`，返回以下 JSON 格式。这种格式旨在使用大多数库的原生语言特征轻松解析为对象。

返回的 JSON 文档包含一个字段 `results`，其中包含查询返回值。以下示例显示了常用值的 JSON 格式。

值响应示例：

```
{
  "results": [
```

```
    {
      "count(a)": 121
    }
  ]
}
```

节点响应示例：

```
{
  "results": [
    {
      "a": {
        "~id": "22",
        "~entityType": "node",
        "~labels": [
          "airport"
        ],
        "~properties": {
          "desc": "Seattle-Tacoma",
          "lon": -122.30899810791,
          "runways": 3,
          "type": "airport",
          "country": "US",
          "region": "US-WA",
          "lat": 47.4490013122559,
          "elev": 432,
          "city": "Seattle",
          "icao": "KSEA",
          "code": "SEA",
          "longest": 11901
        }
      }
    }
  ]
}
```

关系响应示例：

```
{
  "results": [
    {
      "r": {
        "~id": "7389",
```

```
        "~entityType": "relationship",
        "~start": "22",
        "~end": "151",
        "~type": "route",
        "~properties": {
          "dist": 956
        }
      }
    }
  ]
}
```

路径响应示例：

```
{
  "results": [
    {
      "p": [
        {
          "~id": "22",
          "~entityType": "node",
          "~labels": [
            "airport"
          ],
          "~properties": {
            "desc": "Seattle-Tacoma",
            "lon": -122.30899810791,
            "runways": 3,
            "type": "airport",
            "country": "US",
            "region": "US-WA",
            "lat": 47.4490013122559,
            "elev": 432,
            "city": "Seattle",
            "icao": "KSEA",
            "code": "SEA",
            "longest": 11901
          }
        },
        {
          "~id": "7389",
          "~entityType": "relationship",
          "~start": "22",
```

```
          "~end": "151",
          "~type": "route",
          "~properties": {
            "dist": 956
          }
        },
        {
          "~id": "151",
          "~entityType": "node",
          "~labels": [
            "airport"
          ],
          "~properties": {
            "desc": "Ontario International Airport",
            "lon": -117.600997924805,
            "runways": 2,
            "type": "airport",
            "country": "US",
            "region": "US-CA",
            "lat": 34.0559997558594,
            "elev": 944,
            "city": "Ontario",
            "icao": "KONT",
            "code": "ONT",
            "longest": 12198
          }
        }
      ]
    }
  ]
}
```

使用 Bolt 协议向 Neptune 进行 openCypher 查询

Bolt 是一种面向语句的客户端/服务器协议，最初由Neo4j开发，并根据知识共享3.0署名许可进行许可。ShareAlike它由客户端驱动，这意味着客户端始终发起消息交换。

要使用 Neo4j 的 Bolt 驱动程序连接到 Neptune，只需使用 `bolt` URI 方案将 URI 和端口号替换为集群端点即可。如果您有单个 Neptune 实例在运行，请使用 read_write 端点。如果有多个实例在运行，则建议使用两个驱动程序，一个用于写入器，另一个用于所有只读副本。如果您只有两个默认端点，则 read_write 和 read_only 驱动程序就足够了，但是如果您还有自定义端点，可以考虑为每个端点创建一个驱动程序实例。

> **Note**
>
> 尽管 Bolt 规范规定 Bolt 可以使用 TCP 或 Bolt 进行连接 WebSockets，但 Neptune 仅支持 Bolt 的 TCP 连接。

Neptune 允许多达 1000 个并发 Bolt 连接。

有关使用 Bolt 驱动程序的各种语言的 openCypher 查询的示例，请参阅 Neo4j 驱动程序和语言指南文档。

> **Important**
>
> 适用于 Python、.NET 和 Golang 的 Neo4j Bolt 驱动程序最初不支持 Sign AWS ature v4 身份验证令牌的自动续订。 JavaScript这意味着在签名过期后（通常在 5 分钟内），驱动程序无法进行身份验证，随后的请求会失败。下面的 Python JavaScript、.NET 和 Go 示例都受到此问题的影响。
>
> 有关更多信息，请参阅 Neo4j Python 驱动程序问题 #834、Neo4j .NET 问题 #664、n eo4j JavaScript 驱动程序问题 #993 和 ne o4j GoLang 驱动程序问题 #429。
>
> 从驱动程序版本 5.8.0 开始，已为 Go 驱动程序发布了新的预览版重新身份验证 API（请参阅 v5.8.0 - 需要有关重新身份验证的反馈）。

使用 Bolt 以及 Java 连接到 Neptune

您可以从 Maven MVN 存储库中下载任何您想使用的版本的驱动程序，也可以将此依赖项添加到您的项目中：

```
<dependency>
  <groupId>org.neo4j.driver</groupId>
  <artifactId>neo4j-java-driver</artifactId>
  <version>4.3.3</version>
</dependency>
```

然后，要使用其中一个 Bolt 驱动程序连接到 Java 中的 Neptune，请使用如下代码为集群中的主/写入器实例创建一个驱动程序实例：

```
import org.neo4j.driver.Driver;
```

```
import org.neo4j.driver.GraphDatabase;

final Driver driver =
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
    AuthTokens.none(),
    Config.builder().withEncryption()
                    .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                    .build());
```

如果您有一个或多个读取器副本，则同样可以使用如下代码为它们创建驱动程序实例：

```
final Driver read_only_driver =              // (without connection timeout)
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
      Config.builder().withEncryption()
                      .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                      .build());
```

或者，对于超时：

```
final Driver read_only_timeout_driver =      // (with connection timeout)
  GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
    Config.builder().withConnectionTimeout(30, TimeUnit.SECONDS)
                    .withEncryption()
                    .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                    .build());
```

如果您有自定义端点，则可能还值得为每个端点创建一个驱动程序实例。

使用 Bolt 的 Python openCypher 查询示例

下面介绍如何使用 Bolt 在 Python 中进行 openCypher 查询：

```
python -m pip install neo4j
```

```
from neo4j import GraphDatabase
uri = "bolt://(your cluster endpoint URL):(your cluster port)"
driver = GraphDatabase.driver(uri, auth=("username", "password"), encrypted=True)
```

请注意，auth 参数会被忽略。

使用 Bolt 的 .NET openCypher 查询示例

要使用 Bolt 在.NET 中进行 OpenCypher 查询，第一步是使用安装 Neo4j 驱动程序。 NuHet要进行同步调用，请使用 .Simple 版本，如下所示：

```
Install-Package Neo4j.Driver.Simple-4.3.0
```

```
using Neo4j.Driver;

namespace hello
{
  // This example creates a node and reads a node in a Neptune
  // Cluster where IAM Authentication is not enabled.
  public class HelloWorldExample : IDisposable
  {
    private bool _disposed = false;
    private readonly IDriver _driver;
    private static string url = "bolt://(your cluster endpoint URL):(your cluster
 port)";
    private static string createNodeQuery = "CREATE (a:Greeting) SET a.message =
 'HelloWorldExample'";
    private static string readNodeQuery = "MATCH(n:Greeting) RETURN n.message";

    ~HelloWorldExample() => Dispose(false);

    public HelloWorldExample(string uri)
    {
      _driver = GraphDatabase.Driver(uri, AuthTokens.None, o =>
 o.WithEncryptionLevel(EncryptionLevel.Encrypted));
    }

    public void createNode()
    {
      // Open a session
      using (var session = _driver.Session())
      {
         // Run the query in a write transaction
        var greeting = session.WriteTransaction(tx =>
        {
          var result = tx.Run(createNodeQuery);
          // Consume the result
          return result.Consume();
        });
```

```
        // The output will look like this:
        //   ResultSummary{Query=`CREATE (a:Greeting) SET a.message =
'HelloWorldExample".....
        Console.WriteLine(greeting);
      }
    }

    public void retrieveNode()
    {
      // Open a session
      using (var session = _driver.Session())
      {
        // Run the query in a read transaction
        var greeting = session.ReadTransaction(tx =>
        {
          var result = tx.Run(readNodeQuery);
          // Consume the result. Read the single node
          // created in a previous step.
          return result.Single()[0].As<string>();
        });
        // The output will look like this:
        //   HelloWorldExample
        Console.WriteLine(greeting);
      }
    }

    public void Dispose()
    {
      Dispose(true);
      GC.SuppressFinalize(this);
    }

    protected virtual void Dispose(bool disposing)
    {
      if (_disposed)
        return;
      if (disposing)
      {
        _driver?.Dispose();
      }
      _disposed = true;
    }
```

```
    public static void Main()
    {
      using (var apiCaller = new HelloWorldExample(url))
      {
        apiCaller.createNode();
        apiCaller.retrieveNode();
      }
    }
  }
}
```

结合使用 Bolt 和 IAM 身份验证的 Java openCypher 查询示例

下面的 Java 代码显示如何结合使用 Bolt 和 IAM 身份验证在 Java 中进行 openCypher 查询。该 JavaDoc 评论描述了它的用法。一旦驱动程序实例可用，您就可以使用它发出多个经过身份验证的请求。

```
package software.amazon.neptune.bolt;

import com.amazonaws.DefaultRequest;
import com.amazonaws.Request;
import com.amazonaws.auth.AWS4Signer;
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.http.HttpMethodName;
import com.google.gson.Gson;
import lombok.Builder;
import lombok.Getter;
import lombok.NonNull;
import org.neo4j.driver.Value;
import org.neo4j.driver.Values;
import org.neo4j.driver.internal.security.InternalAuthToken;
import org.neo4j.driver.internal.value.StringValue;

import java.net.URI;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

import static com.amazonaws.auth.internal.SignerConstants.AUTHORIZATION;
import static com.amazonaws.auth.internal.SignerConstants.HOST;
import static com.amazonaws.auth.internal.SignerConstants.X_AMZ_DATE;
import static com.amazonaws.auth.internal.SignerConstants.X_AMZ_SECURITY_TOKEN;
```

```
/**
 * Use this class instead of `AuthTokens.basic` when working with an IAM
 * auth-enabled server. It works the same as `AuthTokens.basic` when using
 * static credentials, and avoids making requests with an expired signature
 * when using temporary credentials. Internally, it generates a new signature
 * on every invocation (this may change in a future implementation).
 *
 * Note that authentication happens only the first time for a pooled connection.
 *
 * Typical usage:
 *
 * NeptuneAuthToken authToken = NeptuneAuthToken.builder()
 *     .credentialsProvider(credentialsProvider)
 *     .region("aws region")
 *     .url("cluster endpoint url")
 *     .build();
 *
 * Driver driver = GraphDatabase.driver(
 *     authToken.getUrl(),
 *     authToken,
 *     config
 * );
 */

public class NeptuneAuthToken extends InternalAuthToken {
  private static final String SCHEME = "basic";
  private static final String REALM = "realm";
  private static final String SERVICE_NAME = "neptune-db";
  private static final String HTTP_METHOD_HDR = "HttpMethod";
  private static final String DUMMY_USERNAME = "username";
  @NonNull
  private final String region;
  @NonNull
  @Getter
  private final String url;
  @NonNull
  private final AWSCredentialsProvider credentialsProvider;
  private final Gson gson = new Gson();

  @Builder
  private NeptuneAuthToken(
      @NonNull final String region,
      @NonNull final String url,
      @NonNull final AWSCredentialsProvider credentialsProvider
```

```
  ) {
      // The superclass caches the result of toMap(), which we don't want
      super(Collections.emptyMap());
      this.region = region;
      this.url = url;
      this.credentialsProvider = credentialsProvider;
  }

  @Override
  public Map<String, Value> toMap() {
    final Map<String, Value> map = new HashMap<>();
    map.put(SCHEME_KEY, Values.value(SCHEME));
    map.put(PRINCIPAL_KEY, Values.value(DUMMY_USERNAME));
    map.put(CREDENTIALS_KEY, new StringValue(getSignedHeader()));
    map.put(REALM_KEY, Values.value(REALM));

    return map;
  }

  private String getSignedHeader() {
    final Request<Void> request = new DefaultRequest<>(SERVICE_NAME);
    request.setHttpMethod(HttpMethodName.GET);
    request.setEndpoint(URI.create(url));
    // Comment out the following line if you're using an engine version older than
1.2.0.0
    request.setResourcePath("/opencypher");

    final AWS4Signer signer = new AWS4Signer();
    signer.setRegionName(region);
    signer.setServiceName(request.getServiceName());
    signer.sign(request, credentialsProvider.getCredentials());

    return getAuthInfoJson(request);
  }

  private String getAuthInfoJson(final Request<Void> request) {
    final Map<String, Object> obj = new HashMap<>();
    obj.put(AUTHORIZATION, request.getHeaders().get(AUTHORIZATION));
    obj.put(HTTP_METHOD_HDR, request.getHttpMethod());
    obj.put(X_AMZ_DATE, request.getHeaders().get(X_AMZ_DATE));
    obj.put(HOST, request.getHeaders().get(HOST));
    obj.put(X_AMZ_SECURITY_TOKEN, request.getHeaders().get(X_AMZ_SECURITY_TOKEN));

    return gson.toJson(obj);
```

```
  }
}
```

结合使用 Bolt 和 IAM 身份验证的 Python openCypher 查询示例

下面的 Python 类允许您结合使用 Bolt 和 IAM 身份验证在 Python 中进行 openCypher 查询：

```
import json

from neo4j import Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import Credentials
from botocore.auth import (
  SigV4Auth,
  _host_from_url,
)

SCHEME = "basic"
REALM = "realm"
SERVICE_NAME = "neptune-db"
DUMMY_USERNAME = "username"
HTTP_METHOD_HDR = "HttpMethod"
HTTP_METHOD = "GET"
AUTHORIZATION = "Authorization"
X_AMZ_DATE = "X-Amz-Date"
X_AMZ_SECURITY_TOKEN = "X-Amz-Security-Token"
HOST = "Host"


class NeptuneAuthToken(Auth):
  def __init__(
    self,
    credentials: Credentials,
    region: str,
    url: str,
    **parameters
  ):
    # Do NOT add "/opencypher" in the line below if you're using an engine version
 older than 1.2.0.0
    request = AWSRequest(method=HTTP_METHOD, url=url + "/opencypher")
    request.headers.add_header("Host", _host_from_url(request.url))
    sigv4 = SigV4Auth(credentials, SERVICE_NAME, region)
    sigv4.add_auth(request)
```

```
    auth_obj = {
      hdr: request.headers[hdr]
      for hdr in [AUTHORIZATION, X_AMZ_DATE, X_AMZ_SECURITY_TOKEN, HOST]
    }
    auth_obj[HTTP_METHOD_HDR] = request.method
    creds: str = json.dumps(auth_obj)
    super().__init__(SCHEME, DUMMY_USERNAME, creds, REALM, **parameters)
```

您可以使用该类创建驱动程序，如下所示：

```
  authToken = NeptuneAuthToken(creds, REGION, URL)
  driver = GraphDatabase.driver(URL, auth=authToken, encrypted=True)
```

结合使用 IAM 身份验证和 Bolt 的 Node.js 示例

下面的 Node.js 代码使用 JavaScript 版本 3 的 AWS SDK 和 ES6 语法来创建对请求进行身份验证的驱动程序：

```
import neo4j from "neo4j-driver";
import { HttpRequest }  from "@aws-sdk/protocol-http";
import { defaultProvider } from "@aws-sdk/credential-provider-node";
import { SignatureV4 } from "@aws-sdk/signature-v4";
import crypto from "@aws-crypto/sha256-js";
const { Sha256 } = crypto;
import assert from "node:assert";

const region = "us-west-2";
const serviceName = "neptune-db";
const host = "(your cluster endpoint URL)";
const port = 8182;
const protocol = "bolt";
const hostPort = host + ":" + port;
const url = protocol + "://" + hostPort;
const createQuery = "CREATE (n:Greeting {message: 'Hello'}) RETURN ID(n)";
const readQuery = "MATCH(n:Greeting) WHERE ID(n) = $id RETURN n.message";

async function signedHeader() {
  const req = new HttpRequest({
    method: "GET",
    protocol: protocol,
    hostname: host,
    port: port,
```

```
    // Comment out the following line if you're using an engine version older than
 1.2.0.0
    path: "/opencypher",
    headers: {
      host: hostPort
    }
  });

  const signer = new SignatureV4({
    credentials: defaultProvider(),
    region: region,
    service: serviceName,
    sha256: Sha256
  });

  return signer.sign(req, { unsignableHeaders: new Set(["x-amz-content-sha256"]) })
    .then((signedRequest) => {
      const authInfo = {
        "Authorization": signedRequest.headers["authorization"],
        "HttpMethod": signedRequest.method,
        "X-Amz-Date": signedRequest.headers["x-amz-date"],
        "Host": signedRequest.headers["host"],
        "X-Amz-Security-Token": signedRequest.headers["x-amz-security-token"]
      };
      return JSON.stringify(authInfo);
    });
}

async function createDriver() {
  let authToken = { scheme: "basic", realm: "realm", principal: "username",
 credentials: await signedHeader() };

  return neo4j.driver(url, authToken, {
      encrypted: "ENCRYPTION_ON",
      trust: "TRUST_SYSTEM_CA_SIGNED_CERTIFICATES",
      maxConnectionPoolSize: 1,
      // logging: neo4j.logging.console("debug")
    }
  );
}

function unmanagedTxn(driver) {
  const session = driver.session();
  const tx = session.beginTransaction();
```

```
  tx.run(createQuery)
  .then((res) => {
    const id = res.records[0].get(0);
    return tx.run(readQuery, { id: id });
  })
  .then((res) => {
    // All good, the transaction will be committed
    const msg = res.records[0].get("n.message");
    assert.equal(msg, "Hello");
  })
  .catch(err => {
    // The transaction will be rolled back, now handle the error.
    console.log(err);
  })
  .then(() => session.close());
}

createDriver()
.then((driver) => {
  unmanagedTxn(driver);
  driver.close();
})
.catch((err) => {
  console.log(err);
});
```

结合使用 Bolt 和 IAM 身份验证的 .NET openCypher 查询示例

要在 .NET 中启用 IAM 身份验证，您需要在建立连接时签署请求。以下示例显示了如何创建用于生成身份验证令牌的 NeptuneAuthToken 帮助程序：

```
using Amazon.Runtime;
using Amazon.Util;
using Neo4j.Driver;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
using System.Web;

namespace Hello
{
  /*
```

```
  * Use this class instead of `AuthTokens.None` when working with an IAM-auth-enabled
server.
  *
  * Note that authentication happens only the first time for a pooled connection.
  *
  * Typical usage:
  *
  * var authToken = new NeptuneAuthToken(AccessKey, SecretKey,
Region).GetAuthToken(Host);
  * _driver = GraphDatabase.Driver(Url, authToken, o =>
o.WithEncryptionLevel(EncryptionLevel.Encrypted));
  */

 public class NeptuneAuthToken
 {
   private const string ServiceName = "neptune-db";
   private const string Scheme = "basic";
   private const string Realm = "realm";
   private const string DummyUserName = "username";
   private const string Algorithm = "AWS4-HMAC-SHA256";
   private const string AWSRequest = "aws4_request";

   private readonly string _accessKey;
   private readonly string _secretKey;
   private readonly string _region;

   private readonly string _emptyPayloadHash;

   private readonly SHA256 _sha256;


   public NeptuneAuthToken(string awsKey = null, string secretKey = null, string
region = null)
   {
     var awsCredentials = awsKey == null || secretKey == null
       ? FallbackCredentialsFactory.GetCredentials().GetCredentials()
       : null;

     _accessKey = awsKey ?? awsCredentials.AccessKey;
     _secretKey = secretKey ?? awsCredentials.SecretKey;
     _region = region ?? FallbackRegionFactory.GetRegionEndpoint().SystemName; //ex:
us-east-1

     _sha256 = SHA256.Create();
```

```
      _emptyPayloadHash = Hash(Array.Empty<byte>());
    }

    public IAuthToken GetAuthToken(string url)
    {
      return AuthTokens.Custom(DummyUserName, GetCredentials(url), Realm, Scheme);
    }

    /******************* AWS SIGNING FUNCTIONS ********************/
    private string Hash(byte[] bytesToHash)
    {
      return ToHexString(_sha256.ComputeHash(bytesToHash));
    }

    private static byte[] HmacSHA256(byte[] key, string data)
    {
      return new HMACSHA256(key).ComputeHash(Encoding.UTF8.GetBytes(data));
    }

    private byte[] GetSignatureKey(string dateStamp)
    {
      var kSecret = Encoding.UTF8.GetBytes($"AWS4{_secretKey}");
      var kDate = HmacSHA256(kSecret, dateStamp);
      var kRegion = HmacSHA256(kDate, _region);
      var kService = HmacSHA256(kRegion, ServiceName);
      return HmacSHA256(kService, AWSRequest);
    }

    private static string ToHexString(byte[] array)
    {
      return Convert.ToHexString(array).ToLowerInvariant();
    }

    private string GetCredentials(string url)
    {
      var request = new HttpRequestMessage
      {
        Method = HttpMethod.Get,
        RequestUri = new Uri($"https://{url}/opencypher")
      };

      var signedrequest = Sign(request);

      var headers = new Dictionary<string, object>
```

```
      {
        [HeaderKeys.AuthorizationHeader] =
signedrequest.Headers.GetValues(HeaderKeys.AuthorizationHeader).FirstOrDefault(),
        ["HttpMethod"] = HttpMethod.Get.ToString(),
        [HeaderKeys.XAmzDateHeader] =
signedrequest.Headers.GetValues(HeaderKeys.XAmzDateHeader).FirstOrDefault(),
        // Host should be capitalized, not like in Amazon.Util.HeaderKeys.HostHeader
        ["Host"] =
signedrequest.Headers.GetValues(HeaderKeys.HostHeader).FirstOrDefault(),
      };

      return JsonSerializer.Serialize(headers);
    }

    private HttpRequestMessage Sign(HttpRequestMessage request)
    {
      var now = DateTimeOffset.UtcNow;
      var amzdate = now.ToString("yyyyMMddTHHmmssZ");
      var datestamp = now.ToString("yyyyMMdd");

      if (request.Headers.Host == null)
      {
        request.Headers.Host = $"{request.RequestUri.Host}:{request.RequestUri.Port}";
      }

      request.Headers.Add(HeaderKeys.XAmzDateHeader, amzdate);

      var canonicalQueryParams = GetCanonicalQueryParams(request);

      var canonicalRequest = new StringBuilder();
      canonicalRequest.Append(request.Method + "\n");
      canonicalRequest.Append(request.RequestUri.AbsolutePath + "\n");
      canonicalRequest.Append(canonicalQueryParams + "\n");

      var signedHeadersList = new List<string>();
      foreach (var header in request.Headers.OrderBy(a => a.Key.ToLowerInvariant()))
      {
        canonicalRequest.Append(header.Key.ToLowerInvariant());
        canonicalRequest.Append(':');
        canonicalRequest.Append(string.Join(",", header.Value.Select(s => s.Trim())));
        canonicalRequest.Append('\n');
        signedHeadersList.Add(header.Key.ToLowerInvariant());
      }
      canonicalRequest.Append('\n');
```

```
      var signedHeaders = string.Join(";", signedHeadersList);
      canonicalRequest.Append(signedHeaders + "\n");
      canonicalRequest.Append(_emptyPayloadHash);

      var credentialScope = $"{datestamp}/{_region}/{ServiceName}/{AWSRequest}";
      var stringToSign = $"{Algorithm}\n{amzdate}\n{credentialScope}\n"
        + Hash(Encoding.UTF8.GetBytes(canonicalRequest.ToString()));

      var signing_key = GetSignatureKey(datestamp);
      var signature = ToHexString(HmacSHA256(signing_key, stringToSign));

      request.Headers.TryAddWithoutValidation(HeaderKeys.AuthorizationHeader,
        $"{Algorithm} Credential={_accessKey}/{credentialScope},
 SignedHeaders={signedHeaders}, Signature={signature}");

      return request;
    }

    private static string GetCanonicalQueryParams(HttpRequestMessage request)
    {
      var querystring = HttpUtility.ParseQueryString(request.RequestUri.Query);

      // Query params must be escaped in upper case (i.e. "%2C", not "%2c").
      var queryParams = querystring.AllKeys.OrderBy(a => a)
        .Select(key => $"{key}={Uri.EscapeDataString(querystring[key])}");
      return string.Join("&", queryParams);
    }
  }
}
```

下面介绍了如何结合使用 Bolt 和 IAM 身份验证在 .NET 中进行 openCypher 查询。下面的示例使用 NeptuneAuthToken 帮助程序：

```
using Neo4j.Driver;

namespace Hello
{
  public class HelloWorldExample
  {
    private const string Host = "(your hostname):8182";
    private const string Url = $"bolt://{Host}";
```

```
   private const string CreateNodeQuery = "CREATE (a:Greeting) SET a.message =
'HelloWorldExample'";
   private const string ReadNodeQuery = "MATCH(n:Greeting) RETURN n.message";

   private const string AccessKey = "(your access key)";
   private const string SecretKey = "(your secret key)";
   private const string Region = "(your AWS region)"; // e.g. "us-west-2"

   private readonly IDriver _driver;

   public HelloWorldExample()
   {
     var authToken = new NeptuneAuthToken(AccessKey, SecretKey,
Region).GetAuthToken(Host);

     // Note that when the connection is reinitialized after max connection lifetime
     // has been reached, the signature token could have already been expired (usually
5 min)
     // You can face exceptions like:
     //   `Unexpected server exception 'Signature expired: XXXX is now earlier than
YYYY (ZZZZ - 5 min.)`
     _driver = GraphDatabase.Driver(Url, authToken, o =>

o.WithMaxConnectionLifetime(TimeSpan.FromMinutes(60)).WithEncryptionLevel(EncryptionLevel.Encr
   }

   public async Task CreateNode()
   {
     // Open a session
     using (var session = _driver.AsyncSession())
     {
       // Run the query in a write transaction
       var greeting = await session.WriteTransactionAsync(async tx =>
       {
         var result = await tx.RunAsync(CreateNodeQuery);
         // Consume the result
         return await result.ConsumeAsync();
       });

       // The output will look like this:
       //   ResultSummary{Query=`CREATE (a:Greeting) SET a.message =
'HelloWorldExample".....
       Console.WriteLine(greeting.Query);
     }
```

```
    }

    public async Task RetrieveNode()
    {
      // Open a session
      using (var session = _driver.AsyncSession())
      {
        // Run the query in a read transaction
        var greeting = await session.ReadTransactionAsync(async tx =>
        {
          var result = await tx.RunAsync(ReadNodeQuery);
          var records = await result.ToListAsync();

          // Consume the result. Read the single node
          // created in a previous step.
          return records[0].Values.First().Value;
        });
        // The output will look like this:
        //   HelloWorldExample
        Console.WriteLine(greeting);
      }
    }
  }
}
```

此示例可以通过使用以下软件包在 .NET 6 或 .NET 7 上运行以下代码来启动：

- **Neo4j**.Driver=4.3.0
- **AWSSDK**.Core=3.7.102.1

```
namespace Hello
{
  class Program
  {
    static async Task Main()
    {
      var apiCaller = new HelloWorldExample();

      await apiCaller.CreateNode();
      await apiCaller.RetrieveNode();
    }
  }
```

```
}
```

结合使用 Bolt 和 IAM 身份验证的 Golang openCypher 查询示例

下面的 Golang 软件包展示了如何结合使用 Bolt 和 IAM 身份验证以 Go 语言进行 openCypher 查询：

```
package main

import (
  "context"
  "encoding/json"
  "fmt"
  "github.com/aws/aws-sdk-go/aws/credentials"
  "github.com/aws/aws-sdk-go/aws/signer/v4"
  "github.com/neo4j/neo4j-go-driver/v5/neo4j"
  "log"
  "net/http"
  "os"
  "time"
)

const (
  ServiceName   = "neptune-db"
  DummyUsername = "username"
)

// Find node by id using Go driver
func findNode(ctx context.Context, region string, hostAndPort string, nodeId string)
 (string, error) {
  req, err := http.NewRequest(http.MethodGet, "https://"+hostAndPort+"/opencypher",
 nil)

  if err != nil {
    return "", fmt.Errorf("error creating request, %v", err)
  }

  // credentials must have been exported as environment variables
  signer := v4.NewSigner(credentials.NewEnvCredentials())
  _, err = signer.Sign(req, nil, ServiceName, region, time.Now())

  if err != nil {
    return "", fmt.Errorf("error signing request: %v", err)
  }
```

```
  hdrs := []string{"Authorization", "X-Amz-Date", "X-Amz-Security-Token"}
  hdrMap := make(map[string]string)
  for _, h := range hdrs {
    hdrMap[h] = req.Header.Get(h)
  }

  hdrMap["Host"] = req.Host
  hdrMap["HttpMethod"] = req.Method

  password, err := json.Marshal(hdrMap)
  if err != nil {
    return "", fmt.Errorf("error creating JSON, %v", err)
  }
  authToken := neo4j.BasicAuth(DummyUsername, string(password), "")
  // +s enables encryption with a full certificate check
  // Use +ssc to disable client side TLS verification
  driver, err := neo4j.NewDriverWithContext("bolt+s://"+hostAndPort+"/opencypher",
authToken)
  if err != nil {
    return "", fmt.Errorf("error creating driver, %v", err)
  }

  defer driver.Close(ctx)

  if err := driver.VerifyConnectivity(ctx); err != nil {
    log.Fatalf("failed to verify connection, %v", err)
  }

  config := neo4j.SessionConfig{}

  session := driver.NewSession(ctx, config)
  defer session.Close(ctx)

  result, err := session.Run(
    ctx,
    fmt.Sprintf("MATCH (n) WHERE ID(n) = '%s' RETURN n", nodeId),
    map[string]any{},
  )
  if err != nil {
    return "", fmt.Errorf("error running query, %v", err)
  }

  if !result.Next(ctx) {
```

```
    return "", fmt.Errorf("node not found")
  }

  n, found := result.Record().Get("n")
  if !found {
    return "", fmt.Errorf("node not found")
  }

  return fmt.Sprintf("+%v\n", n), nil
}

func main() {
  if len(os.Args) < 3 {
    log.Fatal("Usage: go main.go (region) (host and port)")
  }
  region := os.Args[1]
  hostAndPort := os.Args[2]
  ctx := context.Background()

  res, err := findNode(ctx, region, hostAndPort,
 "72c2e8c1-7d5f-5f30-10ca-9d2bb8c4afbc")
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(res)
}
```

Neptune 中的 Bolt 连接行为

以下是有关 Neptune Bolt 连接的一些注意事项：

- 由于 Bolt 连接是在 TCP 层创建的，因此您不能像使用 HTTP 端点那样在它们前面使用应用程序负载均衡器。
- Neptune 用于 Bolt 连接的端口是您的数据库集群的端口。
- 根据传递给它的 Bolt 序言，Neptune 服务器会选择最合适的 Bolt 版本（1、2、3 或 4.0）。
- 客户端在任何时间点可以打开的与 Neptune 服务器的最大连接数为 1000。
- 如果客户端在查询后没有关闭连接，则该连接可用于执行下一个查询。
- 但是，如果连接空闲了 20 分钟，服务器会自动将其关闭。
- 如果未启用 IAM 身份验证，则可以使用 `AuthTokens.none()`，而不是提供虚拟用户名和密码。例如，在 Java 中：

```
GraphDatabase.driver("bolt://(your cluster endpoint URL):(your cluster port)",
 AuthTokens.none(),

 Config.builder().withEncryption().withTrustStrategy(TrustStrategy.trustSystemCertificates())
```

- 启用 IAM 身份验证后，如果 Bolt 连接由于某种其它原因尚未关闭，则始终会在建立连接 10 天后的几分钟内断开连接。
- 如果客户端在没有消耗先前查询的结果的情况下通过连接发送查询以供执行，则新查询将被丢弃。要改为丢弃之前的结果，客户端必须通过连接发送重置消息。
- 对于给定连接，一次只能创建一个事务。
- 如果在事务期间发生异常，Neptune 服务器会回滚事务并关闭连接。在这种情况下，驱动程序会为下一个查询创建一个新连接。
- 请注意，会话不是线程安全的。多个并行操作必须使用多个单独的会话。

openCypher 参数化查询示例

Neptune 支持参数化的 openCypher 查询。这样，您就可以通过不同的参数多次使用相同的查询结构。由于查询结构不会变化，Neptune 可以缓存其抽象语法树 (AST)，而不必多次解析它。

使用 HTTPS 端点进行的 openCypher 参数化查询示例

以下是在 Neptune openCypher HTTPS 端点上使用参数化查询的示例。查询为：

```
MATCH (n {name: $name, age: $age})
RETURN n
```

参数定义如下：

```
parameters={"name": "john", "age": 20}
```

使用 GET，您可以按如下所示提交参数化查询：

```
curl -k \
  "https://localhost:8182/openCypher?query=MATCH%20%28n%20%7Bname:\$name,age:\$age%7D
%29%20RETURN%20n&parameters=%7B%22name%22:%22john%22,%22age%22:20%7D"
```

或者，您也可以使用 POST：

```
curl -k \
  https://localhost:8182/openCypher \
  -d "query=MATCH (n {name: \$name, age: \$age}) RETURN n" \
  -d "parameters={\"name\": \"john\", \"age\": 20}"
```

或者，使用 DIRECT POST：

```
curl -k \
   -H "Content-Type: application/opencypher" \
  "https://localhost:8182/openCypher?parameters=%7B%22name%22:%22john%22,%22age
%22:20%7D" \
  -d "MATCH (n {name: \$name, age: \$age}) RETURN n"
```

使用 Bolt 进行 openCypher 参数化查询的示例

以下是使用 Bolt 协议进行 openCypher 参数化查询的 Python 示例：

```
from neo4j import GraphDatabase
uri = "bolt://[neptune-endpoint-url]:8182"
driver = GraphDatabase.driver(uri, auth=("", ""))

def match_name_and_age(tx, name, age):
  # Parameterized Query
  tx.run("MATCH (n {name: $name, age: $age}) RETURN n", name=name, age=age)

with driver.session() as session:
  # Parameters
  session.read_transaction(match_name_and_age, "john", 20)

driver.close()
```

以下是使用 Bolt 协议进行 openCypher 参数化查询的 Java 示例：

```
Driver driver = GraphDatabase.driver("bolt+s://(your cluster endpoint URL):8182");
HashMap<String, Object> parameters = new HashMap<>();
parameters.put("name", "john");
parameters.put("age", 20);
String queryString = "MATCH (n {name: $name, age: $age}) RETURN n";
Result result = driver.session().run(queryString, parameters);
```

openCypher 数据模型

Neptune openCypher 引擎建立在与 Gremlin 相同的属性图模型之上。具体而言：

- 每个节点都有一个或多个标签。如果您插入一个没有标签的节点，则会附加一个名为 `vertex` 的默认标签。如果您尝试删除节点的所有标签，则会引发错误。
- 关系是一个正好只有一种关系类型的实体，它在两个节点之间形成单向连接（即从其中一个节点到另一个节点）。
- 节点和关系都可以具有属性，但不是必须具有属性。Neptune 支持具有零个属性的节点和关系。
- Neptune 不支持元属性，这些元属性也未包含在 openCypher 规范中。
- 如果图形中的属性是使用 Gremlin 创建的，则它们可以是多值的。也就是说，节点或关系属性可以有一组不同的值，而不仅仅是一个值。Neptune 扩展了 openCypher 语义，以从容地处理多值属性。

openCypher 数据格式中记录了支持的数据类型。但是，我们目前不建议将 `Array` 属性值插入到 openCypher 图形中。尽管可以使用批量加载程序插入数组属性值，但当前 Neptune openCypher 版本将其视为一组多值属性，而不是单个列表值。

以下是此版本支持的数据类型列表：

- `Bool`
- `Byte`
- `Short`
- `Int`
- `Long`
- `Float`（包括正负无穷大和 NaN，但不包括 INF）
- `Double`（包括正负无穷大和 NaN，但不包括 INF）
- `DateTime`
- `String`

openCypher **`explain`** 特征

openCypher `explain` 特征是 Amazon Neptune 中的一种自助式工具，可帮助您了解 Neptune 引擎所采用的执行方法。要调用 explain，您需要使用 `explain=`*`mode`* 向 openCypher HTTPS 请求传递一个参数，其中 *`mode`* 值可以是以下值之一：

- **static** – 在 static 模式下，explain 仅输出查询计划的静态结构。它不实际运行查询。
- **dynamic** – 在 dynamic 模式下，explain 还会运行查询，并包括查询计划的动态方面。这些方面可能包括流经运算符的中间绑定的数量、传入绑定与传出绑定的比率以及每个运算符使用的总时间。
- **details** – 在 details 模式下，explain 输出动态模式中显示的信息以及其它详细信息，例如实际的 openCypher 查询字符串和某个连接运算符之下模式的估计范围计数。

例如，使用 POST：

```
curl HTTPS://server:port/openCypher \
  -d "query=MATCH (n) RETURN n LIMIT 1;" \
  -d "explain=dynamic"
```

或者，使用 GET：

```
curl -X GET \
  "HTTPS://server:port/openCypher?query=MATCH%20(n)%20RETURN%20n%20LIMIT
%201&explain=dynamic"
```

Neptune 中 openCypher **explain** 的限制

openCypher Explain 的当前版本存在以下限制：

- Explain 计划目前仅适用于执行只读操作的查询。不支持执行任何类型的突变（例如 CREATE、DELETE、MERGE、SET 等）的查询。
- 特定计划的运算符和输出可能会在将来版本中发生变化。

openCypher **explain** 输出中的 DFE 运算符

要使用 openCypher explain 特征提供的信息，您需要了解有关 DFE 查询引擎工作原理的一些详细信息（DFE 是 Neptune 用来处理 openCypher 查询的引擎）。

DFE 引擎将每个查询转换到一个运算符管道中。从第一个运算符开始，中间解将通过这个运算符管道从一个运算符流到下一个运算符。Explain 表中的每一行表示一个结果（截至到评估点）。

可以出现在 DFE 查询计划中的运算符如下所示：

DFEApply - 对存储在指定变量中的值执行在参数部分中指定的函数

DFE BindRelation-将具有指定名称的变量绑定在一起

DFE ChunkLocal SubQuery — 这是一种非阻塞操作，充当正在执行的子查询的封装器。

DFE DistinctColumn-根据指定的变量返回输入值的不同子集。

DFE DistinctRelation-根据指定的变量返回输入解决方案的不同子集。

DFEDrain – 出现在子查询的末尾，用作该子查询的终止步骤。解的数量记录为 `Units In`。`Units Out` 始终为零。

DFE ForwardValue — 将所有输入区块直接复制为输出块，然后传递给其下游运算符。

DFE GroupBy HashIndex — 根据先前计算的哈希索引（使用运算）对输入解执行分组运算。`DFEHashIndexBuild`作为输出，给定输入通过包含每个输入解的组键的列进行扩展。

DFE B HashIndex uild — 在一组变量上构建哈希索引作为副作用。此哈希索引通常会在以后的操作中重复使用。有关此哈希索引可能用在何处，请参阅 `DFEHashIndexJoin` 或 `DFEGroupByHashIndex`。

DFE HashIndex Join — 根据先前构建的哈希索引对传入的解决方案执行联接。有关此哈希索引可能在何处构建，请参阅 `DFEHashIndexBuild`。

DFE JoinExists — 采用左手和右手输入关系，并保留左侧关系中具有相应值的值，这些值由给定连接变量定义。

- 这是一种非阻塞操作，它充当子查询的包装器，允许重复运行子查询以在循环中使用。

DFE MergeChunks — 这是一种阻塞操作，它将来自上游运算符的区块组合成单个解决方案块，然后传递给其下游运算符（相反）。`DFESplitChunks`

DFEMinus - 获取左侧和右侧输入关系，并保留左侧输入关系中的值，这些值在由给定联接变量定义的右侧输入关系中不具有相应的值。如果两个关系中的变量没有重叠，则此运算符只返回左侧输入关系。

DFE NotExists — 采用左手和右手输入关系，并保留左侧关系中在给定连接变量定义的右关系中没有相应值的值。如果两个关系中的变量没有重叠，则此运算符会返回空关系。

DFE OptionalJoin — 执行左外联接（也称为 OPTIONAL join）：来自左侧的解决方案如果右侧至少有一个加入伙伴，则按原样转发来自左侧的解决方案，而右侧没有加入伙伴的解决方案则按原样转发。这是一项阻止操作。

DFE PipelineJoin — 根据参数定义的元组模式连接输入。`pattern`

DFE C PipelineRange ount — 计算与给定模式匹配的解的数量，并返回包含计数值的单个一元解。

DFE PipelineScan — 扫描数据库中的给定`pattern`参数，无论是否对列使用给定过滤器。

DFEProject – 采用多个输入列并仅投影所需的列。

DFEReduce – 对指定的变量执行指定的聚合函数。

DFE RelationalJoin — 使用合并联接根据指定的模式键加入前一个运算符的输入。这是一项阻止操作。

DFE RouteChunks — 从其单个传入边缘获取输入块，然后沿着其多个传出边缘路由这些块。

DFE SelectRows — 此运算符有选择地从其左侧输入关系解中提取行，然后转发给其下游运算符。根据运算符右侧输入关系中提供的行标识符选择的行。

DFESerialize - 将查询的最终结果序列化为 JSON 字符串序列化，将每个输入解映射到相应的变量名称。对于节点和边缘结果，这些结果将序列化为实体属性和元数据的映射。

DFESort - 获取输入关系并根据提供的排序键生成排序关系。

DFE SplitBy Grou p — 将来自一个传入边的每个输入块拆分为较小的输出块，这些块对应于行组，行组由来自另一个传入边的相应输入区块的行 ID 标识。

DFE SplitChunks — 将每个输入块拆分为较小的输出块（相反）。`DFEMergeChunks`

DFE StreamingHash IndexBuild —直播版本的. `DFEHashIndexBuild`

DFE StreamingGroup ByHash 索引 —直播版本. `DFEGroupByHashIndex`

DFESubquery – 此运算符出现在所有计划的开头，它封装计划中在 DFE 引擎上运行的部分，这是 openCypher 的整个计划。

DFE SymmetricHash Join — 使用哈希联接根据指定的模式键加入前一个运算符的输入。这是一个非阻止操作。

DFESync - 此运算符是支持非阻塞计划的同步运算符。它从两个传入边缘获取解，然后将这些解转发到相应的下游边缘。出于同步目的，可以对其中一个边缘的输入进行内部缓冲。

DFETee – 这是一个分支运算符，它向多个运算符发送相同的解集。

DFE TermResolution — 对其输入执行本地化或全局化操作，生成分别包含本地化或全球化标识符的列。

- 将输入列中的值列表作为单个元素展开到输出列。

DFEUnion - 获取两个或多个输入关系，并使用所需的输出架构生成这些关系的并集。

SolutionInjection— 显示在 explain 输出中的其他所有内容之前，在 Units Out 列中的值为 1。但是，它不起作用，实际上并没有向 DFE 引擎注入任何解。

TermResolution— 出现在计划末尾，将海王星引擎中的对象转换为 OpenCypher 对象。

openCypher **explain** 输出中的列

Neptune 作为 openCypher explain 输出生成的查询计划信息包含每行具有一个运算符的表。此表包含以下各列：

ID – 计划中此运算符的数字 ID。

Out #1（和 Out #2）– 此运算符下游的运算符的 ID。最多可以有两个下游运算符。

Name – 此运算符的名称。

Arguments – 运算符的任何相关详细信息。这包括如输入架构、输出架构、模式（对于 `PipelineScan` 和 `PipelineJoin`）等。

Mode – 描述基本运算符行为的标签。此列一般为空白 (-)。一个例外是 `TermResolution`，其中模式可以是 `id2value_opencypher`，表示从 ID 到 openCypher 值的解。

Units In – 作为输入传递给该运算符的解的数量。没有上游运算符的运算符（例如 `DFEPipelineScan`、`SolutionInjections` 和未注入静态值的 `DFESubquery`）的值为零。

Units Out – 作为该运算符的输出而生成的解的数量。`DFEDrain` 是一种特殊情况，其中被排除的解的数量记录在 `Units In` 中，并且 `Units Out` 始终为零。

Ratio – `Units Out` 与 `Units In` 的比率。

Time (ms) – 此运算符消耗的 CPU 时间，以毫秒为单位。

openCypher explain 输出的一个基本示例

以下是 openCypher `explain` 输出的基本示例。该查询是在航线数据集中针对机场代码为 `ATL` 的节点执行的单节点查找，它使用默认 ASCII 输出格式的 `details` 模式调用 `explain`：

```
curl -d "query=MATCH (n {code: 'ATL'}) RETURN n" -k https://localhost:8182/openCypher -
d "explain=details"
                                    ~

Query:
MATCH (n {code: 'ATL'}) RETURN n

################################################################################################
```

```
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
#########################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 1         # 0.00  # 4.00      #
#########################################################################################
# 2  # -      # -      # TermResolution    # vars=[?n]          # id2value_opencypher #
 1        # 1         # 1.00  # 2.00      #
#########################################################################################


subQuery1
#########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFEPipelineScan       # pattern=Node(?n) with property 'code'
 as ?n_code2 and label 'ALL'                                       # -    # 0
   # 1         # 0.00  # 0.21      #
#    #        #        #                      # inlineFilters=[(?n_code2 IN
 ["ATL"^^xsd:string])]                                                  #
  #          #           #       #           #
#    #        #        #                        # patternEstimate=1
                                                                 #      #
  #           #       #           #
#########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#9d84f97c-c3b0-459a-98d5-955a8726b159/graph_1 # -    #
 1        # 1         # 1.00  # 0.04      #
#########################################################################################
# 2  # 3      # -      # DFEProject            # columns=[?n]
                                                                  # -    # 1
  # 1         # 1.00  # 0.04      #
#########################################################################################
# 3  # -      # -      # DFEDrain              # -
                                                                  # -    # 1
  # 0         # 0.00  # 0.03      #
#########################################################################################
```

```
subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#9d84f97c-
c3b0-459a-98d5-955a8726b159/graph_1
#################################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                       # Mode # Units In # Units Out # Ratio # Time (ms) #
#################################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?n, ?n_code2]
                       # -    # 0         # 1          # 0.00  # 0.02      #
#################################################################################################
# 1  # 2      # 3      # DFETee               # -
                       # -    # 1         # 2          # 2.00  # 0.02      #
#################################################################################################
# 2  # 4      # -      # DFEDistinctColumn    # column=?n
                       # -    # 1         # 1          # 1.00  # 0.20      #
#    #        #        #                     # ordered=false
                       #      #           #            #       #           #
#################################################################################################
# 3  # 5      # -      # DFEHashIndexBuild    # vars=[?n]
                       # -    # 1         # 1          # 1.00  # 0.04      #
#################################################################################################
# 4  # 5      # -      # DFEPipelineJoin      # pattern=Node(?n) with property 'ALL'
 and label '?n_label1' # -    # 1         # 1          # 1.00  # 0.25      #
#    #        #        #                     # patternEstimate=3506
                       #      #           #            #       #           #
#################################################################################################
# 5  # 6      # 7      # DFESync              # -
                       # -    # 2         # 2          # 1.00  # 0.02      #
#################################################################################################
# 6  # 8      # -      # DFEForwardValue      # -
                       # -    # 1         # 1          # 1.00  # 0.01      #
#################################################################################################
# 7  # 8      # -      # DFEForwardValue      # -
                       # -    # 1         # 1          # 1.00  # 0.01      #
#################################################################################################
# 8  # 9      # -      # DFEHashIndexJoin     # -
                       # -    # 2         # 1          # 0.50  # 0.35      #
#################################################################################################
# 9  # -      # -      # DFEDrain             # -
                       # -    # 1         # 0          # 0.00  # 0.02      #
#################################################################################################
```

在顶层，SolutionInjection 出现在其它所有内容之前，其 Units Out 为 1。请注意，它实际上并没有注入任何解。您可以看到下一个运算符 DFESubquery 的 Units In 为 0。

在顶层的 SolutionInjection 之后是 DFESubquery 和 TermResolution 运算符。DFESubquery 封装查询执行计划中正在推送到 DFE 引擎的部分（对于 openCypher 查询，整个查询计划由 DFE 执行）。查询计划中的所有运算符都嵌套在由 DFESubquery 引用的 subQuery1 内部。唯一的例外是 TermResolution，它将内部 ID 实体化为完全序列化的 openCypher 对象。

所有下推到 DFE 引擎的运算符的名称都以 DFE 前缀开头。如上所述，整个 openCypher 查询计划由 DFE 执行，因此，除最后一个 TermResolution 运算符之外的所有运算符都以 DFE 开头。

在 subQuery1 内部，可以有零个或多个 DFEChunkLocalSubQuery 或 DFELoopSubQuery 运算符来封装在内存受限机制中执行的推送执行计划的一部分。这里的 DFEChunkLocalSubQuery 包含一个 SolutionInjection，它用作子查询的输入。要在输出中查找该子查询的表，请在 Arguments 列中搜索为 DFEChunkLocalSubQuery 或 DFELoopSubQuery 运算符指定的 subQuery=*graph URI*。

在 subQuery1 中，ID 为 0 的 DFEPipelineScan 扫描数据库以查找指定的 pattern。该模式在所有标签上扫描属性 code 保存为变量 ?n_code2 的实体（可以通过将 airport 附加到 n:airport 来针对特定标签进行筛选）。inlineFilters 参数显示了针对等于 ATL 的 code 属性的筛选。

接下来，DFEChunkLocalSubQuery 运算符联接包含 DFEPipelineJoin 的子查询的中间结果。这可以确保 ?n 实际上是一个节点，因为之前的 DFEPipelineScan 会扫描任何具有 code 属性的实体。

具有限制的关系查找的 **explain** 输出示例

此查询查找两个类型为 route 的匿名节点之间的关系，最多返回 10。同样，explain 模式为 details，输出格式为默认 ASCII 格式。下面是 explain 输出：

此处，DFEPipelineScan 扫描从匿名节点 ?anon_node7 开始并在另一个匿名节点 ?anon_node21 处结束的边缘，关系类型另存为 ?p_type1。对于 ?p_type1（为 el://route）存在一个筛选条件（其中 el 代表边缘标签），它对应于查询字符串中的 [p:route]。

DFEDrain 收集限制为 10 的输出解，如其 Arguments 列所示。一旦达到限制或生成了所有解（以先发生者为准），DFEDrain 就会终止。

```
curl -d "query=MATCH ()-[p:route]->() RETURN p LIMIT 10" -k https://localhost:8182/
openCypher -d "explain=details"
                                        ~
Query:
MATCH ()-[p:route]->() RETURN p LIMIT 10

################################################################################################
```

```
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
#########################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 10        # 0.00  # 5.00      #
#########################################################################################
# 2  # -      # -      # TermResolution    # vars=[?p]          # id2value_opencypher #
 10       # 10        # 1.00  # 1.00      #
#########################################################################################


subQuery1
#########################################################################################
# ID # Out #1 # Out #2 # Name            # Arguments
              # Mode # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFEPipelineScan # pattern=Edge((?anon_node7)-[?p:?p_type1]->(?
anon_node21)) # -    # 0        # 1000      # 0.00  # 0.66      #
#    #        #        #                 # inlineFilters=[(?p_type1 IN [<el://route>])]
              #      #          #           #       #           #
#    #        #        #                 # patternEstimate=26219
              #      #          #           #       #           #
#########################################################################################
# 1  # 2      # -      # DFEProject      # columns=[?p]
              # -    # 1000     # 1000      # 1.00  # 0.14      #
#########################################################################################
# 2  # -      # -      # DFEDrain        # limit=10
              # -    # 1000     # 0         # 0.00  # 0.11      #
#########################################################################################
```

值表达式函数的 **explain** 输出示例

函数为：

```
MATCH (a) RETURN DISTINCT labels(a)
```

在下面的 explain 输出中，DFEPipelineScan (ID 0)扫描所有节点标签。这对应于 MATCH (a)。

DFEChunkLocalSubquery (ID 1) 汇总每个 ?a 的 ?a 的标签。这对应于 labels(a)。您可以通过 DFEApply 和 DFEReduce 看到这一点。

BindRelation (ID 2) 用于将列通用的 ?__gen_labelsOfa2 重命名为 ?labels(a)。

DFEDistinctRelation (ID 4) 仅检索不同的标签（多个 :airport 节点会给出重复的标签 (a): ["airport"] ）。这对应于 DISTINCT labels(a)。

```
curl -d "query=MATCH (a) RETURN DISTINCT labels(a)" -k https://localhost:8182/
openCypher -d "explain=details"
                                          ~
Query:
MATCH (a) RETURN DISTINCT labels(a)

################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 5         # 0.00  # 81.00     #
################################################################################################
# 2  # -      # -      # TermResolution    # vars=[?labels(a)]  # id2value_opencypher #
 5        # 5         # 1.00  # 1.00      #
################################################################################################


subQuery1
################################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # DFEPipelineScan       # pattern=Node(?a) with property 'ALL'
 and label '?a_label1'                                             # -    # 0
    # 3750      # 0.00  # 26.77     #
#    #        #        #                      # patternEstimate=3506
                                                                 #      #
  #           #       #           #
################################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#8b314f55-2cc7-456a-a48a-c76a0465cfab/graph_1 # -    #
 3750     # 3750      # 1.00  # 0.04      #
################################################################################################
```

```
# 2  # 3      # -      # DFEBindRelation       # inputVars=[?a, ?__gen_labelsOfa2, ?
__gen_labelsOfa2]                                                    # -    # 3750
     # 3750      # 1.00  # 0.08      #
#    #        #        #                      # outputVars=[?a, ?__gen_labelsOfa2, ?
labels(a)]                                                           #      #
    #           #       #           #
#############################################################################################
# 3  # 4      # -      # DFEProject            # columns=[?labels(a)]
                                                                     # -    # 3750
  # 3750      # 1.00  # 0.05      #
#############################################################################################
# 4  # 5      # -      # DFEDistinctRelation   # -
                                                                     # -    # 3750
  # 5         # 0.00  # 2.78      #
#############################################################################################
# 5  # -      # -      # DFEDrain              # -
                                                                     # -    # 5
  # 0         # 0.00  # 0.03      #
#############################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#8b314f55-2cc7-456a-
a48a-c76a0465cfab/graph_1
#############################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments
                     # Mode     # Units In # Units Out # Ratio # Time (ms) #
#############################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?a]
                     # -        # 0         # 3750      # 0.00  # 0.02      #
#############################################################################################
# 1  # 2      # 3      # DFETee                # -
                     # -        # 3750      # 7500      # 2.00  # 0.02      #
#############################################################################################
# 2  # 4      # -      # DFEProject            # columns=[?a]
                     # -        # 3750      # 3750      # 1.00  # 0.04      #
#############################################################################################
# 3  # 17     # -      # DFEOptionalJoin       # -
                     # -        # 7500      # 3750      # 0.50  # 0.44      #
#############################################################################################
# 4  # 5      # -      # DFEDistinctRelation   # -
                     # -        # 3750      # 3750      # 1.00  # 2.23      #
#############################################################################################
# 5  # 6      # -      # DFEDistinctColumn     # column=?a
                     # -        # 3750      # 3750      # 1.00  # 1.50      #
```

```
#    #        #        #                      # ordered=false
                    #          #          #           #       #           #
#####################################################################################################
# 6  # 7      # -      # DFEPipelineJoin      # pattern=Node(?a) with property 'ALL'
 and label '?a_label3' # -        # 3750     # 3750      # 1.00  # 10.58     #
#    #        #        #                      # patternEstimate=3506
                    #          #          #           #       #           #
#####################################################################################################
# 7  # 8      # 9      # DFETee               # -
                    # -        # 3750     # 7500      # 2.00  # 0.02      #
#####################################################################################################
# 8  # 10     # -      # DFEBindRelation      # inputVars=[?a_label3]
                    # -        # 3750     # 3750      # 1.00  # 0.04      #
#    #        #        #                      # outputVars=[?100]
                    #          #          #           #       #           #
#####################################################################################################
# 9  # 11     # -      # DFEBindRelation      # inputVars=[?a, ?a_label3, ?100]
                    # -        # 7500     # 3750      # 0.50  # 0.07      #
#    #        #        #                      # outputVars=[?a, ?a_label3, ?100]
                    #          #          #           #       #           #
#####################################################################################################
# 10 # 9      # -      # DFETermResolution    # column=?100
                    # id2value # 3750     # 3750      # 1.00  # 7.60      #
#####################################################################################################
# 11 # 12     # -      # DFEBindRelation      # inputVars=[?a, ?a_label3, ?100]
                    # -        # 3750     # 3750      # 1.00  # 0.06      #
#    #        #        #                      # outputVars=[?a, ?100, ?a_label3]
                    #          #          #           #       #           #
#####################################################################################################
# 12 # 13     # -      # DFEApply             # functor=nodeLabel(?a_label3)
                    # -        # 3750     # 3750      # 1.00  # 0.55      #
#####################################################################################################
# 13 # 14     # -      # DFEProject           # columns=[?a, ?a_label3_alias4]
                    # -        # 3750     # 3750      # 1.00  # 0.05      #
#####################################################################################################
# 14 # 15     # -      # DFEMergeChunks       # -
                    # -        # 3750     # 3750      # 1.00  # 0.02      #
#####################################################################################################
# 15 # 16     # -      # DFEReduce            # functor=collect(?a_label3_alias4)
                    # -        # 3750     # 3750      # 1.00  # 6.37      #
#    #        #        #                      # segmentationKey=[?a]
                    #          #          #           #       #           #
#####################################################################################################
```

```
# 16 # 3      # -      # DFEMergeChunks       # -
                    # -        # 3750     # 3750      # 1.00  # 0.03      #
##########################################################################################
# 17 # -      # -      # DFEDrain             # -
                    # -        # 3750     # 0         # 0.00  # 0.02      #
##########################################################################################
```

数学值表达式函数的 **explain** 输出示例

在此示例中，RETURN abs(-10) 使用常量 -10 的绝对值执行简单求值。

DFEChunkLocalSubQuery (ID 1) 对存储在变量 ?100 中的静态值 -10 执行解注入。

DFEApply (ID 2) 是对存储在变量 ?100 中的静态值执行绝对值函数 abs() 的运算符。

以下是查询和生成的 explain 输出：

```
curl -d "query=RETURN abs(-10)" -k https://localhost:8182/openCypher  -d
 "explain=details"
                                                    ~
Query:
RETURN abs(-10)

##########################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments             # Mode
  # Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]        # -
  # 0        # 1        # 0.00  # 0         #
##########################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1    # -
  # 0        # 1        # 0.00  # 4.00      #
##########################################################################################
# 2  # -      # -      # TermResolution    # vars=[?_internalVar1] #
 id2value_opencypher # 1        # 1         # 1.00  # 1.00      #
##########################################################################################


subQuery1
##########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
##########################################################################################
```

```
# 0  # 1      # -      # DFESolutionInjection  # outSchema=[]
                                                                    # -    # 0
  # 1         # 0.00  # 0.01      #
#########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#c4cc6148-cce3-4561-93c0-deb91f257356/graph_1 # -    #
 1        # 1         # 1.00  # 0.03      #
#########################################################################################
# 2  # 3      # -      # DFEApply              # functor=abs(?100)
                                                                    # -    # 1
  # 1         # 1.00  # 0.26      #
#########################################################################################
# 3  # 4      # -      # DFEBindRelation       # inputVars=[?_internalVar2, ?
_internalVar2]                                                                  # -
 # 1        # 1         # 1.00  # 0.04      #
#    #        #        #                       # outputVars=[?_internalVar2, ?
_internalVar1]                                                                #
 #          #          #       #           #
#########################################################################################
# 4  # 5      # -      # DFEProject            # columns=[?_internalVar1]
                                                                    # -    # 1
  # 1         # 1.00  # 0.06      #
#########################################################################################
# 5  # -      # -      # DFEDrain              # -
                                                                    # -    # 1
  # 0         # 0.00  # 0.05      #
#########################################################################################

subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#c4cc6148-
cce3-4561-93c0-deb91f257356/graph_1
#########################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments                          #
 Mode # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFESolutionInjection # solutions=[?100 -> [-10^^<LONG>]]   # -
    # 0        # 1         # 0.00  # 0.01      #
#    #        #        #                       # outSchema=[?100]                   #
    #          #          #       #           #
#########################################################################################
# 1  # 3      # -      # DFERelationalJoin    # joinVars=[]                        # -
    # 2        # 1         # 0.50  # 0.18      #
#########################################################################################
# 2  # 1      # -      # DFESolutionInjection # outSchema=[]                       # -
    # 0        # 1         # 0.00  # 0.01      #
```

```
#########################################################################################################
# 3  # -      # -      # DFEDrain            # -                                    # -
    # 1        # 0         # 0.00  # 0.02      #
#########################################################################################################
```

可变长度路径 (VLP) 查询的 **explain** 输出示例

这是处理可变长度路径查询的更复杂的查询计划的示例。为清楚起见，此示例仅显示部分 explain 输出。

在 subQuery1 中，DFEPipelineScan (ID 0) 和 DFEChunkLocalSubQuery (ID 1) 注入 ...graph_1 子查询，负责使用 YPO 代码扫描节点。

在 subQuery1 中，DFEChunkLocalSubQuery (ID 2) 注入 ...graph_2 子查询，负责使用 LAX 代码扫描节点。

在 subQuery1 中，DFEChunkLocalSubQuery (ID 3) 注入 ...graph3 子查询，该子查询包含 DFELoopSubQuery (ID 17)，它转而注入 ...graph5 子查询。此操作负责求解两个节点之间查询字符串中的 -[*2]-> 可变长度模式。

```
curl -d "query=MATCH p=(a {code: 'YPO'})-[*2]->(b{code: 'LAX'}) return p" -k https://
localhost:8182/openCypher -d "explain=details"

                                                ~
Query:
MATCH p=(a {code: 'YPO'})-[*2]->(b{code: 'LAX'}) return p

#########################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments          # Mode                #
 Units In # Units Out # Ratio # Time (ms) #
#########################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]     # -                   #
 0        # 1         # 0.00  # 0         #
#########################################################################################################
# 1  # 2      # -      # DFESubquery       # subQuery=subQuery1 # -                   #
 0        # 0         # 0.00  # 84.00     #
#########################################################################################################
# 2  # -      # -      # TermResolution    # vars=[?p]          # id2value_opencypher #
 0        # 0         # 0.00  # 0         #
#########################################################################################################


subQuery1
```

```
#################################################################################################
# ID # Out #1 # Out #2 # Name                    # Arguments
                                                                                  # Mode # Units
 In # Units Out # Ratio # Time (ms) #
#################################################################################################
# 0  # 1      # -      # DFEPipelineScan        # pattern=Node(?a) with property 'code'
 as ?a_code7 and label 'ALL'                                                       # -    # 0
   # 1         # 0.00  # 0.68      #
#    #        #        #                        # inlineFilters=[(?a_code7 IN
 ["YPO"^^xsd:string])]                                                                  #
  #          #           #       #           #
#    #        #        #                        # patternEstimate=1
                                                                                 #      #
  #           #       #           #
#################################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_1 # -    #
 1        # 1         # 1.00  # 0.03      #
#################################################################################################
# 2  # 3      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_2 # -    #
 1        # 1         # 1.00  # 0.02      #
#################################################################################################
# 3  # 4      # -      # DFEChunkLocalSubQuery # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_3 # -    #
 1        # 0         # 0.00  # 0.04      #
#################################################################################################
# 4  # 5      # -      # DFEBindRelation       # inputVars=[?__gen_path6, ?
anon_rel26, ?b_code8, ?b, ?a_code7, ?a, ?__gen_path6]                                  # -
   # 0        # 0         # 0.00  # 0.10      #
#    #        #        #                        # outputVars=[?__gen_path6, ?
anon_rel26, ?b_code8, ?b, ?a_code7, ?a, ?p]                                            #
  #          #           #       #           #
#################################################################################################
# 5  # 6      # -      # DFEProject            # columns=[?p]
                                                                                  # -    # 0
  # 0         # 0.00  # 0.05      #
#################################################################################################
# 6  # -      # -      # DFEDrain              # -
                                                                                  # -    # 0
  # 0         # 0.00  # 0.02      #
#################################################################################################
```

```
subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_1
#########################################################################################
# ID # Out #1 # Out #2 # Name                 # Arguments
                       # Mode # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # DFESolutionInjection # outSchema=[?a, ?a_code7]
                       # -    # 0        # 1         # 0.00  # 0.01      #
#########################################################################################
# 1  # 2      # 3      # DFETee               # -
                       # -    # 1        # 2         # 2.00  # 0.01      #
#########################################################################################
# 2  # 4      # -      # DFEDistinctColumn    # column=?a
                       # -    # 1        # 1         # 1.00  # 0.25      #
#    #        #        #                      # ordered=false
                       #      #          #           #       #           #
#########################################################################################
# 3  # 5      # -      # DFEHashIndexBuild    # vars=[?a]
                       # -    # 1        # 1         # 1.00  # 0.05      #
#########################################################################################
# 4  # 5      # -      # DFEPipelineJoin      # pattern=Node(?a) with property 'ALL'
 and label '?a_label1' # -    # 1        # 1         # 1.00  # 0.47      #
#    #        #        #                      # patternEstimate=3506
                       #      #          #           #       #           #
#########################################################################################
# 5  # 6      # 7      # DFESync              # -
                       # -    # 2        # 2         # 1.00  # 0.04      #
#########################################################################################
# 6  # 8      # -      # DFEForwardValue      # -
                       # -    # 1        # 1         # 1.00  # 0.01      #
#########################################################################################
# 7  # 8      # -      # DFEForwardValue      # -
                       # -    # 1        # 1         # 1.00  # 0.01      #
#########################################################################################
# 8  # 9      # -      # DFEHashIndexJoin     # -
                       # -    # 2        # 1         # 0.50  # 0.26      #
#########################################################################################
# 9  # -      # -      # DFEDrain             # -
                       # -    # 1        # 0         # 0.00  # 0.02      #
#########################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_2
```

```
##################################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments
                           # Mode # Units In # Units Out # Ratio # Time (ms) #
##################################################################################################
# 0  # 1      # -      # DFEPipelineScan      # pattern=Node(?b) with property 'code'
 as ?b_code8 and label 'ALL' # -    # 0        # 1         # 0.00  # 0.38      #
#    #        #        #                      # inlineFilters=[(?b_code8 IN
 ["LAX"^^xsd:string])]                #      #          #           #       #
  #
#    #        #        #                      # patternEstimate=1
                           #      #          #           #       #           #
##################################################################################################
# 1  # 2      # -      # DFEMergeChunks       # -
                           # -    # 1        # 1         # 1.00  # 0.02      #
##################################################################################################
# 2  # 4      # -      # DFERelationalJoin    # joinVars=[]
                           # -    # 2        # 1         # 0.50  # 0.19      #
##################################################################################################
# 3  # 2      # -      # DFESolutionInjection # outSchema=[?a, ?a_code7]
                           # -    # 0        # 1         # 0.00  # 0         #
##################################################################################################
# 4  # -      # -      # DFEDrain             # -
                           # -    # 1        # 0         # 0.00  # 0.01      #
##################################################################################################


subQuery=http://aws.amazon.com/neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-
bbe3-9e99558eca46/graph_3
##################################################################################################
# ID # Out #1 # Out #2 # Name                   # Arguments
                                                                    # Mode     #
 Units In # Units Out # Ratio # Time (ms) #
##################################################################################################
...
# 17 # 18     # -      # DFELoopSubQuery       # subQuery=http://aws.amazon.com/
neptune/vocab/v01/dfe/past/graph#cc05129f-d07e-4622-bbe3-9e99558eca46/graph_5 # -
  # 1        # 2         # 2.00  # 0.31      #
...
```

Neptune openCypher 中的事务

Amazon Neptune 中的 openCypher 实现使用 Neptune 定义的事务语义。但是，如以下各节所述，Bolt 驱动程序提供的隔离级别对 Bolt 事务语义有一些特定的影响。

只读 Bolt 事务查询

可以处理只读查询的方式多种多样，并可使用不同的事务模型和隔离级别，如下所示：

隐式只读事务查询

以下是只读隐式事务的示例：

```
public void executeReadImplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // read query
  final String READ_QUERY = "MATCH (n) RETURN n limit 10";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
          Config.builder().withEncryption()
                          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
                          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig.builder()
                                             .withFetchSize(1000)
                                             .withDefaultAccessMode(AccessMode.READ)
                                             .build();

  // run the query as access mode read
  driver.session(sessionConfig).readTransaction(new TransactionWork<String>()
    {
      final StringBuilder resultCollector = new StringBuilder();

      @Override
      public String execute(final Transaction tx)
      {
        // execute the query
        Result queryResult = tx.run(READ_QUERY);

        // Read the result
        for (Record record : queryResult.list())
        {
          for (String key : record.keys())
          {
```

```
            resultCollector.append(key)
                           .append(":")
                           .append(record.get(key).asNode().toString());
          }
        }
        return resultCollector.toString();
      }

    }
  );

  // close the driver.
  driver.close();
}
```

由于只读副本仅接受只读查询，因此，无论在会话配置中设置的访问模式如何，所有针对只读副本的查询都将作为读取隐式事务执行。Neptune 在 SNAPSHOT 隔离语义下将读取隐式事务计算为只读查询。

如果失败，默认情况下会重试读取隐式事务。

自动提交只读事务查询

以下是只读自动提交事务的示例：

```
public void executeAutoCommitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // read query
  final String READ_QUERY = "MATCH (n) RETURN n limit 10";

  // Create the session config.
  final SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.READ)
    .build();

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
```

```
          .build());

  // result collector
  final StringBuilder resultCollector = new StringBuilder();

  // create a session
  final Session session = driver.session(sessionConfig);

  // run the query
  final Result queryResult = session.run(READ_QUERY);
  for (final Record record : queryResult.list())
  {
    for (String key : record.keys())
    {
      resultCollector.append(key)
                     .append(":")
                     .append(record.get(key).asNode().toString());
    }
  }

  // close the session
  session.close();

  // close the driver
  driver.close();
}
```

如果在会话配置中将访问模式设置为 READ，则 Neptune 会在 SNAPSHOT 隔离语义下将自动提交事务查询计算为只读查询。请注意，只读副本仅接受只读查询。

如果您未传入会话配置，则默认情况下会使用突变查询隔离来处理自动提交查询，因此传入将访问模式显式设置为 READ 的会话配置非常重要。

如果失败，则不会重试只读自动提交查询。

显式只读事务查询

以下是显式只读事务的示例：

```
public void executeReadExplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";
```

```
// read query
final String READ_QUERY = "MATCH (n) RETURN n limit 10";

// Create the session config.
final SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.READ)
  .build();

// create the driver
final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
  Config.builder()
        .withEncryption()
        .withTrustStrategy(TrustStrategy.trustSystemCertificates())
        .build());

// result collector
final StringBuilder resultCollector = new StringBuilder();

// create a session
final Session session = driver.session(sessionConfig);

// begin transaction
final Transaction tx = session.beginTransaction();

// run the query on transaction
final List<Record> list = tx.run(READ_QUERY).list();

// read the result
for (final Record record : list)
{
  for (String key : record.keys())
  {
    resultCollector
      .append(key)
      .append(":")
      .append(record.get(key).asNode().toString());
  }
}

// commit the transaction and for rollback we can use beginTransaction.rollback();
tx.commit();
```

```
  // close the driver
  driver.close();
}
```

如果在会话配置中将访问模式设置为 READ，则 Neptune 会在 SNAPSHOT 隔离语义下将显式只读事务计算为只读查询。请注意，只读副本仅接受只读查询。

如果您未传入会话配置，则默认情况下会使用突变查询隔离来处理显式只读事务，因此传入将访问模式显式设置为 READ 的会话配置非常重要。

如果失败，默认情况下会重试只读显式查询。

突变 Bolt 事务查询

与只读查询一样，可以通过多种方式处理突变查询，并可使用不同的事务模型和隔离级别，如下所示：

隐式突变事务查询

以下是隐式突变事务的示例：

```
public void executeWriteImplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // create node with label as label and properties.
  final String WRITE_QUERY = "CREATE (n:label {name : 'foo'})";

  // Read the vertex created with label as label.
  final String READ_QUERY = "MATCH (n:label) RETURN n";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.WRITE)
    .build();
```

```
  final StringBuilder resultCollector = new StringBuilder();

  // run the query as access mode write
  driver.session(sessionConfig).writeTransaction(new TransactionWork<String>()
  {
    @Override
    public String execute(final Transaction tx)
    {
      // execute the write query and consume the result.
      tx.run(WRITE_QUERY).consume();

      // read the vertex written in the same transaction
      final List<Record> list = tx.run(READ_QUERY).list();

      // read the result
      for (final Record record : list)
      {
        for (String key : record.keys())
        {
          resultCollector
            .append(key)
            .append(":")
            .append(record.get(key).asNode().toString());
        }
      }
      return resultCollector.toString();
    }
  }); // at the end, the transaction is automatically committed.

  // close the driver.
  driver.close();
}
```

作为突变查询一部分进行的读取是在 READ COMMITTED 隔离下执行的，对 Neptune 突变事务具有通常的保证。

无论您是否专门传入会话配置，该事务始终被视为写入事务。

有关冲突，请参阅使用锁定等待超时解决冲突。

自动提交突变事务查询

突变自动提交查询继承与突变隐式事务相同的行为。

如果未传入会话配置，默认情况下，该事务将被视为写入事务。

如果失败，则不会自动重试突变自动提交查询。

显式突变事务查询

以下是显式突变事务的示例：

```
public void executeWriteExplicitTransaction()
{
  // end point
  final String END_POINT = "(End Point URL)";

  // create node with label as label and properties.
  final String WRITE_QUERY = "CREATE (n:label {name : 'foo'})";

  // Read the vertex created with label as label.
  final String READ_QUERY = "MATCH (n:label) RETURN n";

  // create the driver
  final Driver driver = GraphDatabase.driver(END_POINT, AuthTokens.none(),
    Config.builder()
          .withEncryption()
          .withTrustStrategy(TrustStrategy.trustSystemCertificates())
          .build());

  // create the session config
  SessionConfig sessionConfig = SessionConfig
    .builder()
    .withFetchSize(1000)
    .withDefaultAccessMode(AccessMode.WRITE)
    .build();

  final StringBuilder resultCollector = new StringBuilder();

  final Session session = driver.session(sessionConfig);

  // run the query as access mode write
  final Transaction tx = driver.session(sessionConfig).beginTransaction();

  // execute the write query and consume the result.
  tx.run(WRITE_QUERY).consume();

  // read the result from the previous write query in a same transaction.
```

```
  final List<Record> list = tx.run(READ_QUERY).list();

  // read the result
  for (final Record record : list)
  {
    for (String key : record.keys())
    {
      resultCollector
        .append(key)
        .append(":")
        .append(record.get(key).asNode().toString());
    }
  }

  // commit the transaction and for rollback we can use tx.rollback();
  tx.commit();

  // close the session
  session.close();

  // close the driver.
  driver.close();
}
```

显式突变查询继承与隐式突变事务相同的行为。

如果未传入会话配置，默认情况下，该事务将被视为写入事务。

有关冲突，请参阅使用锁定等待超时解决冲突。

Neptune openCypher 限制

Amazon Neptune 版本的 openCypher 仍然不支持密码查询语言参考版本 9 中规定的所有内容，如OpenCypher 规范合规性中所述。预计未来的版本将解决其中的许多限制。

Neptune openCypher 异常

在 Amazon Neptune 上使用 openCypher 时，可能会出现各种异常。以下是您可能收到的来自 HTTPS 端点或 Bolt 驱动程序的常见异常（Bolt 驱动程序中的所有异常都报告为服务器状态异常）：

HTTP 代码	错误消息	是否可检索？	补救措施	
400	（语法错误，直接从 openCypher 解析器传播）	否	请更正查询语法，然后重试。	
500	`Operation terminated (out of memory)`	是	重新处理查询以添加其它筛选条件来减少所需的内存	
500	操作已终止（超过截止日期）	是	增加数据库集群参数组中的查询超时，或者重试请求。	
500	操作已终止（被用户取消）	是	重试 请求。	
500	数据库重置正在进行中。请在集群可用后重试查询。	是	重置完成后重试。	
500	由于并发操作冲突，操作失败（请重试）。事务目前正在回滚。	是	使用指数回退和重试策略重试。	
400	######操作/特征不受支持的异常	否	不支持指定的操作。	
400	已尝试在只读副本上更新 openCypher	否	将目标端点更改为写入器端点。	

HTTP 代码	错误消息	是否可检索？	补救措施	
400	MalformedQueryException（Neptune 不显示内部解析器状态）	否	请更正查询语法并重试。	
400	无法删除节点，因为它仍然具有关系。要删除此节点，必须先删除其关系。	否	不使用 `MATCH (n) DELETE n`，而是使用 `MATCH(n) DETACH DELETE(n)`	
400	操作无效：正在尝试移除节点的最后一个标签。节点必须具有至少一个标签。	否	Neptune 要求所有节点至少有一个标签，如果创建的节点没有显式标签，则会分配默认标签 `vertex`。更改查询和/或应用程序逻辑，以免删除最后一个标签。可以通过设置新标签然后删除旧标签，更新节点的单例标签。	
500	已违规请求的最大数量，ConfiguredQueueCapacity= {} for connID = {}	是	目前，无论堆栈和协议如何，只能处理 8192 个并发请求。	

HTTP 代码	错误消息	是否可检索？	补救措施	
500	突破最大连接限制。	是	每个实例只允许 1000 个并发 Bolt 连接（对于 HTTP 没有限制）。	
400	应为 [节点、关系或路径之一]，而得到的是文本	否	请检查您传递的参数是否正确，查询语法是否正确，然后重试。	
400	属性值必须是简单的文本。或：需要 Set 属性的映射，但找不到。	否	SET 子句只接受简单文本，不接受复合类型。	
400	找不到已传递的供删除的实体	否	检查数据库中是否存在您尝试删除的实体。	
400	用户无权访问数据库。	否	查看有关正在使用的 IAM 角色的策略。	
400	没有作为请求的一部分传递的令牌	否	在启用 IAM 的集群上，必须将经过正确签名的令牌作为查询请求的一部分传递。	
400	错误消息已传播。	否	使用请求编号与 Support 联系 AWS 。	

HTTP 代码	错误消息	是否可检索？	补救措施	
500	操作已终止（内部错误）	是	使用请求编号与 Support 联系 AWS 。	

使用 SPARQL 访问 Neptune 图形

SPARQL 是一种用于资源描述框架 (RDF) 的查询语言，这是一种专为 Web 设计的图形数据格式。Amazon Neptune 与 SPARQL 1.1 兼容。这表示您可以连接到 Neptune 数据库实例并使用 SPARQL 1.1 查询语言规范中所述的查询语言查询图形。

SPARQL 中的查询包含一个用于指定要返回的变量的 `SELECT` 子句和一个用于指定要在图形中匹配的数据的 `WHERE` 子句。如果您不熟悉 SPARQL 查询，请参阅 SPARQL 1.1 查询语言中的编写简单查询。

> **Important**
>
> 要加载数据，对于较小的数据集 `SPARQL UPDATE INSERT` 可能效果很好，但如果您需要从文件加载大量数据，请参阅使用 Amazon Neptune 批量加载程序收集数据。

有关 Neptune 的 SPARQL 实施细节的更多信息，请参阅SPARQL 标准合规性。

在开始之前，您必须具有以下内容：

- 一个 Neptune 数据库实例。有关创建 Neptune 数据库实例的信息，请参阅创建新的 Neptune 数据库集群。
- 与 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例。

主题

- 使用 RDF4J 控制台连接到 Neptune 数据库实例
- 使用 RDF4J Workbench 连接到 Neptune 数据库实例
- 使用 Java 连接到 Neptune 数据库实例
- SPARQL HTTP API
- SPARQL 查询提示

- SPARQL DESCRIBE 相对于默认图形的行为
- SPARQL 查询状态 API
- SPARQL 查询取消
- 在 Amazon Neptune 中使用 SPARQL 1.1 图形存储 HTTP 协议 (GSP)
- 使用 SPARQL explain 分析 Neptune 查询执行
- Neptune 中使用 SERVICE 扩展的 SPARQL 联合查询

使用 RDF4J 控制台连接到 Neptune 数据库实例

RDF4J 控制台允许您在 REPL（read-eval-print 循环）环境中尝试资源描述框架 (RDF) 图形和查询。

您可以将远程图形数据库添加为存储库并从 RDF4J 控制台查询它。此部分将指导您完成 RDF4J 控制台的配置以远程连接到 Neptune 数据库实例。

使用 RDF4J 控制台连接到 Neptune

1. 从 RDF4J 网站上的下载页面下载 RDF4J SDK。
2. 解压缩 RDF4J SDK zip 文件。
3. 在终端中，导航到 RDF4J SDK 目录，然后输入以下命令来运行 RDF4J 控制台：

   ```
   bin/console.sh
   ```

   您应该可以看到类似于如下所示的输出内容：

   ```
   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 3.6.1

   3.6.1
   Type 'help' for help.
   >
   ```

   您现在位于 > 提示符处。这是面向 RDF4J 控制台的一般提示符。您使用此提示符设置存储库及其他操作。存储库具有其自己的用于运行查询的提示符。

4. 在 > 提示符处，输入以下命令为您的 Neptune 数据库实例创建 SPARQL 存储库：

```
create sparql
```

5. RDF4J 控制台提示您提供要连接到 SPARQL 终端节点所需的变量的值。

```
Please specify values for the following variables:
```

指定以下值：

变量名称	值
SPARQL 查询终端节点	`https://your-neptune-endpoint :port/sparql`
SPARQL 更新终端节点	`https://your-neptune-endpoint :port/sparql`
本地存储库 ID [endpoint@localhost]	`neptune`
存储库标题 [SPARQL endpoint repository @localhost]	`Neptune DB instance`

有关查找 Neptune 数据库实例的地址的信息，请参阅连接到 Amazon Neptune 端点部分。

如果此操作成功，您会看到以下消息：

```
Repository created
```

6. 在 > 提示符处，输入以下命令以连接到 Neptune 数据库实例：

```
open neptune
```

如果此操作成功，您会看到以下消息：

```
Opened repository 'neptune'
```

您现在位于 neptune> 提示符处。在此提示符处，您可以针对 Neptune 图形运行查询。

> **Note**
>
> 您现在已添加存储库，在下次运行 bin/console.sh 时，您可以立即运行 open neptune 命令来连接到 Neptune 数据库实例。

7. 在 neptune> 提示符处，输入以下命令以运行 SPARQL 查询，该查询通过使用限制为 10 的 ? s ?p ?o 查询最多返回图形中的 10 个三元组（主-谓-宾）。要查询其他内容，请将 sparql 命令后面的文本替换为其他 SPARQL 查询。

```
sparql select ?s ?p ?o where {?s ?p ?o} limit 10
```

使用 RDF4J Workbench 连接到 Neptune 数据库实例

此部分将指导您完成使用 RDF4J Workbench 和 RDF4J Server 连接到 Amazon Neptune 数据库实例。RDF4J Server 是必需的，因为它充当着 Neptune SPARQL HTTP REST 端点与 RDF4J Workbench 之间的代理。

RDF4J Workbench 提供了一个用于试验图形 (包括加载本地文件) 的简单接口。有关信息，请参阅 RDF4J 文档中的添加部分。

先决条件

开始之前，请执行以下操作：

- 安装 Java 1.8 或更高版本。
- 安装 RDF4J Server 和 RDF4J Workbench。有关信息，请参阅安装 RDF4J Server 和 RDF4J Workbench。

使用 RDF4J Workbench 连接到 Neptune

1. 在 Web 浏览器中，导航到在其中部署 RDF4J Workbench Web 应用程序的 URL。例如，如果您使用的是 Apache Tomcat，该 URL 为：https://*ec2_hostname*:8080/rdf4j-workbench/。
2. 如果系统要求您 Connect to RDF4J Server (连接到 RDF4J 服务器)，请验证RDF4J Server (RDF4J 服务器) 是否已安装、正在运行并且服务器 URL 是否正确。然后，继续执行下一步。

3. 在左侧窗格中，选择新建存储库。

   在 New repository (新建存储库) 中：

   - 在类型下拉列表中，选择 SPARQL 终端节点代理。
   - 对于 ID，键入 neptune。
   - 对于标题，键入 Neptune 数据库实例。

   选择下一步。

4. 在 New repository (新建存储库) 中：

   - 对于 SPARQL query endpoint URL (SPARQL 查询终端节点 URL)，键入 `https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`。
   - 对于 SPARQL update endpoint URL (SPARQL 更新终端节点 URL)，键入 `https://`*`your-neptune-endpoint`*`:`*`port`*`/sparql`。

   有关查找 Neptune 数据库实例的地址的信息，请参阅连接到 Amazon Neptune 端点部分。

   选择创建。

5. neptune 存储库现在将显示在存储库列表中。可能要在几分钟后您才能使用新存储库。
6. 在表的 ID 列中，选择 neptune 链接。
7. 在左侧窗格中，选择查询。

   > **Note**
   >
   > 如果浏览下的菜单项已禁用，您可能需要重新连接到 RDF4J Server 并再次选择 neptune 存储库。
   > 您可以通过使用右上角中的 [更改] 链接执行此操作。

8. 在查询字段中，键入以下 SPARQL 查询，然后选择执行。

```
select ?s ?p ?o where {?s ?p ?o} limit 10
```

上述示例使用限制为 10 的 `?s ?p ?o` 查询最多返回图形中的 10 个三元组 (主-谓-宾)。

使用 Java 连接到 Neptune 数据库实例

此部分将指导您完成运行完整的 Java 示例，该示例连接到 Amazon Neptune 数据库实例并执行 SPARQL 查询。

从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

使用 Java 连接到 Neptune

1. 在您的 EC2 实例上安装 Apache Maven。首先，输入以下命令以添加具有 Maven 程序包的存储库：

   ```
   sudo wget https://repos.fedorapeople.org/repos/dchen/apache-maven/epel-apache-maven.repo -O /etc/yum.repos.d/epel-apache-maven.repo
   ```

   输入以下命令以设置该程序包的版本号：

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

   然后，可以使用 yum 安装 Maven：

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

2. 此示例仅使用 Java 8 进行测试。输入以下命令以在 EC2 实例上安装 Java 8：

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

3. 输入以下命令以在 EC2 实例上将 Java 8 设置为默认运行时系统：

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

   在系统提示时，输入 Java 8 的版本号。

4. 输入以下命令以在 EC2 实例上将 Java 8 设置为默认编译器：

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

   在系统提示时，输入 Java 8 的版本号。

5. 在新目录中，创建 `pom.xml` 文件，然后在文本编辑器中打开它。

6. 将以下内容复制到 pom.xml 文件中并保存它（您通常可以将版本号调整为最新的稳定版本）：

```
<project xmlns="https://maven.apache.org/POM/4.0.0" xmlns:xsi="https://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://maven.apache.org/POM/4.0.0 https://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>https://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>org.eclipse.rdf4j</groupId>
      <artifactId>rdf4j-runtime</artifactId>
      <version>3.6</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**
>
> 如果您要修改现有的 Maven 项目，则所需依赖项将在上述代码中突出显示。

7. 要为示例源代码创建子目录 (`src/main/java/com/amazonaws/`)，请在命令行处输入以下命令：

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

8. 在 `src/main/java/com/amazonaws/` 目录中，创建名为 `App.java` 的文件，然后在文本编辑器中打开它。

9. 将以下内容复制到 `App.java` 文件中。将 *`your-neptune-endpoint`* 替换为 Neptune 数据库实例的地址。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点部分。

```
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 = "https://your-neptune-endpoint:port/sparql";
        Repository repo = new SPARQLRepository(sparqlEndpoint);
        repo.initialize();

        try (RepositoryConnection conn = repo.getConnection()) {
           String queryString = "SELECT ?s ?p ?o WHERE { ?s ?p ?o } limit 10";

           TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
 queryString);

           try (TupleQueryResult result = tupleQuery.evaluate()) {
              while (result.hasNext()) {  // iterate over the result
                   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. 使用以下 Maven 命令编译并运行示例：

```
mvn compile exec:java
```

上述示例使用限制为 10 的 ?s ?p ?o 查询最多返回图形中的 10 个三元组 (主-谓-宾)。要查询其他内容，请将此查询替换为其他 SPARQL 查询。

示例中结果的迭代输出返回的每个变量的值。Value 对象将转换为 String，然后进行输出。如果更改查询的 SELECT 部分，您必须修改代码。

SPARQL HTTP API

在以下终端节点处接受 SPARQL HTTP 请求：`https://your-neptune-endpoint:port/sparql`

有关使用 SPARQL 连接到 Amazon Neptune 的更多信息，请参阅使用 SPARQL 访问 Neptune 图形。

有关 SPARQL 协议和查询语言的更多信息，请参阅 SPARQL 1.1 协议和 SPARQL 1.1 查询语言规范。

以下主题提供有关 SPARQL RDF 序列化格式的信息，以及如何将 SPARQL HTTP API 与 Neptune 结合使用。

目录

使用 HTTP REST 端点连接到 Neptune 数据库实例

Amazon Neptune 为 SPARQL 查询提供 HTTP 端点。REST 接口与 SPARQL 版本 1.1 兼容。

> **Important**
>
> 版本：1.0.4.0（2020 年 10 月 12 日）规定所有与 Amazon Neptune 的连接都必须使用 TLS 1.2 和 HTTPS。无法再使用不安全的 HTTP 或使用 TLS 版本低于 1.2 的 HTTPS 连接到 Neptune。

以下说明将带您演练使用 curl 命令、通过 HTTPS 的连接以及使用 HTTP 语法来连接到 SPARQL 终端节点。从与您的 Neptune 数据库实例位于同一虚拟私有云 (VPC) 中的 Amazon EC2 实例中按照这些说明操作。

针对 Neptune 数据库实例的 SPARQL 查询的 HTTP 端点为 `https://your-neptune-endpoint:port/sparql`。

> **Note**
>
> 有关查找 Neptune 数据库实例的主机名的信息，请参阅连接到 Amazon Neptune 端点部分。

使用 HTTP POST 的查询

以下示例使用 curl 通过 HTTP POST 提交 SPARQL **QUERY**。

```
curl -X POST --data-binary 'query=select ?s ?p ?o where {?s ?p ?o} limit 10'
 https://your-neptune-endpoint:port/sparql
```

上述示例使用限制为 10 的 `?s ?p ?o` 查询最多返回图形中的 10 个三元组 (主-谓-宾)。要查询其他内容，请将其替换为其他 SPARQL 查询。

> **Note**
>
> 对于 SELECT 和 ASK 查询，响应的默认 MIME 媒体类型为 `application/sparql-results+json`。
> 对于 CONSTRUCT 和 DESCRIBE 查询，响应的默认 MIME 类型为 `application/n-quads`。
> 有关 Neptune 用于序列化的媒体类型列表，请参阅Neptune SPARQL 使用的 RDF 序列化格式。

使用 HTTP POST 的更新

以下示例使用 curl 通过 HTTP POST 提交 SPARQL **UPDATE**。

```
curl -X POST --data-binary 'update=INSERT DATA { <https://test.com/s> <https://
test.com/p> <https://test.com/o> . }' https://your-neptune-endpoint:port/sparql
```

上述示例将以下三元组插入 SPARQL 默认图形中：`<https://test.com/s> <https://test.com/p> <https://test.com/o>`

用于多部分 SPARQL 响应的可选 HTTP 尾随标头

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.3.0 开始推出。

对 SPARQL 查询和更新的 HTTP 响应通常以多个部分或块的形式返回。可能很难诊断在查询或更新开始发送这些块后发生的故障，尤其是因为第一个块到达时的 HTTP 状态代码为 `200`。

除非您明确请求尾随标头，否则 Neptune 只能通过在消息正文（通常已损坏）中附加错误消息来报告此类故障。

为了更轻松地检测和诊断此类问题，您可以在请求中包含传输编码 (TE) 后缀标头 (`te: trailers`)（例如，请参阅有关 TE 请求标头的 MDN 页面）。这样做会导致 Neptune 在响应块的尾随标头中加入两个新的标头字段：

- `X-Neptune-Status` – 包含响应代码后跟一个短名称。例如，如果成功，则尾随标头将是：`X-Neptune-Status: 200 OK`。如果出现故障，响应代码将是 Neptune 引擎错误代码，例如 `X-Neptune-Status: 500 TimeLimitExceededException`。
- `X-Neptune-Detail` – 对于成功的请求，为空。如果出现错误，则它包含 JSON 错误消息。由于 HTTP 标头值中只允许使用 ASCII 字符，因此 JSON 字符串是经过 URL 编码的。错误消息还会附加到响应消息正文中。

Neptune 中由 SPARQL 使用的 RDF 媒体类型

资源描述框架 (RDF) 数据可以通过多种不同方式序列化，SPARQL 可以使用或输出其中的大部分方式：

Neptune SPARQL 使用的 RDF 序列化格式

- RDF/XML – RDF 的 XML 序列化，在 RDF 1.1 XML 语法中定义。媒体类型：`application/rdf+xml`。典型文件扩展名：`.rdf`。
- N-Triples – 基于行的纯文本格式，用于编码 RDF 图形，在 RDF 1.1 N-Triples 中定义。媒体类型：`application/n-triples`、`text/turtle` 或 `text/plain`。典型文件扩展名：`.nt`。
- N-Quads – 基于行的纯文本格式，用于编码 RDF 图形，在 RDF 1.1 N-Quads 中定义。它是 N-Triples 的扩展。媒体类型：`application/n-quads` 或 `text/x-nquads`（使用 7 位 US-ASCII 编码时）。典型文件扩展名：`.nq`。

- Turtle – 在 RDF 1.1 Turtle 中定义的 RDF 文本语法，允许 RDF 图形完全使用紧凑自然的文本形式编写，并为常用使用模式和数据类型使用缩写。Turtle 提供了与 N-Triples 格式以及 SPARQL 的三元组模式语法一定水平的兼容性。媒体类型：`text/turtle`。典型文件扩展名：`.ttl`。
- TriG – 在 RDF 1.1 TriG 中定义的 RDF 文本语法，允许 RDF 图形完全使用紧凑自然的文本形式编写，并为常用使用模式和数据类型使用缩写。TriG 是 Turtle 格式的扩展。媒体类型：`application/trig`。典型文件扩展名：`.trig`。
- N3 (Notation3) – 在 Notation3 (N3)：易于阅读的 RDF 语法中定义的一种断言和逻辑语言。N3 通过添加公式（作为图形本身的文字）、变量、逻辑含义和功能谓词，扩展了 RDF 数据模型，并且提供了对 RDF/XML 的文本语法替代。媒体类型：`text/n3`。典型文件扩展名：`.n3`。
- JSON-LD - 在 JSON-LD 1.0 中定义的数据序列化和消息传送格式。媒体类型：`application/ld+json`。典型文件扩展名：`.jsonld`。
- TriX – XML 格式的 RDF 序列化，在 TriX：XML 格式的 RDF Triples 中定义。媒体类型：`application/trix`。典型文件扩展名：`.trix`。
- SPARQL JSON 结果 - RDF 序列化，使用 SPARQL 1.1 查询结果 JSON 格式。媒体类型：`application/sparql-results+json`。典型文件扩展名：`.srj`。
- RDF4J 二进制格式 - 用于编码 RDF 数据的二进制格式，在 RDF4J 二进制 RDF 格式中记录。媒体类型：`application/x-binary-rdf`。

Neptune SPARQL 使用的 SPARQL 结果序列化格式

- SPARQL XML 结果 - SPARQL 查询语言提供的变量绑定和布尔值结果格式的 XML 格式，在 SPARQL 查询结果 XML 格式（第二版）中定义。媒体类型：`application/sparql-results+xml`。典型文件扩展名：`.srx`。
- SPARQL CSV 和 TSV 结果 - 使用逗号分隔值和制表符分隔值来表示 SELECT 查询的 SPARQL 查询结果，在 SPARQL 1.1 查询结果 CSV 和 TSV 格式中定义。媒体类型：对于逗号分隔值为 `text/csv`，对于制表符分隔值为 `text/tab-separated-values`。典型文件扩展名：对于逗号分隔值为 `.csv`，对于制表符分隔值为 `.tsv`。
- 二进制结果表 - 对 SPARQL 查询的输出进行编码的二进制格式。媒体类型：`application/x-binary-rdf-results-table`。
- SPARQL JSON 结果 - RDF 序列化，使用 SPARQL 1.1 查询结果 JSON 格式。媒体类型：`application/sparql-results+json`。

Neptune 可用于导入 RDF 数据的媒体类型

Neptune 批量加载程序支持的媒体类型

- N-Triples
- N-Quads
- RDF/XML
- Turtle

SPARQL UPDATE LOAD 可以导入的媒体类型

- N-Triples
- N-Quads
- RDF/XML
- Turtle
- TriG
- N3
- JSON-LD

Neptune 可用于导出查询结果的媒体类型

要指定 SPARQL 查询响应的输出格式，请随查询请求发送 "Accept: *media-type*" 标头。例如：

```
curl -H "Accept: application/nquads" ...
```

SPARQL SELECT 可从 Neptune 输出的 RDF 媒体类型

- SPARQL JSON 结果（这是默认值）
- SPARQL XML 结果
- 二进制结果表（媒体类型 application/x-binary-rdf-results-table）
- 逗号分隔值 (CSV)
- 制表符分隔值 (TSV)

SPARQL ASK 可从 Neptune 输出的 RDF 媒体类型

- SPARQL JSON 结果（这是默认值）
- SPARQL XML 结果
- 布尔值（媒体类型：`text/boolean`，表示“true”或“false”）

SPARQL CONSTRUCT 可从 Neptune 输出的 RDF 媒体类型

- N-Quads（这是默认值）
- RDF/XML
- JSON-LD
- N-Triples
- Turtle
- N3
- TriX
- TriG
- SPARQL JSON 结果
- RDF4J 二进制 RDF 格式

SPARQL DESCRIBE 可从 Neptune 输出的 RDF 媒体类型

- N-Quads（这是默认值）
- RDF/XML
- JSON-LD
- N-Triples
- Turtle
- N3
- TriX
- TriG
- SPARQL JSON 结果
- RDF4J 二进制 RDF 格式

使用 SPARQL UPDATE LOAD 将数据导入到 Neptune

SPARQL UPDATE LOAD 命令的语法在 SPARQL 1.1 更新建议中指定：

```
LOAD SILENT (URL of data to be loaded) INTO GRAPH (named graph into which to load the
 data)
```

- **SILENT** – (可选) 即使在处理过程中出现错误，也使操作返回成功。

  当单个事务包含多个语句 (例如 "LOAD ...; LOAD ...; UNLOAD ...; LOAD ...;") 并且即使某些远程数据无法处理也希望事务完成时，这可能很有用。

- ####### URL - (必需) 指定包含要加载到图形中的数据的远程数据文件。

  远程文件必须具有以下扩展名之一：

  - .nt 表示 NTriples。
  - .nq 表示 NQuads。
  - .trig 表示 Trig。
  - .rdf 表示 RDF/XML。
  - .ttl 表示 Turtle。
  - .n3 表示 N3。
  - .jsonld 表示 JSON-LD。

- **INTO GRAPH**###############– (可选) 指定应将数据加载到的图形。

  Neptune 将每个三元组与一个命名图形相关联。您可以使用后备命名图形 URI http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph 来指定默认的命名图形，如下所示：

  ```
  INTO GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
  ```

> **Note**
>
> 当您需要加载大量数据时，我们建议您使用 Neptune 批量加载程序，而不是 UPDATE LOAD。有关批量加载程序的更多信息，请参阅使用 Amazon Neptune 批量加载程序收集数据。

您可以使用 SPARQL UPDATE LOAD 直接从 Amazon S3 加载数据，或者从通过自行托管 Web 服务器获取的文件加载数据。要加载的资源必须与 Neptune 服务器位于相同区域，并且资源的端点必须在 VPC 中得到允许。有关创建 Amazon S3 端点的信息，请参阅创建 Amazon S3 VPC 端点。

所有 SPARQL UPDATE LOAD URI 都必须以 `https://` 开头。这包括 Amazon S3 URL。

与 Neptune 批量加载程序相反，对 SPARQL UPDATE LOAD 的调用是完全事务性的。

使用 SPARQL UPDATE LOAD 直接将文件从 Amazon S3 加载到 Neptune

由于 Neptune 不允许您在使用 SPARQL UPDATE LOAD 时将 IAM 角色传递给 Amazon S3，因此所讨论的 Amazon S3 桶必须是公有的，或者您必须在 LOAD 查询中使用预签名的 Amazon S3 URL。

要为 Amazon S3 文件生成预签名 URL，您可以使用如下 AWS CLI 命令：

```
aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of
 data to load)
```

然后，您就可以在 LOAD 命令中使用生成的预签名 URL：

```
curl https://(a Neptune endpoint URL):8182/sparql \
  --data-urlencode 'update=load (pre-signed URL of the remote Amazon S3 file of data to
 be loaded) \
                           into graph (named graph)'
```

有关更多信息，请参阅身份验证请求：使用查询参数。Boto3 文档中介绍了如何使用 Python 脚本生成预签名 URL。

此外，要加载的文件的内容类型必须正确设置。

1. 将文件上传到 Amazon S3 时，使用 `-metadata` 参数设置文件的内容类型，如下所示：

   ```
   aws s3 cp test.nt s3://bucket-name/my-plain-text-input/test.nt --metadata Content-
   Type=text/plain
   aws s3 cp test.rdf s3://bucket-name/my-rdf-input/test.rdf --metadata Content-
   Type=application/rdf+xml
   ```

2. 确认媒体类型信息实际存在。运行：

   ```
   curl -v bucket-name/folder-name
   ```

   此命令的输出应显示您在上传文件时设置的媒体类型信息。

3. 然后，您可以使用 SPARQL UPDATE LOAD 命令将这些文件导入到 Neptune：

```
curl https://your-neptune-endpoint:port/sparql \
  -d "update=LOAD <https://s3.amazonaws.com/bucket-name/my-rdf-input/test.rdf>"
```

以上步骤仅适用于公有 Amazon S3 桶或您使用 LOAD 查询中的预签名的 Amazon S3 URL 访问的桶。

您还可以设置 Web 代理服务器，以便从私有 Amazon S3 桶中加载，如下所示：

使用 Web 服务器，通过 SPARQL UPDATE LOAD 将文件加载到 Neptune 中

1. 在运行于 VPC 中且托管了 Neptune 和要加载的文件的计算机上安装 Web 服务器。例如，使用 Amazon Linux，您可以如下所示安装 Apache：

```
sudo yum install httpd mod_ssl
sudo /usr/sbin/apachectl start
```

2. 定义您将要加载的 RDF 文件内容的 MIME 类型。SPARQL 使用 Web 服务器发送的 Content-type 标头确定内容的输入格式，因此您必须为 Web 服务器定义相关的 MIME 类型。

   例如，假设您使用以下文件扩展名来标识文件格式：

   - .nt 表示 NTriples。
   - .nq 表示 NQuads。
   - .trig 表示 Trig。
   - .rdf 表示 RDF/XML。
   - .ttl 表示 Turtle。
   - .n3 表示 N3。
   - .jsonld 表示 JSON-LD。

   如果您使用 Apache 2 作为 Web 服务器，您将编辑文件 /etc/mime.types 并添加以下类型：

```
text/plain nt
application/n-quads nq
application/trig trig
application/rdf+xml rdf
application/x-turtle ttl
text/rdf+n3 n3
```

```
application/ld+json jsonld
```

3. 确认 MIME 类型映射可以正常使用。在您启动并运行 Web 服务器并托管了所选格式的 RDF 文件之后，您可以通过从本地主机向 Web 服务器发送请求来测试配置。

   例如，您可以如下所示发送请求：

   ```
   curl -v http://localhost:80/test.rdf
   ```

   然后，在 `curl` 的详细输出中您应看到如下所示的行：

   ```
   Content-Type: application/rdf+xml
   ```

   这指示已成功定义内容类型映射。

4. 现在，您已准备好使用 SPARQL UPDATE 命令加载数据：

   ```
   curl https://your-neptune-endpoint:port/sparql \
       -d "update=LOAD <http://web_server_private_ip:80/test.rdf>"
   ```

> **Note**
>
> 当加载的源文件很大时，在 Web 服务器上使用 SPARQL UPDATE LOAD 可能会触发超时。Neptune 在文件数据流式传入时进行处理，对于大文件，其用时可能会超过在服务器上配置的超时。这反过来可能会导致服务器关闭连接，使得 Neptune 在流中遇到意外的 EOF 时出现以下错误消息：
>
> ```
> {
>   "detailedMessage":"Invalid syntax in the specified file",
>   "code":"InvalidParameterException"
> }
> ```
>
> 如果您收到此消息，但不认为自己的源文件包含无效语法，请尝试增加 Web 服务器上的超时设置。您还可以通过在服务器上启用调试日志并查看超时来诊断问题。

使用 SPARQL UPDATE UNLOAD 从 Neptune 中删除数据

Neptune 还提供了自定义 SPARQL 操作 UNLOAD，用于删除远程源中指定的数据。UNLOAD 可以视为 LOAD 操作的反操作。它的语法如下：

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.4.1 开始推出。

```
UNLOAD SILENT (URL of the remote data to be unloaded) FROM GRAPH (named graph from
 which to remove the data)
```

- **SILENT** – （可选）即使在处理数据时出错，也使操作返回成功。

  当单个事务包含多个语句（例如 "LOAD ...; LOAD ...; UNLOAD ...; LOAD ...;"）并且即使某些远程数据无法处理也希望事务完成时，这可能很有用。
- ######### URL - （必需）指定包含要从图形中卸载的数据的远程数据文件。

  远程文件必须具有以下扩展名之一（这些扩展名与 UPDATE-LOAD 支持的格式相同）：
  - .nt 表示 NTriples。
  - .nq 表示 NQuads。
  - .trig 表示 Trig。
  - .rdf 表示 RDF/XML。
  - .ttl 表示 Turtle。
  - .n3 表示 N3。
  - .jsonld 表示 JSON-LD。

  UNLOAD 操作将从您的数据库集群中删除该文件包含的所有数据。

  要卸载的数据的 URL 中必须包含任何 Amazon S3 身份验证。您可以预签署 Amazon S3 文件，然后使用生成的 URL 来安全地访问该文件。例如：

  ```
  aws s3 presign --expires-in (number of seconds) s3://(bucket name)/(path to file of
   data to unload)
  ```

  然后：

```
curl https://(a Neptune endpoint URL):8182/sparql \
  --data-urlencode 'update=unload (pre-signed URL of the remote Amazon S3 data to be
 unloaded) \
                           from graph (named graph)'
```

有关更多信息，请参阅[身份验证请求：使用查询参数]。

- **FROM GRAPH** *##############*–（可选）指定应从中卸载远程数据的命名图形。

  Neptune 将每个三元组与一个命名图形相关联。您可以使用后备命名图形 URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` 来指定默认的命名图形，如下所示：

  ```
  FROM GRAPH <http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph>
  ```

按照 LOAD 与 INSERT DATA { *(inline data)* } 对应的相同方式，UNLOAD 对应于 DELETE DATA { *(inline data)* }。与 DELETE DATA 相似，UNLOAD 不适用于包含空白节点的数据。

例如，如果本地 Web 服务器提供的文件命名为 `data.nt`，其中包含以下 2 个三元组：

```
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/
resource#b> .
<http://example.org/resource#a> <http://example.org/resource#p> <http://example.org/
resource#c> .
```

以下 UNLOAD 命令将从命名图形 `<http://example.org/graph1>` 中删除这两个三元组：

```
UNLOAD <http://localhost:80/data.nt> FROM GRAPH <http://example.org/graph1>
```

这与使用以下 DELETE DATA 命令的效果相同：

```
DELETE DATA {
  GRAPH <http://example.org/graph1> {
    <http://example.org/resource#a> <http://example.org/resource#p> <http://
example.org/resource#b> .
    <http://example.org/resource#a> <http://example.org/resource#p> <http://
example.org/resource#c> .
  }
}
```

UNLOAD 命令引发的异常

- **InvalidParameterException** – 数据中有空白节点。HTTP 状态：400 错误请求。

  消息： `Blank nodes are not allowed for UNLOAD`

- **InvalidParameterException** – 数据中存在语法错误。HTTP 状态：400 错误请求。

  消息：`Invalid syntax in the specified file.`

- **UnloadUrlAccessDeniedException** – 访问被拒绝。HTTP 状态：400 错误请求。

  消息：`Update failure: Endpoint (Neptune endpoint) reported access denied error. Please verify access.`

- **BadRequestException** – 无法检索远程数据。HTTP 状态：400 错误请求。

  消息：（取决于 HTTP 响应）。

SPARQL 查询提示

您可使用查询提示为 Amazon Neptune 中的特定 SPARQL 查询指定优化和计算策略。

查询提示是使用 SPARQL 查询中嵌入的附加三元组模式表示的，包含以下部分：

```
scope hint value
```

- scope – 确定查询提示应用于的查询部分，例如，查询中的特定组或完整查询。
- hint – 确定要应用的提示的类型。
- value – 确定要考虑的系统方面的行为。

查询提示和范围在 Amazon Neptune 命名空间 `http://aws.amazon.com/neptune/vocab/v01/QueryHints#` 中显示为预定义的术语。本部分中的示例通过在查询中定义和包含 `hint` 前缀来包括该命名空间：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

例如，以下示例说明了如何在 SELECT 查询中包含 joinOrder 提示：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ... {
 hint:Query hint:joinOrder "Ordered" .
 ...
}
```

上述查询指示 Neptune 引擎按指定 顺序计算查询中的联接并禁用任何自动重新排序。

使用查询提示时注意以下事项：

- 您可以在单个查询中组合不同的查询提示。例如，您可以使用 bottomUp 查询提示来注释子查询以进行自下而上的计算，并使用 joinOrder 查询提示来修复子查询中的联接顺序。
- 您可以在不同的非重叠范围内多次使用相同的查询提示。
- 查询提示仅仅作为提示。尽管查询引擎通常会考虑给定的查询提示，但它仍然可能会忽略它们。
- 查询提示将保留语义。添加查询提示不会更改查询的输出（未提供排序保证时可能的结果顺序除外，也就是说，当未使用 ORDER BY 显式强制实施结果顺序时）。

以下各节提供了有关 Neptune 中的可用查询提示及其用法的更多信息。

主题

- Neptune 中的 SPARQL 查询提示范围
- joinOrder SPARQL 查询提示
- evaluationStrategy SPARQL 查询提示
- queryTimeout SPARQL 查询提示
- rangeSafe SPARQL 查询提示
- queryId SPARQL 查询提示
- useDFE SPARQL 查询提示
- 与 DESCRIBE 结合使用的 SPARQL 查询提示

Neptune 中的 SPARQL 查询提示范围

下表显示 Amazon Neptune 中 SPARQL 查询提示的可用范围、关联提示和描述。这些条目中的 `hint` 前缀表示提示的 Neptune 命名空间：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
```

范围	支持的提示	描述
`hint:Query`	joinOrder	查询提示应用于整个查询。
`hint:Query`	queryTimeout	超时值适用于整个查询。
`hint:Query`	rangeSafe	对整个查询禁用了类型提升。
`hint:Query`	queryId	查询 ID 值适用于整个查询。
`hint:Query`	useDFE	对整个查询启用（或禁用）DFE。
`hint:Group`	joinOrder	查询提示应用于指定组中的顶级元素，但不应用于嵌套元素（如子查询）或父元素。
`hint:SubQuery`	evaluationStrategy	指定提示并将它应用于嵌套 SELECT 子查询。单独计算子查询，而不考虑在子查询之前计算的解。

`joinOrder` SPARQL 查询提示

在提交 SPARQL 查询时，Amazon Neptune 查询引擎将调查查询的结构。它将对查询的各个部分重新排序，并尝试最大程度地减少计算所需的工作量和查询响应时间。

例如，通常不会按给定顺序计算一系列连接的三元组模式。使用试探法和统计数据（例如，各个模式的选择性以及它们通过共享变量连接的方式）来对查询重新排序。此外，如果您的查询包含更复杂的模式（例如，子查询、FILTER 或复杂的 OPTIONAL 或 MINUS 数据块），则 Neptune 查询引擎将在可能的情况下对查询重新排序，以获得高效的计算顺序。

对于更复杂的查询，Neptune 选择计算查询的顺序可能并不总是最佳的。例如，Neptune 可能会丢失在计算查询期间发生的实例数据特定的特征（例如命中图形中的电源节点）。

如果您知道确切的数据特性并且想要手动指示查询执行顺序，请使用 Neptune `joinOrder` 查询提示指定按给定的顺序计算查询。

`joinOrder` SPARQL 提示语法

`joinOrder` 查询提示被指定为 SPARQL 查询中包含的三元组模式。

为清晰起见，以下语法使用查询中定义并包含的 `hint` 前缀来指定 Neptune 查询提示命名空间：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
scope hint:joinOrder "Ordered" .
```

可用范围

- `hint:Query`
- `hint:Group`

有关查询提示范围的更多信息，请参阅 Neptune 中的 SPARQL 查询提示范围。

`joinOrder` SPARQL 提示示例

本节介绍使用和未使用 `joinOrder` 查询提示和相关优化编写的查询。

在此示例中，假定数据集包含以下内容：

- 一个名为 `John` 的人（`:likes` 1000 人，包括 `Jane`）。
- 一个名为 `Jane` 的人（`:likes` 10 人，包括 `John`）。

无查询提示

以下 SPARQL 查询将从一组社交网络数据中提取所有名为 `John` 和 `Jane` 的人员（相互关注）对：

```
PREFIX : <https://example.com/>
SELECT ?john ?jane {
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
```

```
}
```

Neptune 查询引擎可能会以不同于编写的顺序计算语句。例如，它可能选择按以下顺序计算：

1. 查找所有名为 John 的人。
2. 查找通过 :likes 边缘连接到 John 的所有人。
3. 按名为 Jane 的人筛选此集。
4. 按通过 :likes 边缘连接到 John 的人筛选此集。

根据此数据集，按此顺序进行计算会导致在第二步中提取 1000 个实体。第三步将此范围缩小到单个节点 Jane。之后，最后一步确定 Jane 也将 :likes 节点 John。

查询提示

从 Jane 节点开始是有利的，因为她只有 10 个传出 :likes 边缘。这将避免在第二步中提取 1000 个实体，从而减少了查询计算期间的工作量。

以下示例使用 joinOrder 查询提示，通过对查询禁用所有自动联接重新排序来确保先处理 Jane 节点及其传出边缘：

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

一个适用的现实场景可能是社交网络应用程序，其中网络中的人员被分为具有许多连接的影响者和具有很少连接的普通用户。在此场景中，您可以确保在类似于上一个示例的查询中，先处理普通用户 (Jane)，再处理影响者 (John)。

查询提示和重新排序

您可以进一步了解此示例。如果您知道 :name 属性对于单一节点是唯一的，则可通过重新排序和使用 joinOrder 查询提示来加快查询速度。此步骤确保首先提取唯一节点。

```
PREFIX : <https://example.com/>
```

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?john ?jane {
  hint:Query hint:joinOrder "Ordered" .
  ?person1 :name "Jane" .
  ?person2 :name "John" .
  ?person1 :likes ?person2 .
  ?person2 :likes ?person1 .
}
```

在此情况下，您可以在每个步骤中将查询减少到以下单个操作：

1. 查找包含 `:name Jane` 的单个人节点。
2. 查找包含 `:name John` 的单个人节点。
3. 检查使用 `:likes` 边缘连接到第二个节点的第一个节点。
4. 检查使用 `:likes` 边缘连接到第一个节点的第二个节点。

> **Important**
>
> 如果您选择了错误的顺序，则 `joinOrder` 查询提示可能会导致性能显著下降。例如，如果 `:name` 属性不是唯一的，则上一个示例的效率低下。如果所有 100 个节点都名为 `Jane` 并且所有 1000 个节点都名为 `John`，则查询将最终检查 `:likes` 边缘的 1000 * 100 (100000) 个对。

evaluationStrategy SPARQL 查询提示

`evaluationStrategy` 查询提示告知 Amazon Neptune 查询引擎，注释的查询片段应作为独立单元按自下而上的顺序计算。这意味着，不使用之前计算步骤中的任何解来计算此查询片段。此查询片段将作为独立单元计算，并且其生成的解将在计算后与查询剩余的部分联接。

使用 `evaluationStrategy` 查询提示是指一个阻止（非管道）查询计划，这意味着使用查询提示注释的片段的解将在主内存中实现并缓冲。使用此查询提示可能会显著增加计算查询所需的主内存量，尤其是在带注释的查询片段计算大量结果时。

evaluationStrategy SPARQL 提示语法

`evaluationStrategy` 查询提示被指定为 SPARQL 查询中包含的三元组模式。

为清晰起见，以下语法使用查询中定义并包含的 `hint` 前缀来指定 Neptune 查询提示命名空间：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
hint:SubQuery hint:evaluationStrategy "BottomUp" .
```

可用范围

- `hint:SubQuery`

> **Note**
>
> 仅嵌套子查询中支持此查询提示。

有关查询提示范围的更多信息，请参阅 Neptune 中的 SPARQL 查询提示范围。

`evaluationStrategy` SPARQL 提示示例

本节介绍使用和未使用 `evaluationStrategy` 查询提示和相关优化编写的查询。

在此示例中，假定数据集具有以下特征：

- 它包含 1000 个标记有 `:connectedTo` 的边缘。
- 每个 `component` 节点平均连接到另外 100 个 `component` 节点。
- 各个节点之间的四跃点周期性连接的典型数量约为 100。

无查询提示

以下 SPARQL 查询提取通过四个跃点来循环互连的所有 `component` 节点：

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  ?component1 :connectedTo ?component2 .
  ?component2 :connectedTo ?component3 .
  ?component3 :connectedTo ?component4 .
  ?component4 :connectedTo ?component1 .
}
```

Neptune 查询引擎的方法旨在使用以下步骤计算此查询：

- 提取图形中的所有 1000 个 connectedTo 边缘。
- 扩展 100 倍（component2 中的传出 connectedTo 边缘的数量）。

  中间结果：100000 个节点。
- 扩展 100 倍（component3 中传出 connectedTo 边缘的数量）。

  中间结果：10000000 个节点。
- 扫描 10000000 个节点以查找循环关闭。

这将产生一个流式处理查询计划，此计划具有固定数量的主内存。

查询提示和子查询

您可能希望权衡主内存空间以加速计算。通过使用 evaluationStrategy 查询提示重新编写查询，您可强制引擎计算两个更小、更具体的子集之间的联接。

```
PREFIX : <https://example.com/>
          PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
  {
    SELECT * WHERE {
      hint:SubQuery hint:evaluationStrategy "BottomUp" .
      ?component1 :connectedTo ?component2 .
      ?component2 :connectedTo ?component3 .
    }
  }
  {
    SELECT * WHERE {
      hint:SubQuery hint:evaluationStrategy "BottomUp" .
      ?component3 :connectedTo ?component4 .
      ?component4 :connectedTo ?component1 .
    }
  }
}
```

当迭代使用来自前一个三元组模式的结果作为后续模式的输入时，evaluationStrategy 提示不是按顺序计算三元组模式，而是使这两个子查询被单独计算。这两个子查询将为中间结果生成 100000 个节点，之后它们将互联以形成最终输出。

具体而言，当您在较大的实例类型上运行 Neptune 时，将这两个 100000 子集临时存储在主内存中会增加内存占用量，从而显著加快计算速度。

queryTimeout SPARQL 查询提示

queryTimeout 查询提示指定超时，该值短于在数据库参数组中设置的 neptune_query_timeout 值。

如果查询因为此提示而终止，则会引发 TimeLimitExceededException，并显示 Operation terminated (deadline exceeded) 消息。

queryTimeout SPARQL 提示语法

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ... WHERE {
    hint:Query hint:queryTimeout 10 .
    # OR
    hint:Query hint:queryTimeout "10" .
    # OR
    hint:Query hint:queryTimeout "10"^^xsd:integer .
 ...
}
```

该超时值以毫秒表示。

该超时值必须小于数据库参数组中设置的 neptune_query_timeout 值。否则，将会引发带 Malformed query: Query hint 'queryTimeout' must be less than neptune_query_timeout DB Parameter Group 消息的 MalformedQueryException 异常。

queryTimeout 查询提示应在主查询的 WHERE 子句中指定，或者在某个子查询的 WHERE 子句中指定，如下例所示。

它必须在所有查询/子查询和 SPARQL 更新部分（例如 INSERT 和 DELETE）中只设置一次。否则，将会引发带 Malformed query: Query hint 'queryTimeout' must be set only once 消息的 MalformedQueryException 异常。

可用范围

queryTimeout 提示可以应用于 SPARQL 查询和更新。

- 在 SPARQL 查询中，它可以出现在主查询或子查询的 WHERE 子句中。
- 在 SPARQL 更新中，可以在 INSERT、DELETE 或 WHERE 子句中对其进行设置。如果有多个 update 子句，则只能在其中一个中进行设置。

有关查询提示范围的更多信息，请参阅 Neptune 中的 SPARQL 查询提示范围。

queryTimeout SPARQL 提示示例

此处是在 UPDATE 主查询的 WHERE 子句中使用 hint:queryTimeout 的示例。

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
INSERT {
    ?s ?p ?o
} WHERE {
    hint:Query hint:queryTimeout 100 .
    ?s ?p ?o .
}
```

此处，hint:queryTimeout 位于子查询的 WHERE 子句中：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
   ?s ?p ?o .
   {
      SELECT ?s WHERE {
         hint:Query hint:queryTimeout 100 .
         ?s ?p1 ?o1 .
      }
   }
}
```

rangeSafe SPARQL 查询提示

使用此查询提示可关闭 SPARQL 查询的类型提升。

当您提交对数值或范围进行 FILTER 的 SPARQL 查询时，Neptune 查询引擎在执行查询时通常必须使用类型提升。这意味着它必须检查每种类型的值，这些值可能包含您正在筛选的值。

例如，如果要筛选等于 55 的值，则引擎必须查找等于 55 的整数、等于 55L 的长整数、等于 55.0 的浮点数，依此类推。每个类型提升都需要在存储上进行额外查找，这可能会导致看似简单的查询需要意想不到的长时间才能完成。

通常没有必要进行类型提升，因为您事先知道只需要查找一种特定类型的值即可。在这种情况下，您可以使用 rangeSafe 查询提示来关闭类型提升，从而大大加快查询速度。

`rangeSafe` SPARQL 提示语法

`rangeSafe` 查询提示的值为 `true` 将关闭类型提升。它还接受值 `false`（默认值）。

示例。以下示例说明在对大于 1 的 o 整数值进行筛选时如何关闭类型提升：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * {
   ?s ?p ?o .
   hint:Prior hint:rangeSafe 'true' .
   FILTER (?o > '1'^^<http://www.w3.org/2001/XMLSchema#int>)
```

`queryId` SPARQL 查询提示

使用该查询提示将您自己的 queryId 值分配给 SPARQL 查询。

例如：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT * WHERE {
  hint:Query hint:queryId "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47"
  {?s ?p ?o}}
```

您分配的值必须在 Neptune 数据库的所有查询间保持唯一。

`useDFE` SPARQL 查询提示

使用此查询提示可以启用 DFE 来执行查询。默认情况下，如果不将此查询提示设置为 `true`，Neptune 就不会使用 DFE，因为 neptune_dfe_query_engine 实例参数默认为 `viaQueryHint`。如果将该实例参数设置为 `enabled`，则除 `useDFE` 查询提示设置为 `false` 的查询外，所有查询都将使用 DFE 引擎。

启用 DFE 进行查询的示例：

```
PREFIX : <https://example.com/>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>

SELECT ?john ?jane
{
  hint:Query hint:useDFE true .
```

```
  ?person1 :name "Jane" .
  ?person1 :likes ?person2 .
  ?person2 :name "John" .
  ?person2 :likes ?person1 .
}
```

与 DESCRIBE 结合使用的 SPARQL 查询提示

SPARQL DESCRIBE 查询为请求资源描述提供了一种灵活的机制。但是，SPARQL 规范并未定义 DESCRIBE 的精确语义。

从引擎版本 1.2.0.2 开始，Neptune 支持几种适合不同情况的不同 DESCRIBE 模式和算法。

此示例数据集可以帮助说明不同的模式：

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <https://example.com/> .

:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JohnDoe :firstName "John" .
:JaneDoe :knows _:b1 .
_:b1 :knows :RichardRoe .

:RichardRoe :knows :JaneDoe .
:RichardRoe :firstName "Richard" .

_:s1 rdf:type rdf:Statement .
_:s1 rdf:subject :JaneDoe .
_:s1 rdf:predicate :knows .
_:s1 rdf:object :JohnDoe .
_:s1 :knowsFrom "Berlin" .

:ref_s2 rdf:type rdf:Statement .
:ref_s2 rdf:subject :JaneDoe .
:ref_s2 rdf:predicate :knows .
:ref_s2 rdf:object :JohnDoe .
:ref_s2 :knowsSince 1988 .
```

以下示例假设使用如下所示的 SPARQL 查询来请求资源 :JaneDoe 的描述：

```
DESCRIBE <https://example.com/JaneDoe>
```

describeMode SPARQL 查询提示

hint:describeMode SPARQL 查询提示用于选择 Neptune 支持的以下 SPARQL DESCRIBE 模式之一：

ForwardOneStep DESCRIBE 模式

您可以使用 describeMode 查询提示调用 ForwardOneStep 模式，如下所示：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "ForwardOneStep"
}
```

ForwardOneStep 模式仅返回要描述的资源的属性和正向链接。在示例中，这意味着它将具有 :JaneDoe（待描述的资源）的三元组作为主题返回：

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b301990159 .
```

请注意，DESCRIBE 查询可能会返回带有空白节点的三元组，例如 _:b301990159，它与输入数据集相比，每次具有不同的 ID。

SymmetricOneStep DESCRIBE 模式

如果您不提供查询提示，则 SymmetricOneStep 是默认的 DESCRIBE 模式。您也可以使用 describeMode 查询提示显式调用它，如下所示：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "SymmetricOneStep"
}
```

在 SymmetricOneStep 语义下，DESCRIBE 返回要描述的资源的属性、正向链接和反向链接：

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b318767375 .
```

```
_:b318767631 rdf:subject :JaneDoe .

:RichardRoe :knows :JaneDoe .

:ref_s2 rdf:subject :JaneDoe .
```

简明界限描述 (**CBD**) DESCRIBE 模式

使用 `describeMode` 查询提示调用简明界限描述 (CBD) 模式，如下所示：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "CBD"
}
```

在 CBD 语义下，DESCRIBE 返回要描述的资源的简明界限描述（如 W3C 定义）：

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b285212943 .
_:b285212943 :knows :RichardRoe .

_:b285213199 rdf:subject :JaneDoe .
_:b285213199 rdf:type rdf:Statement .
_:b285213199 rdf:predicate :knows .
_:b285213199 rdf:object :JohnDoe .
_:b285213199 :knowsFrom "Berlin" .

:ref_s2 rdf:subject :JaneDoe .
```

RDF 资源（即 RDF 图形中的节点）的简明界限描述是以能够独立的节点为中心的最小子图形。实际上，这意味着，如果您将此图形视为一棵树，以指定的节点为根，则该树的叶子中没有空白节点 (bnode)。由于空白节点无法在外部寻址，也无法在后续查询中使用，因此仅仅浏览图形来查找当前节点中接下来的一个或多个单跳是不够的。您还必须找到可以在后续查询中使用的节点（即空白节点以外的节点）。

计算 CBD

给定源 RDF 图形中的特定节点（起始节点或根），该节点的 CBD 计算如下：

1. 在子图形中包括源图形中所有以语句的主题为起始节点的语句。
2. 以递归方式，对于子图形中迄今为止具有空白节点对象的所有语句，请在子图形中包含源图形中语句的主题为该空白节点且尚未包含在子图形中的所有语句。
3. 以递归方式，对于子图形中迄今为止包含的所有语句，对于源图形中对这些语句的所有修改，包括从每个具体化的 `rdf:Statement` 节点开始的 CBD。

这会生成一个子图形，其中对象节点要么是 IRI 引用，要么是文本，要么是空白节点，它们不作为图形中任何语句的主题。请注意，不能使用单个 SPARQL SELECT 或 CONSTRUCT 查询来计算 CBD。

对称简明界限描述 (**SCBD**) DESCRIBE 模式

使用 `describeMode` 查询提示调用对称简明界限描述 (SCBD) 模式，如下所示：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE <https://example.com/JaneDoe>
{
  hint:Query hint:describeMode "SCBD"
}
```

在 SCBD 语义下，DESCRIBE 返回资源的对称简明界限描述（由 W3C 在使用 VoID 词汇描述关联数据集中定义：

```
:JaneDoe :firstName "Jane" .
:JaneDoe :knows :JohnDoe .
:JaneDoe :knows _:b335544591 .
_:b335544591 :knows :RichardRoe .

:RichardRoe :knows :JaneDoe .

_:b335544847 rdf:subject :JaneDoe .
_:b335544847 rdf:type rdf:Statement .
_:b335544847 rdf:predicate :knows .
_:b335544847 rdf:object :JohnDoe .
_:b335544847 :knowsFrom "Berlin" .

:ref_s2 rdf:subject :JaneDoe .
```

与 `ForwardOneStep` 和 `SymmetricOneStep` 模式相比，CBD 和 SCBD 的优势在于，空白节点总是会扩展为包括它们的表示形式。这可能是一个重要的优势，因为您无法使用 SPARQL 查询空白节点。此外，CBD 和 SCBD 模式也考虑具体化。

请注意，describeMode 查询提示也可以是 WHERE 子句的一部分：

```
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
DESCRIBE ?s
WHERE {
  hint:Query hint:describeMode "CBD" .
  ?s rdf:type <https://example.com/Person>
}
```

describeIterationLimit SPARQL 查询提示

hint:describeIterationLimit SPARQL 查询提示为要针对迭代 DESCRIBE 算法（例如 CBD 和 SCBD）执行的最大迭代扩展次数提供了一个可选约束。

DESCRIBE 限制需要同时满足才行。因此，如果同时指定了迭代限制和语句限制，则必须同时满足这两个限制，然后才能断开 DESCRIBE 查询。

此值的默认值为 5。您可以将其设置为零 (0)，以指定对迭代扩展的数量没有限制。

describeStatementLimit SPARQL 查询提示

hint:describeStatementLimit SPARQL 查询提示提供了对 DESCRIBE 查询响应中可能存在的最大语句数的可选约束。它仅适用于迭代 DESCRIBE 算法，例如 CBD 和 SCBD。

DESCRIBE 限制需要同时满足才行。因此，如果同时指定了迭代限制和语句限制，则必须同时满足这两个限制，然后才能断开 DESCRIBE 查询。

此值的默认值为 5000。您可以将其设置为零 (0)，以指定对返回的语句数量没有限制。

SPARQL DESCRIBE 相对于默认图形的行为

SPARQL DESCRIBE 查询表单允许您在不了解数据结构和不必编写查询的情况下检索有关资源的信息。如何收集这些信息留待 SPARQL 实现来决定。Neptune 提供了几个查询提示，以调用不同的模式和算法供 DESCRIBE 使用。

在 Neptune 的实现中，无论模式如何，DESCRIBE 都只使用 SPARQL 默认图形中存在的数据。这与 SPARQL 处理数据集的方式一致（请参阅 SPARQL 规范中的指定 RDF 数据集）。

在 Neptune 中，除非使用 FROM 和/或 FROM NAMED 子句指定特定的命名图形，否则默认图形包含数据库中所有命名图形的并集中的所有唯一三元组。Neptune 中的所有 RDF 数据都存储在命名图形中。如果在没有命名图形上下文的情况下插入三元组，Neptune 会将其存储在名为 http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph 的命名图形中。

当使用 FROM 子句指定一个或多个命名图形时，默认图形是这些命名图形中所有唯一三元组的并集。如果没有 FROM 子句但有一个或多个 FROM NAMED 子句，则默认图形为空。

SPARQL **DESCRIBE** 示例

请考虑以下数据：

```
PREFIX ex: <https://example.com/>

GRAPH ex:g1 {
    ex:s ex:p1 "a" .
    ex:s ex:p2 "c" .
}

GRAPH ex:g2 {
    ex:s ex:p3 "b" .
    ex:s ex:p2 "c" .
}

ex:s ex:p3 "d" .
```

对于此查询：

```
PREFIX ex: <https://example.com/>
DESCRIBE ?s
FROM ex:g1
FROM NAMED ex:g2
WHERE {
  GRAPH ex:g2 { ?s ?p "b" . }
}
```

Neptune 将返回：

```
ex:s ex:p1 "a" .
ex:s ex:p2 "c" .
```

在此，首先计算图形模式 GRAPH ex:g2 { ?s ?p "b" }（这会生成 ?s 的绑定），然后在默认图形上计算 DESCRIBE 部分，现在仅仅是 ex:g1。

但是，对于此查询：

```
PREFIX ex: <https://example.com/>
```

```
DESCRIBE ?s
FROM NAMED ex:g1
WHERE {
  GRAPH ex:g1 { ?s ?p "a" . }
}
```

Neptune 不会返回任何内容，因为当存在 FROM NAMED 子句而没有任何 FROM 子句时，默认图形为空。

在以下查询中，在不存在 FROM 或 FROM NAMED 子句的情况下使用 DESCRIBE：

```
PREFIX ex: <https://example.com/>
DESCRIBE ?s
WHERE {
  GRAPH ex:g1 { ?s ?p "a" . }
}
```

在这种情况下，默认图形由数据库中所有命名图形并集中的所有唯一三元组组成（正式名称为 RDF 合并），因此 Neptune 将返回：

```
ex:s ex:p1 "a" .
ex:s ex:p2 "c" .
ex:s ex:p3 "b" .
ex:s ex:p3 "d" .
```

SPARQL 查询状态 API

要获取 SPARQL 查询的状态，请使用 HTTP GET 或 POST 来向 https://*your-neptune-endpoint*:*port*/sparql/status 终端节点提出请求。

SPARQL 查询状态请求参数

queryId（可选）

正在运行的 SPARQL 查询的 ID。仅显示指定查询的状态。

SPARQL 查询状态响应语法

```
{
    "acceptedQueryCount": integer,
    "runningQueryCount": integer,
    "queries": [
```

```
      {
        "queryId":"guid",
        "queryEvalStats":
          {
            "subqueries": integer,
            "elapsed": integer,
            "cancelled": boolean
          },
        "queryString": "string"
      }
    ]
}
```

SPARQL 查询状态响应值

已接受 QueryCount

自上次重启 Neptune 引擎以来接受的查询数。

正在运行 QueryCount

当前正在运行的 SPARQL 查询数。

查询

当前 SPARQL 查询的列表。

queryId

查询的 GUID id。Neptune 为每个查询自动分配该 ID 值，或者您也可以分配自己的 ID（请参阅将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中）。

查询 EvalStats

此查询的统计数据。

subqueries

此查询中的子查询数。

elapsed

到目前为止，查询已运行的毫秒数。

cancelled

True 指示查询已取消。

queryString

已提交的查询。

SPARQL 查询状态示例

以下是使用 curl 和 HTTP GET 的状态命令的示例。

```
curl https://your-neptune-endpoint:port/sparql/status
```

此输出显示了一个正在运行的查询。

```
{
    "acceptedQueryCount":9,
    "runningQueryCount":1,
    "queries": [
        {
            "queryId":"fb34cd3e-f37c-4d12-9cf2-03bb741bf54f",
            "queryEvalStats":
                {
                    "subqueries": 0,
                    "elapsed": 29256,
                    "cancelled": false
                },
            "queryString": "SELECT ?s ?p ?o WHERE {?s ?p ?o}"
        }
    ]
}
```

SPARQL 查询取消

要获取 SPARQL 查询的状态，请使用 HTTP GET 或 POST 来向 https://*your-neptune-endpoint*:*port*/sparql/status 终端节点提出请求。

SPARQL 查询取消请求参数

cancelQuery

（必需）告知状态命令取消查询。该参数不接收值。

queryId

（必需）要取消的正在运行的 SPARQL 查询的 ID。

silent

（可选）如果 `silent=true`，则取消正在运行的查询，并且 HTTP 响应代码为 200。如果不存在 `silent` 或 `silent=false`，则取消查询，并且返回 HTTP 500 状态代码。

SPARQL 查询取消示例

示例 1：在 **`silent=false`** 的情况下取消

以下是在将 `silent` 参数设置为 `false` 的情况下使用 `curl` 取消查询的状态命令的示例：

```
curl https://your-neptune-endpoint:port/sparql/status \
  -d "cancelQuery" \
  -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \
  -d "silent=false"
```

除非查询已开始流式传输结果，否则取消的查询将返回一个 HTTP 500 代码，其响应如下：

```
{
  "code": "CancelledByUserException",
  "requestId": "4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47",
  "detailedMessage": "Operation terminated (cancelled by user)"
}
```

如果查询返回了一个 HTTP 200 代码 (OK)，并且在取消之前已开始流式传输结果，则超时异常信息将发送到常规输出流。

示例 2：在 **`silent=true`** 的情况下取消

以下状态命令与上面的示例相同，但在本示例中，`silent` 参数设置为 `true`：

```
curl https://your-neptune-endpoint:port/sparql/status \
  -d "cancelQuery" \
  -d "queryId=4d5c4fae-aa30-41cf-9e1f-91e6b7dd6f47" \
  -d "silent=true"
```

该命令将返回与 `silent=false` 时相同的响应，但这次取消的查询将返回 HTTP 200 代码，其响应如下：

```
{
```

```
  "head" : {
    "vars" : [ "s", "p", "o" ]
  },
  "results" : {
    "bindings" : [ ]
  }
}
```

在 Amazon Neptune 中使用 SPARQL 1.1 图形存储 HTTP 协议 (GSP)

在 SPARQL 1.1 图形存储 HTTP 协议建议中，W3C 定义了一种用于管理 RDF 图形的 HTTP 协议。它定义了用于删除、创建和替换 RDF 图形内容以及向现有内容添加 RDF 语句的操作。

图形存储协议 (GSP) 提供了一种无需编写复杂的 SPARQL 查询即可操作整个图形的便捷方式。

截至版本：1.0.5.0（2021 年 7 月 27 日），Neptune 完全支持该协议。

图形存储协议 (GSP) 的端点是：

```
https://your-neptune-cluster:port/sparql/gsp/
```

要使用 GSP 访问默认图形，请使用：

```
https://your-neptune-cluster:port/sparql/gsp/?default
```

要使用 GSP 访问命名图形，请使用：

```
https://your-neptune-cluster:port/sparql/gsp/?graph=named-graph-URI
```

Neptune GSP 实现的特别细节

Neptune 完全实现了定义 GSP 的 W3C 建议。但是，此规范没有涵盖某些情况。

其中一种情况是，PUT 或 POST 请求在请求正文中指定了一个或多个命名图形，这些图形与由请求 URL 指定的图形不同。只有当请求正文 RDF 格式支持命名图形时，才会发生这种情况，例如使用 `Content-Type: application/n-quads` 或 `Content-Type: application/trig`。

在这种情况下，Neptune 会添加或更新正文中存在的所有命名图形，以及在 URL 中指定的命名图形。

例如，假设您从一个空数据库开始，您发送 PUT 请求将选票更新插入到三个图形中。其中一个图形名为 `urn:votes`，它包含所有选举年份的所有选票。另外两个名为 `urn:votes:2005` 和 `urn:votes:2019` 的图形包含特定选举年份的选票。请求及其有效负载如下所示：

```
PUT "http://your-Neptune-cluster:port/sparql/gsp/?graph=urn:votes"
  Host: example.com
  Content-Type: application/n-quads

  PAYLOAD:

  <urn:JohnDoe> <urn:votedFor> <urn:Labour> <urn:votes:2005>
  <urn:JohnDoe> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
  <urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
  <urn:JaneSmith> <urn:votedFor> <urn:Conservative> <urn:votes:2019>
```

请求执行后，数据库中的数据如下所示：

```
<urn:JohnDoe>   <urn:votedFor> <urn:Labour>           <urn:votes:2005>
<urn:JohnDoe>   <urn:votedFor> <urn:Conservative>     <urn:votes:2019>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes:2005>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative>     <urn:votes:2019>
<urn:JohnDoe>   <urn:votedFor> <urn:Labour>           <urn:votes>
<urn:JohnDoe>   <urn:votedFor> <urn:Conservative>     <urn:votes>
<urn:JaneSmith> <urn:votedFor> <urn:LiberalDemocrats> <urn:votes>
<urn:JaneSmith> <urn:votedFor> <urn:Conservative>     <urn:votes>
```

另一种模棱两可的情况是，使用 PUT、POST、GET 或 DELETE 中的任何一个操作在请求 URL 本身中指定多个图形。例如：

```
POST "http://your-Neptune-cluster:port/sparql/gsp/?
graph=urn:votes:2005&graph=urn:votes:2019"
```

或者：

```
GET "http://your-Neptune-cluster:port/sparql/gsp/?default&graph=urn:votes:2019"
```

在这种情况下，Neptune 返回 HTTP 400，并显示一条消息指示在请求 URL 中只能指定一个图形。

使用 SPARQL **explain** 分析 Neptune 查询执行

Amazon Neptune 添加了名为 explain 的 SPARQL 特征。此特征是一种自助式工具，用于了解 Neptune 引擎所采用的执行方法。您可以通过将 `explain` 参数添加到提交 SPARQL 查询的 HTTP 调用来调用它。

explain 功能提供了有关查询执行计划的逻辑结构的信息。您可以使用此信息来识别潜在的评估和执行瓶颈。然后，您可以使用查询提示来改进查询执行计划。

主题

- SPARQL 查询引擎在 Neptune 中的工作原理
- 如何使用 SPARQL explain 分析 Neptune 查询执行
- 在 Neptune 中调用 SPARQL explain 的示例
- Neptune SPARQL explain 运算符
- Neptune 中 SPARQL explain 的限制

SPARQL 查询引擎在 Neptune 中的工作原理

要使用 SPARQL explain 特征提供的信息，您需要了解有关 Amazon Neptune SPARQL 查询引擎的工作原理的一些详细信息。

该引擎将每个 SPARQL 查询转换到一个运算符管道中。从第一个运算符开始，称为绑定列表 的中间解将流经此运算符管道。您可以将绑定列表视为一个表，其中，表标头是查询中使用的变量的子集。表中的每一行表示一个结果（截至到评估点）。

假设已为我们的数据定义两个命名空间前缀：

```
@prefix ex:   <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

下面将是此上下文中一个简单的绑定列表的示例：

```
?person       | ?firstName
-------------------------------------------------------
ex:JaneDoe    | "Jane"
ex:JohnDoe    | "John"
ex:RichardRoe | "Richard"
```

对于三个人员中的每一个人，列表会将 ?person 变量绑定到该人员的标识符，并将 ?firstName 变量绑定到该人员的名字。

在一般情况下，变量可以保持未绑定状态，例如，查询中的某个变量存在 OPTIONAL 选项，对于此选项，数据中不存在任何值。

PipelineJoin 运算符是 explain 输出中存在的 Neptune 查询引擎运算符的示例。它将来自上一个运算符的传入绑定集作为输入，并将其与三元组模式（即 (?person, foaf:lastName, ?lastName)）进行联接。此操作在其输入流中使用 ?person 变量的绑定，将绑定替换到三元组模式中并从数据库中查找三元组。

在来自上一个表的传入绑定的上下文中执行时，PipelineJoin 将计算三个查找，即以下内容：

```
(ex:JaneDoe,    foaf:lastName, ?lastName)
(ex:JohnDoe,    foaf:lastName, ?lastName)
(ex:RichardRoe, foaf:lastName, ?lastName)
```

此方法称为按绑定 计算。此计算过程的解将与传入的解进行联接，同时在传入的解中填充检测到的 ?lastName。假设您发现全部三位人员均有一个姓氏，该运算符将生成一个内容如下所示的传出绑定列表：

```
?person       | ?firstName | ?lastName
---------------------------------------
ex:JaneDoe    | "Jane"     | "Doe"
ex:JohnDoe    | "John"     | "Doe"
ex:RichardRoe | "Richard"  | "Roe"
```

然后，此传出绑定列表将用作管道中的下一个运算符的输入。最后，管道中的最后一个运算符的输出定义查询结果。

运算符管道通常是线性的，这意味着，每个运算符将针对单个连接的运算符发送解。但是，在某些情况下，它们可能具有更复杂的结构。例如，SPARQL 查询中的 UNION 运算会将映射到 Copy 操作。此操作将复制绑定并将副本转发到两个子计划中，其中一个子计划用于 UNION 的左侧，另一个则用于右侧。

有关运算符的更多信息，请参阅Neptune SPARQL explain 运算符。

如何使用 SPARQL **explain** 分析 Neptune 查询执行

SPARQL explain 特征是 Amazon Neptune 中的一种自助式工具，可帮助您了解 Neptune 引擎所采用的执行方法。要调用 explain，请采用 explain=*mode* 形式将参数传递到 HTTP 或 HTTPS 请求。

模式值可为 static、dynamic 或 details 之一。

- 在静态 模式下，`explain` 仅输出查询计划的静态结构。
- 在动态 模式下，`explain` 还包含查询计划的动态方面。这些方面可能包括流经运算符的中间绑定的数量、传入绑定与传出绑定的比率以及运算符使用的总时间。
- 在详细信息模式下，`explain` 输出 `dynamic` 模式中显示的信息以及其他详细信息，例如实际的 SPARQL 查询字符串和某个连接运算符之下模式的估计范围计数。

Neptune 支持将 `explain` 与 W3C SPARQL 1.1 协议规范中列出的所有三个 SPARQL 查询访问协议结合使用，即：

1. HTTP GET
2. 使用 URL 编码的参数的 HTTP POST
3. 使用文本参数的 HTTP POST

有关 SPARQL 查询引擎的信息，请参阅 SPARQL 查询引擎在 Neptune 中的工作原理。

有关通过调用 SPARQL `explain` 生成的输出类型的更多信息，请参阅 在 Neptune 中调用 SPARQL `explain` 的示例。

在 Neptune 中调用 SPARQL **`explain`** 的示例

本部分中的示例显示了您通过在 Amazon Neptune 中调用 SPARQL `explain` 特征来分析查询执行可生成的各种输出。

主题

- 了解 Explain 输出
- 详细信息模式输出示例
- 静态模式输出的示例
- 为参数编码的不同方式
- text/plain 以外的其它输出类型
- 启用 DFE 时的 SPARQL explain 输出示例

了解 Explain 输出

在此示例中，Jane Doe 认识两个人，即 John Doe 和 Richard Roe：

```
@prefix ex: <http://example.com> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:JaneDoe foaf:knows ex:JohnDoe .
ex:JohnDoe foaf:firstName "John" .
ex:JohnDoe foaf:lastName "Doe" .
ex:JaneDoe foaf:knows ex:RichardRoe .
ex:RichardRoe foaf:firstName "Richard" .
ex:RichardRoe foaf:lastName "Roe" .
.
```

要确定 Jane Doe 认识的所有人员的名字，您可以编写以下查询：

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -H "Accept: text/csv"
```

此简单查询将返回以下内容：

```
firstName
John
Richard
```

接下来，更改 `curl` 命令以调用 `explain`，方式是添加 `-d "explain=dynamic"` 并使用默认输出类型而不是 `text/csv`：

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=dynamic"
```

查询现在将返回采用美观的 ASCII 格式（HTTP 内容类型 `text/plain`）的输出，这是默认输出类型：

```
################################################################################################
```

```
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     # Units In # Units Out # Ratio # Time (ms) #
#########################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        # 0        # 1         # 0.00  # 0         #
#########################################################################################
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        # 1        # 2         # 2.00  # 1         #
#    #        #        #                   # joinType=join
            #          #          #           #       #           #
#    #        #        #                   # joinProjectionVars=[?person]
            #          #          #           #       #           #
#########################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        # 2        # 2         # 1.00  # 1         #
#    #        #        #                   # joinType=join
            #          #          #           #       #           #
#    #        #        #                   # joinProjectionVars=[?person, ?firstName]
            #          #          #           #       #           #
#########################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
            # retain   # 2        # 2         # 1.00  # 0         #
#########################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
            # id2value # 2        # 2         # 1.00  # 1         #
#########################################################################################
```

有关 Name 列中的操作及其参数的详细信息，请参阅 explain 运算符。

下面对输出进行了逐行介绍：

1. 主查询中的第一步始终使用 SolutionInjection 运算符来注入解。然后，注入的解将通过计算过程扩展到最终结果。

   在这种情况下，它会注入所谓的通用解 { }。在存在 VALUES 子句或 BIND 的情况下，此步骤还可能注入更复杂的变量绑定来开始计算。

   Units Out 列指示此单一解流出运算符。Out #1 列指定此运算符将结果注入到的运算符。在本示例中，所有运算符均连接到表中的下一个运算符。

2. 第二步是 PipelineJoin。它将接收作为上一个运算符 (Units In := 1) 生成的单一通用（完全不受约束的）解。它将它与 pattern 参数定义的三元组模式进行联接。这对应于该模式的简单查找。在这种情况下，三元组模式将定义为以下内容：

```
distinct( ex:JaneDoe, foaf:knows, ?person )
```

joinType := join 参数表示这是一个常规联接（其他类型包括 optional 联接、existence check 联接等）。

distinct := true 参数表示您仅从数据库中提取不同的匹配（无重复），并且您将不同的匹配绑定到变量 joinProjectionVars := ?person（重复）。

事实上，Units Out 列值为 2，表示有两个解流出。具体来说，这些是 ?person 变量的绑定，从而向这两位人员反映该数据显示 Jane Doe 知道：

```
?person
-------------
ex:JohnDoe
ex:RichardRoe
```

3. 阶段 2 中的两个解作为输入 (Units In := 2) 流入第二个 PipelineJoin。此运算符将两个之前的解与以下三元组模式联接：

```
distinct(?person, foaf:firstName, ?firstName)
```

已知 ?person 变量要由运算符的传入解绑定到 ex:JohnDoe 或 ex:RichardRoe。假定 PipelineJoin 提取名字 John 和 Richard。然后，两个传出解 (Units Out := 2) 如下所示：

```
?person       | ?firstName
---------------------------
ex:JohnDoe    | John
ex:RichardRoe | Richard
```

4. 下一个投影运算符将阶段 3 中的两个解 (Units In := 2) 作为输入并投影于 ?firstName 变量上。这将消除映射中的所有其他变量绑定并传递两个绑定 (Units Out := 2)：

```
?firstName
----------
John
Richard
```

5. 为了提高性能，在可能的情况下，Neptune 会针对其分配到一些术语（如 URI 和字符串文本）的内部标识符运算，而不是针对字符串本身运算。最后一个运算符 `TermResolution` 执行从这些内部标识符映射回相应词语字符串的工作。

   在常规（非解释）查询评估中，最后一个运算符计算的结果随后会序列化为请求的序列化格式，并流式传输到客户端。

详细信息模式输出示例

> **Note**
>
> SPARQL explain 详细信息模式从 Neptune 引擎版本 1.0.2.1 开始推出。

假设您在详细信息 模式而不是动态 模式下运行与前面相同的查询：

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=details"
```

如此示例所示，输出相同，但有一些额外的详细信息，例如位于输出顶部的查询字符串，以及 `PipelineJoin` 运算符的 `patternEstimate` 计数：

```
Query:
PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://www.example.com/>
SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person foaf:firstName ?
firstName }

################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        # 0         # 1          # 0.00  # 0         #
################################################################################################
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        # 1         # 2          # 2.00  # 13        #
```

```
#    #        #        #                  # joinType=join
            #          #          #           #       #           #
#    #        #        #                  # joinProjectionVars=[?person]
            #          #          #           #       #           #
#    #        #        #                  # patternEstimate=2
            #          #          #           #       #           #
#####################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        # 2        # 2         # 1.00  # 3         #
#    #        #        #                  # joinType=join
            #          #          #           #       #           #
#    #        #        #                  # joinProjectionVars=[?person, ?firstName]
            #          #          #           #       #           #
#    #        #        #                  # patternEstimate=2
            #          #          #           #       #           #
#####################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
            # retain   # 2        # 2         # 1.00  # 1         #
#####################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
            # id2value # 2        # 2         # 1.00  # 7         #
#####################################################################################
```

静态模式输出的示例

假设您在静态 模式（默认）而不是详细信息 模式下运行与上一个查询相同的查询：

```
 curl http(s)://your_server:your_port/sparql \
   -d "query=PREFIX foaf: <https://xmlns.com/foaf/0.1/> PREFIX ex: <https://
www.example.com/> \
       SELECT ?firstName WHERE { ex:JaneDoe foaf:knows ?person . ?person
 foaf:firstName ?firstName }" \
   -d "explain=static"
```

如本示例所示，输出是相同的，但它省略了最后三个列：

```
#####################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments
            # Mode     #
#####################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
            # -        #
#####################################################################################
```

```
# 1  # 2      # -      # PipelineJoin      # pattern=distinct(ex:JaneDoe, foaf:knows, ?
person)     # -        #
#    #        #        #                   # joinType=join
            #          #
#    #        #        #                   # joinProjectionVars=[?person]
            #          #
##########################################################################################
# 2  # 3      # -      # PipelineJoin      # pattern=distinct(?person,
 foaf:firstName, ?firstName) # -        #
#    #        #        #                   # joinType=join
            #          #
#    #        #        #                   # joinProjectionVars=[?person, ?firstName]
            #          #
##########################################################################################
# 3  # 4      # -      # Projection        # vars=[?firstName]
            # retain   #
##########################################################################################
# 4  # -      # -      # TermResolution    # vars=[?firstName]
            # id2value #
##########################################################################################
```

为参数编码的不同方式

以下示例查询说明了调用 SPARQL `explain` 时编码参数的两种不同方式。

使用 URL 编码 - 此示例使用参数的 URL 编码并指定动态 输出：

```
curl -XGET "http(s)://your_server:your_port/sparql?query=SELECT%20*%20WHERE%20%7B
%20%3Fs%20%3Fp%20%3Fo%20%7D%20LIMIT%20%31&explain=dynamic"
```

直接指定参数 - 这与上一个查询相同，但它是通过 POST 直接传递参数：

```
curl http(s)://your_server:your_port/sparql \
  -d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
  -d "explain=dynamic"
```

text/plain 以外的其它输出类型

前面的示例使用默认 `text/plain` 输出类型。Neptune 还可以将 SPARQL `explain` 输出格式化为另外两种 MIME 类型的格式，即 `text/csv` 和 `text/html`。通过设置 HTTP `Accept` 标头（可使用 `curl` 中的 `-H` 标记来实现）调用它们，如下所示：

```
  -H "Accept: output type"
```

下面是一些示例：

text/csv 输出

此查询通过指定 -H "Accept: text/csv" 调用 CSV MIME 类型输出：

```
curl http(s)://your_server:your_port/sparql \
  -d "query=SELECT * WHERE { ?s ?p ?o } LIMIT 1" \
  -d "explain=dynamic" \
  -H "Accept: text/csv"
```

方便导入到电子表格或数据库中的 CSV 格式使用分号 (;) 分隔每个 explain 行中的字段，如下所示：

```
ID;Out #1;Out #2;Name;Arguments;Mode;Units In;Units Out;Ratio;Time (ms)
0;1;-;SolutionInjection;solutions=[{}];-;0;1;0.00;0
1;2;-;PipelineJoin;pattern=distinct(?s, ?p, ?o),joinType=join,joinProjectionVars=[?s, ?
p, ?o];-;1;6;6.00;1
2;3;-;Projection;vars=[?s, ?p, ?o];retain;6;6;1.00;2
3;-;-;Slice;limit=1;-;1;1;1.00;1
```

text/html 输出

如果您指定 -H "Accept: text/html"，则 explain 将生成 HTML 表：

```
<!DOCTYPE html>
<html>
  <body>
    <table border="1px">
      <thead>
        <tr>
          <th>ID</th>
          <th>Out #1</th>
          <th>Out #2</th>
          <th>Name</th>
          <th>Arguments</th>
          <th>Mode</th>
          <th>Units In</th>
```

```
          <th>Units Out</th>
          <th>Ratio</th>
          <th>Time (ms)</th>
        </tr>
      </thead>

      <tbody>
        <tr>
          <td>0</td>
          <td>1</td>
          <td>-</td>
          <td>SolutionInjection</td>
          <td>solutions=[{}]</td>
          <td>-</td>
          <td>0</td>
          <td>1</td>
          <td>0.00</td>
          <td>0</td>
        </tr>

        <tr>
          <td>1</td>
          <td>2</td>
          <td>-</td>
          <td>PipelineJoin</td>
          <td>pattern=distinct(?s, ?p, ?o)<br>
              joinType=join<br>
              joinProjectionVars=[?s, ?p, ?o]</td>
          <td>-</td>
          <td>1</td>
          <td>6</td>
          <td>6.00</td>
          <td>1</td>
        </tr>

        <tr>
          <td>2</td>
          <td>3</td>
          <td>-</td>
          <td>Projection</td>
          <td>vars=[?s, ?p, ?o]</td>
          <td>retain</td>
          <td>6</td>
          <td>6</td>
```

```
          <td>1.00</td>
          <td>2</td>
        </tr>

        <tr>
          <td>3</td>
          <td>-</td>
          <td>-</td>
          <td>Slice</td>
          <td>limit=1</td>
          <td>-</td>
          <td>1</td>
          <td>1</td>
          <td>1.00</td>
          <td>1</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

HTML 在浏览器中呈现如下所示的内容：

ID	Out #1	Out #2	Name	Arguments	Mode	Units In	Units Out	Ratio	Time (ms)
0	1	-	SolutionInjection	solutions=[{}]	-	0	1	0.00	0
1	2	-	PipelineJoin	pattern=distinct(?s, ?p, ?o) joinType=join joinProjectionVars=[?s, ?p, ?o]	-	1	6	6.00	1
2	3	-	Projection	vars=[?s, ?p, ?o]	retain	6	6	1.00	2
3	-	-	Slice	limit=1	-	1	1	1.00	1

启用 DFE 时的 SPARQL **explain** 输出示例

以下是启用 Neptune DFE 替代查询引擎时的 SPARQL `explain` 输出的示例：

```
################################################################################################
# ID # Out #1 # Out #2 # Name              # Arguments


 # Mode     # Units In # Units Out # Ratio # Time (ms) #
################################################################################################
# 0  # 1      # -      # SolutionInjection # solutions=[{}]
```

```
 # -        # 0        # 1         # 0.00  # 0         #
##########################################################################################
# 1  # 2      # -      # HashIndexBuild    # solutionSet=solutionSet1


 # -        # 1        # 1         # 1.00  # 22        #
#    #        #        #                   # joinVars=[]


 #          #          #           #       #           #
#    #        #        #                   # sourceType=pipeline


 #          #          #           #       #           #
##########################################################################################
# 2  # 3      # -      # DFENode            # DFE Stats=


      # -        # 101      # 100       # 0.99  # 32        #
#    #        #        #                   # ====> DFE execution time (measured by
 DFEQueryEngine)

     #          #          #           #       #           #
#    #        #        #                   # accepted [micros]=127


 #          #          #           #       #           #
#    #        #        #                   # ready [micros]=2


 #          #          #           #       #           #
#    #        #        #                   # running [micros]=5627


 #          #          #           #       #           #
#    #        #        #                   # finished [micros]=0


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
```

```
#    #        #        #                  #


 #          #          #          #       #           #
#    #        #        #                  # ===> DFE execution time (measured in
 DFENode)

      #          #          #          #       #           #
#    #        #        #                  # -> setupTime [ms]=1


 #          #          #          #       #           #
#    #        #        #                  # -> executionTime [ms]=14


 #          #          #          #       #           #
#    #        #        #                  # -> resultReadTime [ms]=0


 #          #          #          #       #           #
#    #        #        #                  #


 #          #          #          #       #           #
#    #        #        #                  #


 #          #          #          #       #           #
#    #        #        #                  # ===> Static analysis statistics


 #          #          #          #       #           #
#    #        #        #                  # --> 35907 micros spent in parser.


 #          #          #          #       #           #
#    #        #        #                  # --> 7643 micros spent in range count
 estimation

      #          #          #          #       #           #
#    #        #        #                  # --> 2895 micros spent in value resolution


 #          #          #          #       #           #
```

```
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # --> 39974925 micros spent in optimizer
 loop

    #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # DFEJoinGroupNode[ children={


  #          #          #           #       #           #
#    #        #        #                  #   DFEPatternNode[(?1, TERM[117442062], ?
2, ?3) . project DISTINCT[?1, ?2] {rangeCountEstimate=100},

  #          #          #           #       #           #
#    #        #        #                  #     OperatorInfoWithAlternative[


  #          #          #           #       #           #
#    #        #        #                  #       rec=OperatorInfo[


  #          #          #           #       #           #
#    #        #        #                  #         type=INCREMENTAL_PIPELINE_JOIN,


 #          #          #           #       #           #
#    #        #        #                  #
 costEstimates=OperatorCostEstimates[

                                   #          #          #           #       #
  #
#    #        #        #                  #
 costEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0002,comp=0.0000,mem=0],
```

```
                                    #           #           #           #       #
 #
#    #        #        #                    #
 worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0002,comp=0.0000,mem=0]

                       #           #           #           #       #           #
#    #        #        #                    #       alt=OperatorInfo[


  #           #           #           #       #           #
#    #        #        #                    #         type=INCREMENTAL_HASH_JOIN,


 #           #           #           #       #           #
#    #        #        #                    #
 costEstimates=OperatorCostEstimates[

                                    #           #           #           #       #
  #
#    #        #        #                    #
 costEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0003,comp=0.0000,mem=3212],

                                #           #           #           #       #           #
#    #        #        #                    #
 worstCaseCostEstimate=OperatorCostEstimate[in=1.0000,out=100.0000,io=0.0003,comp=0.0000,mem=32

                  #           #           #           #       #           #
#    #        #        #                    #   DFEPatternNode[(?1, TERM[150997262], ?
4, ?5) . project DISTINCT[?1, ?4] {rangeCountEstimate=100},

  #           #           #           #       #           #
#    #        #        #                    #     OperatorInfoWithAlternative[


  #           #           #           #       #           #
#    #        #        #                    #       rec=OperatorInfo[


  #           #           #           #       #           #
#    #        #        #                    #         type=INCREMENTAL_HASH_JOIN,


 #           #           #           #       #           #
```

```
#    #        #        #                   #
 costEstimates=OperatorCostEstimates[

                                    #          #          #           #       #
  #
#    #        #        #                   #
 costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=6400],

                              #          #          #           #       #           #
#    #        #        #                   #
 worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0003,comp=0.0000,mem=

                 #          #          #           #       #           #
#    #        #        #                   #       alt=OperatorInfo[


  #          #          #           #       #           #
#    #        #        #                   #         type=INCREMENTAL_PIPELINE_JOIN,


 #          #          #           #       #           #
#    #        #        #                   #
 costEstimates=OperatorCostEstimates[

                                    #          #          #           #       #
  #
#    #        #        #                   #
 costEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=0],

                                 #          #          #           #       #           #
#    #        #        #                   #
 worstCaseCostEstimate=OperatorCostEstimate[in=100.0000,out=100.0000,io=0.0010,comp=0.0000,mem=

                   #          #          #           #       #           #
#    #        #        #                   # },


 #          #          #           #       #           #
#    #        #        #                   # ]


 #          #          #           #       #           #
#    #        #        #                   #
```

```
 #          #          #           #       #           #
#    #        #        #                   # ===> DFE configuration:


 #          #          #           #       #           #
#    #        #        #                   # solutionChunkSize=5000


 #          #          #           #       #           #
#    #        #        #                   # ouputQueueSize=20


 #          #          #           #       #           #
#    #        #        #                   # numComputeCores=3


 #          #          #           #       #           #
#    #        #        #                   # maxParallelIO=10


 #          #          #           #       #           #
#    #        #        #                   # numInitialPermits=12


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
#    #        #        #                   # ====> DFE configuration (reported back)


 #          #          #           #       #           #
#    #        #        #                   # numComputeCores=3


 #          #          #           #       #           #
#    #        #        #                   # maxParallelIO=2
```

```
 #          #          #           #       #           #
#    #        #        #                   # numInitialPermits=12


 #          #          #           #       #           #
#    #        #        #                   #


 #          #          #           #       #           #
#    #        #        #                   # ===> Statistics & operator histogram


 #          #          #           #       #           #
#    #        #        #                   # ==> Statistics


 #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 3668 micros total elapsed (incl.
 wait / excl. wait)

 #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 3 millis total elapse (incl.
 wait / excl. wait)

    #          #          #           #       #           #
#    #        #        #                   # -> 3741 / 0 secs total elapsed (incl.
 wait / excl. wait)

     #          #          #           #       #           #
#    #        #        #                   # ==> Operator histogram


 #          #          #           #       #           #
#    #        #        #                   # -> 47.66% of total time (excl. wait):
 pipelineScan (2 instances)

     #          #          #           #       #           #
#    #        #        #                   # -> 10.99% of total time (excl. wait):
 merge (1 instances)

     #          #          #           #       #           #
#    #        #        #                   # -> 41.17% of total time (excl. wait):
 symmetricHashJoin (1 instances)
```

```
     #           #           #            #       #            #
#    #        #        #                  # -> 0.19% of total time (excl. wait): drain
 (1 instances)

 #          #          #           #       #           #
#    #        #        #                  #


 #          #          #           #       #           #
#    #        #        #                  # nodeId | out0   | out1 | opName
   | args                                          | rowsIn | rowsOut | chunksIn |
 chunksOut | elapsed* | outWait | outBlocked | ratio    | rate* [M/s] | rate [M/s] | %
    #           #           #            #       #            #
#    #        #        #                  # ------ | ------ | ---- | -----------------
 | ------------------------------------------------ | ------ | ------- | -------- |
 --------- | -------- | ------- | ---------- | -------- | ----------- | ---------- |
 ----- #          #          #           #       #           #
#    #        #        #                  # node_0 | node_2 | -    | pipelineScan
  | (?1, TERM[117442062], ?2, ?3) DISTINCT [?1, ?2]  | 0      | 100     | 0        | 1
        | 874      | 0       | 0          | Infinity | 0.1144      | 0.1144     | 23.83
 #          #          #           #       #           #
#    #        #        #                  # node_1 | node_2 | -    | pipelineScan
  | (?1, TERM[150997262], ?4, ?5) DISTINCT [?1, ?4]  | 0      | 100     | 0        | 1
        | 874      | 0       | 0          | Infinity | 0.1144      | 0.1144     | 23.83
 #          #          #           #       #           #
#    #        #        #                  # node_2 | node_4 | -    | symmetricHashJoin
 |                                                 | 200    | 100     | 2        | 2
        | 1510     | 73      | 0          | 0.50     | 0.0662      | 0.0632     | 41.17
 #          #          #           #       #           #
#    #        #        #                  # node_3 | -      | -    | drain
  |                                                 | 100    | 0       | 1        | 0
        | 7        | 0       | 0          | 0.00     | 0.0000      | 0.0000     | 0.19
 #          #          #           #       #           #
#    #        #        #                  # node_4 | node_3 | -    | merge
  |                                                 | 100    | 100     | 2        | 1
        | 403      | 0       | 0          | 1.00     | 0.2481      | 0.2481     | 10.99
 #          #          #           #       #           #
##########################################################################################
# 3  # 4      # -      # HashIndexJoin     # solutionSet=solutionSet1


 # -        # 100      # 100       # 1.00  # 4         #
```

```
#    #        #        #                  # joinType=join


 #          #          #           #       #           #
#########################################################################################
# 4  # 5      # -      # Distinct          # vars=[?s, ?o, ?o1]


 # -        # 100      # 100       # 1.00  # 9         #
#########################################################################################
# 5  # 6      # -      # Projection        # vars=[?s, ?o, ?o1]


 # retain   # 100      # 100       # 1.00  # 2         #
#########################################################################################
# 6  # -      # -      # TermResolution    # vars=[?s, ?o, ?o1]


 # id2value # 100      # 100       # 1.00  # 11        #
#########################################################################################
```

Neptune SPARQL **explain** 运算符

以下部分介绍 Amazon Neptune 中当前可用的 SPARQL `explain` 特征的运算符和参数。

> **Important**
>
> SPARQL `explain` 功能仍在不断优化中。此处记录的运算符和参数可能会在未来版本中发生更改。

主题

- Aggregation 运算符
- ConditionalRouting 运算符
- Copy 运算符
- DFENode 运算符
- Distinct 运算符
- Federation 运算符
- Filter 运算符

- HashIndexBuild 运算符
- HashIndexJoin 运算符
- MergeJoin 运算符
- NamedSubquery 运算符
- PipelineJoin 运算符
- PipelineCountJoin 运算符
- PipelinedHashIndexJoin 运算符
- Projection 运算符
- PropertyPath 运算符
- TermResolution 运算符
- Slice 运算符
- SolutionInjection 运算符
- Sort 运算符
- VariableAlignment 运算符

`Aggregation` 运算符

执行一个或多个聚合，同时实现 SPARQL 聚合运算符（如 `count`、`max`、`min`、`sum` 等）的语义。

`Aggregation` 带有使用 `groupBy` 子句的可选分组和可选 `having` 约束。

参数

- `groupBy` -（可选）提供一个 `groupBy` 子句，该子句指定作为传入解的分组依据的表达式序列。
- `aggregates` -（必需）指定聚合表达式的排序列表。
- `having` -（可选）添加约束以对组进行筛选，与 SPARQL 查询中的 `having` 子句的作用一样。

`ConditionalRouting` 运算符

根据某个给定条件路由传入解。满足该条件的解将会路由到 `Out #1` 引用的运算符 ID，而未满足该条件的解将不会路由到 `Out #2` 引用的运算符。

参数

- `condition` -（必需）路由条件。

Copy 运算符

根据指定模式指定的方式委派解流。

模式

- `forward` – 将解转发至由 `Out #1` 标识的下游运算符。
- `duplicate` - 复制解并将其转发至由 `Out #1` 和 `Out #2` 标识的两个运算符中的每一个。

`Copy` 没有任何参数。

DFENode 运算符

此运算符是对 DFE 备用查询引擎运行的计划的抽象。该运算符的参数中概述了详细的 DFE 计划。该参数目前已过载，无法包含 DFE 计划的详细运行时系统统计数据。它包含 DFE 在执行查询的各个步骤上所花费的时间。

DFE 查询计划的逻辑优化抽象语法树 (AST) 包含有关在计划时考虑的运算符类型以及运行运算符的相关最佳和最坏情况成本的信息。目前，AST 由以下类型的节点组成：

- `DFEJoinGroupNode` – 表示一个或多个 `DFEPatternNodes` 的联接。
- `DFEPatternNode` – 封装一种底层模式，使用该模式将匹配的元组投影到底层数据库之外。

小节 `Statistics & Operator histogram` 包含有关 `DataflowOp` 计划的执行时间和每个运算符使用的 CPU 时间细分的详细信息。下方有一个表，其中包含 DFE 执行的计划的详细运行时系统统计数据。

> **Note**
>
> 由于 DFE 是在实验室模式下发布的实验特征，因此其 `explain` 输出的确切格式可能会发生变化。

Distinct 运算符

计算变量子集上的不同投影，从而消除重复项。因此，流入的解的数量大于或等于流出的解的数量。

参数

- `vars` - (必需) 要将 `Distinct` 投影应用于的变量。

Federation 运算符

将指定的查询传递到指定的远程 SPARQL 终端节点。

参数

- endpoint - (必需) SPARQL SERVICE 语句中的端点 URL。这可以是常量字符串，或者如果查询终端节点是基于相同查询中的变量确定的，这也可以是变量名。
- query - (必需) 要发送到远程端点的重构查询字符串。即使客户端未指定任何前缀，引擎也会向该查询添加默认前缀。
- silent - (必需) 布尔值，指示关键字之后是否出现 SILENT 关键字。SILENT 告知引擎，即使远程 SERVICE 部分失败，也不要使整个查询失败。

Filter 运算符

筛选传入解。只有这些满足筛选条件的解才会转发至上游运算符，而所有其他解将被删除。

参数

- condition - (必需) 筛选条件。

HashIndexBuild 运算符

将绑定和假脱机的列表置于其名称由 solutionSet 参数定义的哈希索引中。通常情况下，后续运算符将针对此解集 (根据名称引用) 执行联接。

参数

- solutionSet - (必需) 哈希索引解的名称。
- sourceType - (必需) 从中获取要存储在哈希索引中的绑定的源的类型：
  - pipeline - 将来自运算符管道中的下游运算符的传入解假脱机到哈希索引中。
  - binding set - 将由 sourceBindingSet 参数指定的固定绑定集假脱机到哈希索引中。
- sourceBindingSet - (可选) 如果 sourceType 参数值为 binding set，此参数指定要假脱机到哈希索引中的静态绑定集。

HashIndexJoin 运算符

将传入解与由 solutionSet 参数标识的哈希索引解集进行联接。

参数

- solutionSet - (必需) 要对其进行联接的解集的名称。这必须是已在之前某一步中使用 HashIndexBuild 运算符构建的哈希索引。
- joinType - (必需) 要执行的联接类型：
  - join - 一个常规联接，要求所有共享变量之间的完全匹配。
  - optional - 一个 optional 联接，该联接使用 SPARQL OPTIONAL 运算符语义。
  - minus - minus 运算使用 SPARQL MINUS 运算符语义保留不存在任何联接合作伙伴的映射。
  - existence check - 检查是否有联接合作伙伴，并将 existenceCheckResultVar 变量绑定到此检查的结果。
- constraints - (可选) 联接期间要考虑的其它联接约束。不满足这些约束的联接将被丢弃。
- existenceCheckResultVar - (可选) 仅用于 joinType 等于 existence check 的联接 (请参阅之前的 joinType 参数)。

MergeJoin 运算符

针对多个解集的合并联接，由 solutionSets 参数标识。

参数

- solutionSets - (必需) 要联接在一起的解集。

NamedSubquery 运算符

触发由 subQuery 参数标识的子查询的评估并将结果假脱机到由 solutionSet 参数指定的解集中。该运算符的传入解将转发至子查询，然后转发至下一个运算符。

参数

- subQuery - (必需) 要计算的子查询的名称。子查询将在输出中明确显示。
- solutionSet - (必需) 要在其中存储子查询结果的解集的名称。

PipelineJoin 运算符

接收上一个运算符的输出作为输入并将它与 pattern 参数定义的元组模式进行联接。

参数

- pattern—（必需）模式，其形式为连接底层的图元组 subject-predicate-object，可选为-graph 元组。如果为该模式指定 distinct，联接将仅从由 projectionVars 参数指定的投影变量中提取不同的解，而不是所有匹配的解。
- inlineFilters -（可选）要应用于模式中的变量的一组筛选条件。将结合这些筛选条件对模式进行评估。
- joinType -（必需）要执行的联接类型：
  - join - 一个常规联接，要求所有共享变量之间的完全匹配。
  - optional - 一个 optional 联接，该联接使用 SPARQL OPTIONAL 运算符语义。
  - minus - minus 运算使用 SPARQL MINUS 运算符语义保留不存在任何联接合作伙伴的映射。
  - existence check - 检查是否有联接合作伙伴，并将 existenceCheckResultVar 变量绑定到此检查的结果。
- constraints -（可选）联接期间要考虑的其它联接约束。不满足这些约束的联接将被丢弃。
- projectionVars -（可选）投影变量。与 distinct := true 结合使用以强制提取某指定变量集中的不同投影。
- cutoffLimit -（可选）提取的联接合作伙伴的数量的截止限制。尽管默认情况下没有任何限制，但您可以在执行联接时将此值设置为 1 以实施 FILTER (NOT) EXISTS 子句，这样可足以证明或反证是否有联接合作伙伴。

PipelineCountJoin 运算符

PipelineJoin 的变体。它仅对匹配的联接合作伙伴进行计数并将计数绑定到由 countVar 参数指定的变量，而不是进行联接。

参数

- countVar -（必需）应将计数结果（即联接合作伙伴的数量）绑定到的变量。
- pattern—（必需）模式，其形式为连接底层的图元组 subject-predicate-object，可选为-graph 元组。如果为该模式指定 distinct，联接将仅从由 projectionVars 参数指定的投影变量中提取不同的解，而不是所有匹配的解。
- inlineFilters -（可选）要应用于模式中的变量的一组筛选条件。将结合这些筛选条件对模式进行评估。
- joinType -（必需）要执行的联接类型：
  - join - 一个常规联接，要求所有共享变量之间的完全匹配。

  - optional - 一个 optional 联接，该联接使用 SPARQL OPTIONAL 运算符语义。
  - minus - minus 运算使用 SPARQL MINUS 运算符语义保留不存在任何联接合作伙伴的映射。
  - existence check - 检查是否有联接合作伙伴，并将 existenceCheckResultVar 变量绑定到此检查的结果。
- constraints - （可选）联接期间要考虑的其它联接约束。不满足这些约束的联接将被丢弃。
- projectionVars - （可选）投影变量。与 distinct := true 结合使用以强制提取某指定变量集中的不同投影。
- cutoffLimit - （可选）提取的联接合作伙伴的数量的截止限制。尽管默认情况下没有任何限制，但您可以在执行联接时将此值设置为 1 以实施 FILTER (NOT) EXISTS 子句，这样可足以证明或反证是否有联接合作伙伴。

PipelinedHashIndexJoin 运算符

这是一个 all-in-one 生成哈希索引和连接运算符。它获取一个绑定列表，将绑定假脱机到哈希索引中，然后根据哈希索引连接传入的解。

参数

- sourceType - （必需）从中获取要存储在哈希索引中的绑定的源的类型，如下一种：
  - pipeline - 使 PipelinedHashIndexJoin 将来自运算符管道中的下游运算符的传入解假脱机到哈希索引中。
  - binding set - 使 PipelinedHashIndexJoin 将由 sourceBindingSet 参数指定的固定绑定集假脱机到哈希索引中。
- sourceSubQuery – （可选）如果 sourceType 参数值为 pipeline，则此参数指定要计算并假脱机到哈希索引中的子查询。
- sourceBindingSet – （可选）如果 sourceType 参数值为 binding set，此参数指定要假脱机到哈希索引中的静态绑定集。
- joinType - （必需）要执行的联接类型：
  - join - 一个常规联接，要求所有共享变量之间的完全匹配。
  - optional - 一个 optional 联接，该联接使用 SPARQL OPTIONAL 运算符语义。
  - minus - minus 运算使用 SPARQL MINUS 运算符语义保留不存在任何联接合作伙伴的映射。
  - existence check - 检查是否有联接合作伙伴，并将 existenceCheckResultVar 变量绑定到此检查的结果。

- existenceCheckResultVar – (可选) 仅用于 joinType 等于 existence check 的联接 (请参阅上面的 joinType 参数) 。

Projection 运算符

针对变量子集投影。流入的解数量等于流出的解数量，但解的形状有所不同，具体取决于模式设置。

模式

- retain - 仅保留解中由 vars 参数指定的变量。
- drop - 删除由 vars 参数指定的所有变量。

参数

- vars - (必需) 要保留或删除的变量，取决于模式设置。

PropertyPath 运算符

启用递归属性路径，例如 + 或 *。Neptune 基于 iterationTemplate 参数指定的模板实现了定点迭代方法。已知的左侧或右侧变量将在每次固定点迭代中绑定到模板中，直到再也找不到新解。

参数

- iterationTemplate - (必需) 用于实施固定点迭代的子查询模板的名称。
- leftTerm - (必需) 属性路径左侧的术语 (变量或常量) 。
- rightTerm - (必需) 属性路径右侧的术语 (变量或常量) 。
- lowerBound - (必需) 固定点迭代的下限 (* 查询的下限为 0，或 + 查询的下限为 1) 。

TermResolution 运算符

将内部字符串标识符值转换回其相应的外部字符串，或将外部字符串转换为内部字符串标识符值，具体取决于模式。

模式

- value2id - 将文本和 URI 等术语映射到相应的内部 ID 值 (编码为内部值) 。
- id2value - 将内部 ID 值映射到相应的术语 (如文本和 URI) (解码为内部值) 。

参数

- vars - (必需) 指定其字符串或内部字符串 ID 应进行映射的变量。

Slice 运算符

在传入解流中使用 SPARQL 的 LIMIT 和 OFFSET 子句的语义实施切片。

参数

- limit – (可选) 针对要进行转发的解的限制。
- offset – (可选) 用于计算解以进行转发的偏移。

SolutionInjection 运算符

未接收输入。将解静态注入查询计划中并将其记录在 solutions 参数中。

查询计划始终以此静态注入开始。如果要注入的静态解可通过组合各种来源的静态绑定 (例如，来自 VALUES 或 BIND 子句) 从查询本身进行派生，则 SolutionInjection 运算符将注入这些派生的静态解。在最简单的情况下，这些反映了外部 VALUES 子句所隐含的绑定。

如果没有任何静态解可从查询派生，SolutionInjection 将注入空的所谓的通用解，它将在整个查询-评估过程中进行扩展和增加。

参数

- solutions - (必需) 该运算符注入的解的序列。

Sort 运算符

使用指定的排序条件对解集排序。

参数

- sortOrder - (必需) 变量的排序列表，每个变量包含一个 ASC (升序) 或 DESC (降序) 标识符，用于对解集排序。

VariableAlignment 运算符

逐个检查解，同时对两个变量中的每个变量执行对齐：指定的 sourceVar 和指定的 targetVar。

如果解中的 sourceVar 和 targetVar 具有相同的值，则这些变量将被视为对齐并将转发解，同时会投影出多余的 sourceVar。

如果这些变量绑定到不同的值，则将完全筛选出该解。

参数

- sourceVar - （必需）要与目标变量比较的源变量。如果解中的对齐成功，则表示这两个变量具有相同的值，将投影出源变量。
- targetVar - （必需）要与源变量进行比较的目标变量。即使在对齐成功时也将保留。

Neptune 中 SPARQL **explain** 的限制

Neptune SPARQL explain 特征的版本具有以下限制。

Neptune 目前仅在 SPARQL SELECT 查询中支持 Explain

有关其他查询形式（如 ASK、CONSTRUCT、DESCRIBE 和 SPARQL UPDATE 查询）的评估过程的信息，您可以将这些查询转换为 SELECT 查询。然后，改用 explain 检查相应的 SELECT 查询。

例如，要获取有关 ASK WHERE {...} 查询的 explain 信息，请使用 explain 运行相应的 SELECT WHERE {...} LIMIT 1 查询。

同样，对于 CONSTRUCT {...} WHERE {...} 查询，删除 CONSTRUCT {...} 部分并使用 explain 在第二个 WHERE {...} 子句上运行 SELECT 查询。评估第二个 WHERE 子句通常会显示出处理 CONSTRUCT 查询的主要挑战，因为从第二个 WHERE 流出到 CONSTRUCT 模板的解通常仅需要简单替代。

Explain 运算符可能会在未来版本中发生更改

SPARQL explain 运算符及其参数可能会在未来版本中发生更改。

Explain 输出可能会在未来版本中发生更改

例如，列标头可能更改，而且可能会有更多列添加到表中。

Neptune 中使用 **SERVICE** 扩展的 SPARQL 联合查询

Amazon Neptune 完全支持使用 SERVICE 关键字的 SPARQL 联合查询扩展。（有关更多信息，请参阅 SPARQL 1.1 联合查询。）

> **Note**
>
> 从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始，此特征可用。

SERVICE 关键字指示 SPARQL 查询引擎针对远程 SPARQL 终端节点执行查询的一部分，并组成最终查询结果。只能执行 READ 操作。不支持 WRITE 和 DELETE 操作。Neptune 只能对在其虚拟私有云 (VPC) 中可访问的 SPARQL 端点运行联合查询。但是，您也可以在 VPC 中使用反向代理使外部数据源在 VPC 内可供访问。

> **Note**
>
> 当使用 SPARQL SERVICE 将查询联合到同一 VPC 中的两个或更多 Neptune 集群时，必须将安全组配置为允许所有这些 Neptune 集群相互通信。

> **Important**
>
> 将查询和参数传递到外部 SPARQL 终端节点时，SPARQL 1.1 Federation 代表您发出服务请求。您负责验证外部 SPARQL 终端节点是否满足应用程序的数据处理和安全要求。

Neptune 联合查询的示例

以下简单示例显示了如何使用 SPARQL 联合查询。

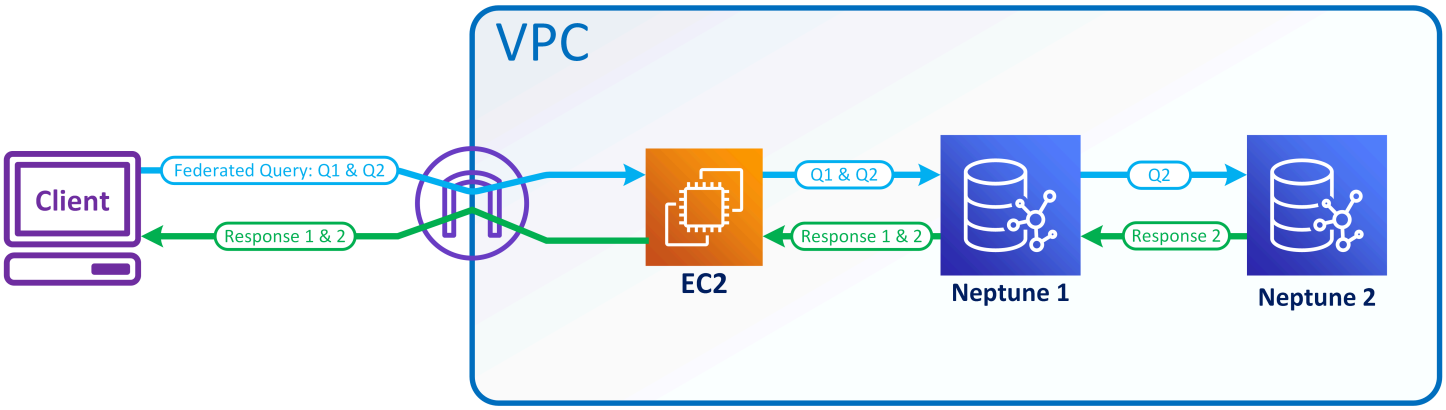
假设某位客户向 Neptune-1（位于 http://neptune-1:8182/sparql）发送了以下查询。

```
SELECT * WHERE {
   ?person rdf:type foaf:Person .
   SERVICE <http://neptune-2:8182/sparql> {
       ?person foaf:knows ?friend .
    }
}
```

1. Neptune-1 计算第一个查询模式 (Q-1)（即 ?person rdf:type foaf:Person），使用结果在 Q-2 (?person foaf:knows ?friend)中对 ?person 求解，然后将生成的模式转发给 Neptune-2（位于 http://neptune-2:8182/sparql）。
2. Neptune-2 对 Q-2 进行计算，并将结果发送回 Neptune-1。

3. Neptune-1 联接两个模式的解，并将结果发回给客户。

下图显示了该流程。



> **Note**
>
> 默认情况下，优化器决定在查询执行的哪个时刻执行 SERVICE 指令。您可以使用 joinOrder 查询提示覆盖此位置。

Neptune 中联合查询的访问控制

Neptune 使用 AWS Identity and Access Management (IAM) 进行身份验证和授权。联合查询的访问控制可以涉及多个 Neptune 数据库实例。这些实例可能对访问控制有不同的要求。在某些情况下，这可能会限制您进行联合查询的能力。

请考虑上一节介绍的简单示例。Neptune-1 使用与调用它时相同的凭证调用 Neptune-2。

- 如果 Neptune-1 需要 IAM 身份验证和授权，而 Neptune-2 不需要，那么只要您有适当的 IAM 权限就可以让 Neptune-1 进行联合查询。
- 如果 Neptune-1 和 Neptune-2 都需要 IAM 身份验证和授权，则您需要为两个数据库都附加 IAM 权限才能进行联合查询。两个集群还必须位于同一个 AWS 账户和同一区域。目前不支持跨区域和/或跨账户联合查询架构。
- 但是，在 Neptune-1 未启用 IAM 但 Neptune-2 启用 IAM 的情况下，您无法进行联合查询。原因是 Neptune-1 无法检索您的 IAM 凭证并将其传递给 Neptune-2 来授权执行查询的第二部分。

Neptune 的图形可视化工具

除了 Nep tune 图形笔记本内置的可视化功能外，您还可以使用 AWS 合作伙伴和第三方供应商构建的解决方案来可视化存储在 Neptune 中的数据。

复杂的图形可视化可以帮助组织中的数据科学家、经理和其它角色以交互方式探索图形数据，而不必知道如何编写复杂的查询。

主题

- 开源图形浏览器
- Tom Sawyer 软件
- Cambridge Intelligence
- Graphistry
- metaphacts
- G.V( )
- Linkurious

开源图形浏览器

图形浏览器是一款用于图形数据的开源低代码可视化探索工具，在 Apache-2.0 许可证下可用。它允许您在图形数据库中浏览带标签的属性图 (LPG) 或资源描述框架 (RDF) 数据，而无需编写图形查询。图形浏览器旨在帮助组织中的数据科学家、业务分析师和其它角色以交互方式探索图形数据，而无需学习图形查询语言。

图形浏览器提供了一个基于 React 的 Web 应用程序，可以将其部署为容器来可视化图形数据。您可以连接到 Amazon Neptune 或其他提供 Apache TinkerPop Gremlin 或 SPARQL 1.1 端点的图形数据库。

- 您可以使用分面筛选条件快速查看数据摘要，也可以通过在搜索栏中键入文本来搜索数据。
- 还可以交互式浏览节点和边缘连接。您可以查看节点邻居以查看对象之间如何相关，然后深入了解以直观地检查边缘和属性。
- 您还可以自定义图形布局、颜色、图标以及要为节点和边缘显示的默认属性。对于 RDF 图形，您也可以为资源 URI 自定义命名空间。
- 对于涉及图形数据的报告和演示文稿，您可以配置并保存以高分辨率 PNG 格式创建的视图。您也可以将关联的数据下载到 CSV 或 JSON 文件中以供进一步处理。

在 Neptune 图形笔记本中使用图形浏览器

在 Neptune 中使用图形浏览器的最简单方法是采用 Neptune 图形笔记本。

如果您使用 Neptune Workbench 托管 Neptune 笔记本，则图形浏览器会自动与笔记本一起部署并连接到 Neptune。

创建笔记本后，转到 Neptune 控制台以启动图形浏览器：

1. 转到 Neptune。
2. 在笔记本下，选择您的笔记本。
3. 在“操作”下，选择打开 Graph Explorer。

如何在 AWS Fargate 上的 Amazon ECS 中运行图形资源管理器并连接到 Neptune

您还可以构建 graph-explorer Docker 镜像，然后在本地计算机或托管服务（例如亚马逊弹性计算云 (Amazon EC2) 或亚马逊弹性容器服务 (Amazon ECS) Elastic Container Service（Amazon ECS）上运行，如图形资源管理器项目自述的入门部分所述。 GitHub

例如，本节提供了在 Amazon ECS 中运行图形浏览器的 step-by-step 说明： AWS Fargate

1. 创建新的 IAM 角色并向其附加这些策略。

   - 亚马逊 ECS TaskExecution RolePolicy
   - CloudWatchLogsFull访问权限

   请随时准备好角色名称，以便在短时间内使用。
2. 创建一个 Amazon ECS 集群，其基础设施设置为 FARGATE，并使用以下联网选项：

   - VPC：设置为 Neptune 数据库所在的 VPC。
   - Subnets：设置为该 VPC 的公有子网（移除所有其它子网）。
3. 按如下方式创建新的 JSON 任务定义：

```
{
  "family": "explorer-test",
  "containerDefinitions": [
    {
```

```
      "name": "graph-explorer",
      "image": "public.ecr.aws/neptune/graph-explorer:latest",
      "cpu": 0,
      "portMappings": [
        {
          "name": "graph-explorer-80-tcp",
          "containerPort": 80,
          "hostPort": 80,
          "protocol": "tcp",
          "appProtocol": "http"
        },
        {
          "name": "graph-explorer-443-tcp",
          "containerPort": 443,
          "hostPort": 443,
          "protocol": "tcp",
          "appProtocol": "http"
        }
      ],
      "essential": true,
      "environment": [
        {
          "name": "HOST",
          "value": "localhost"
        }
      ],
      "mountPoints": [],
      "volumesFrom": [],
      "logConfiguration": {
        "logDriver": "awslogs",
        "options": {
          "awslogs-create-group": "true",
          "awslogs-group": "/ecs/graph-explorer",
          "awslogs-region": "{region}",
          "awslogs-stream-prefix": "ecs"
        }
      }
    }
  ],
  "taskRoleArn": "arn:aws:iam::{account_no}:role/{role_name_from_step_1}",
  "executionRoleArn": "arn:aws:iam::{account_no}:role/{role_name_from_step_1}",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
    "FARGATE"
```

```
  ],
  "cpu": "1024",
  "memory": "3072",
  "runtimePlatform": {
    "cpuArchitecture": "X86_64",
    "operatingSystemFamily": "LINUX"
  }
}
```

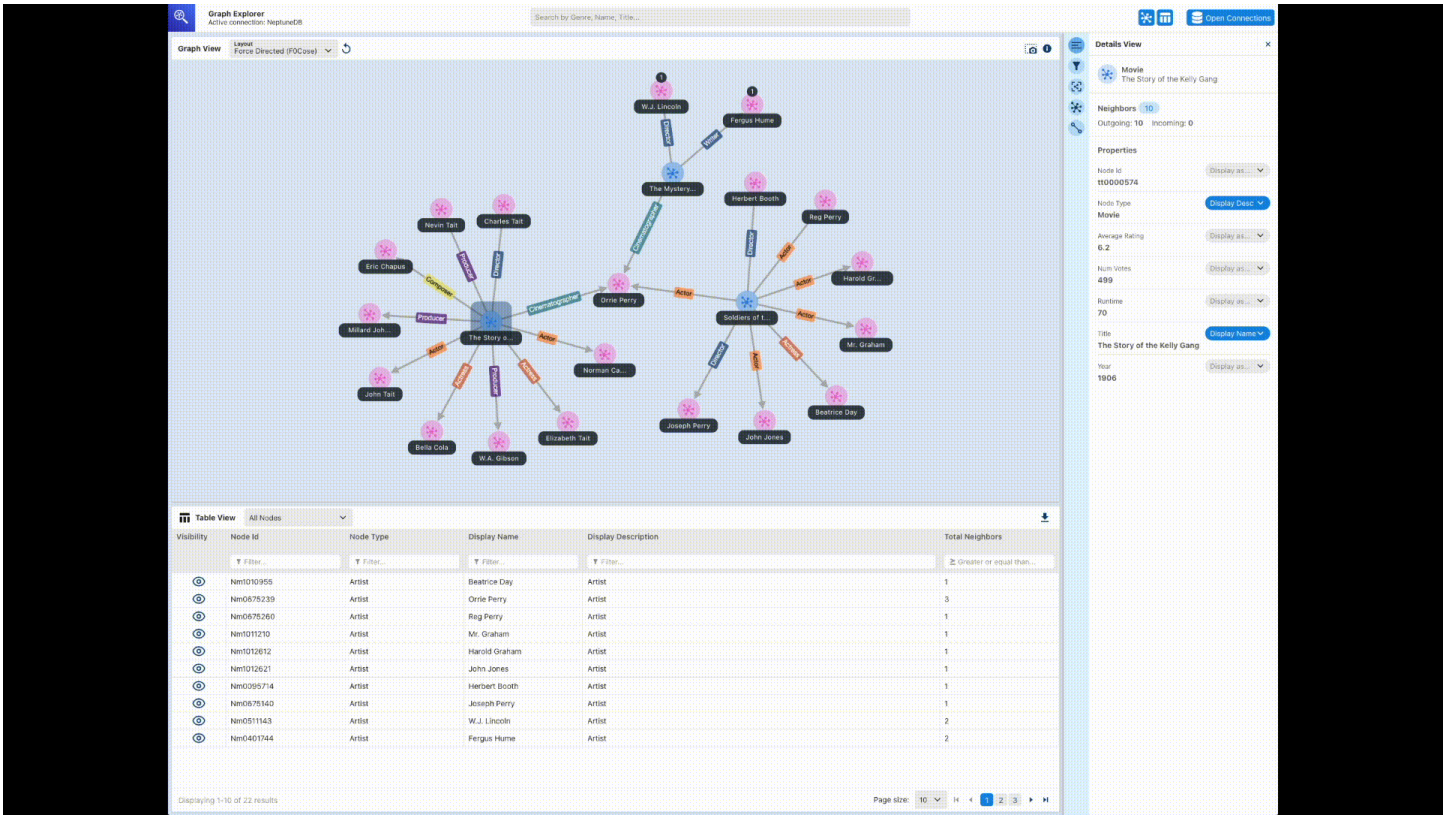
4. 使用默认设置启动新任务，但以下字段除外：
   - 环境
     - 计算选项 => 启动类型
   - 部署配置
     - 应用程序类型 => 任务
     - 系列 => *### JSON #####*
     - 修订版 => *####*
   - 联网
     - VPC => *####### Neptune VPC*)
     - 子网 => *### VPC ##### - #########*
     - 安全组 => 创建新的安全组
     - 安全组名称 => 图形浏览器
     - 安全组描述 = 用于访问图形浏览器的安全组
     - 安全组的入站规则 =>
       1. 80 Anywhere
       2. 443 Anywhere
5. 选择创建。
6. 任务启动后，复制正在运行的任务的公有 IP，然后导航到：`https://(your public IP)/explorer`。
7. 接受使用已生成的无法识别的证书的风险，或者将其添加到您的密钥链中。
8. 现在，您可以添加到 Neptune 的连接。为属性图 (LPG) 或 RDF 创建新连接，并设置以下字段：

```
Using proxy server => true
Public or Proxy Endpoint => https://(your public IP address)
Graph connection URL => https://(your Neptune endpoint):8182
```

您现在应该已连接。

图形浏览器演示

这段简短视频可让您大致了解了如何使用图形浏览器轻松可视化图形数据：



Tom Sawyer 软件

Tom Sawyer Perspectives 是一个低代码图形、数据可视化和分析开发平台，面向存储在 Amazon Neptune 中的数据。集成的设计和预览界面以及广泛的 API 库，使您能够快速创建定制、达到生产质量的可视化应用程序。借助 point-and-click 设计器界面和 30 种内置分析算法，您可以设计和开发应用程序，深入了解来自数十个来源的数据。

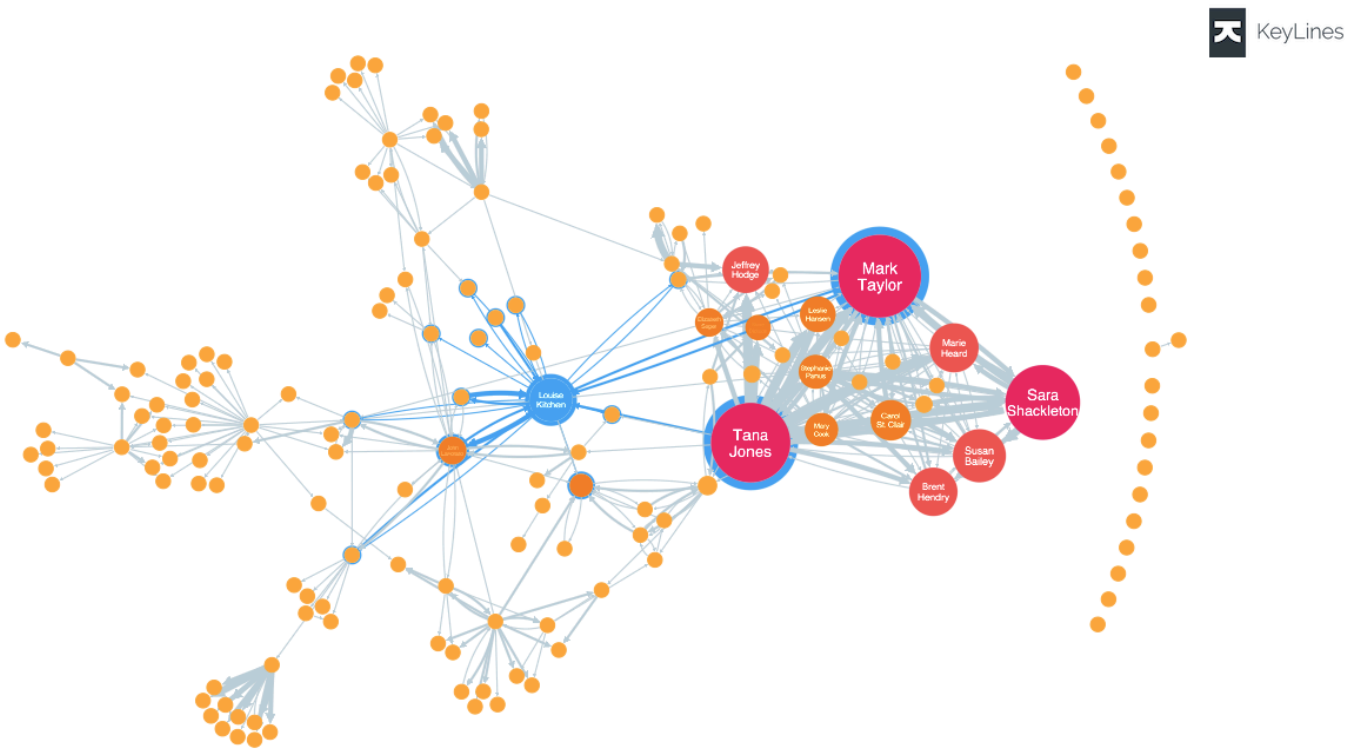
使用 Tom Sawyer 图形数据库浏览器可以轻松可视化和分析 Amazon Neptune 中的数据。无需对查询语言或架构有广泛的了解，即可查看和理解数据中的关联。您可以在没有技术知识的情况下与数据交互，只需加载选定节点的邻居，并在所需的任何方向构建可视化即可。您还可以利用五种独特的图形布局以最具含义的方式显示图形，并可以应用中心性、聚类和路径查找分析来揭示以前看不见的模式。要查看图形数据库浏览器与 Neptune 集成的示例，请查看此博客文章。要开始免费试用图形数据库浏览器，请访问 AWS Marketplace。

Cambridge Intelligence

Cambridge Intelligence 提供数据可视化技术，用于探索和理解 Amazon Neptune 数据。图表可视化工具包（KeyLines适用于 JavaScript 开发人员和 ReGraphReact 开发人员）提供了一种为 Web 应用程序构建高度交互性和可自定义工具的简便方法。这些工具包利用 WebGL 和 HTML5 Canvas 来提高性能，它们支持高级图形分析功能，并将灵活性和可扩展性与安全、强大的架构相结合。这些 SDK 可处理 Neptune Gremlin 和 RDF 数据。

请查看这些集成教程，了解 Gremlin 数据、SPARQL 数据和 Neptune 架构。

以下是一个 KeyLines 可视化示例：

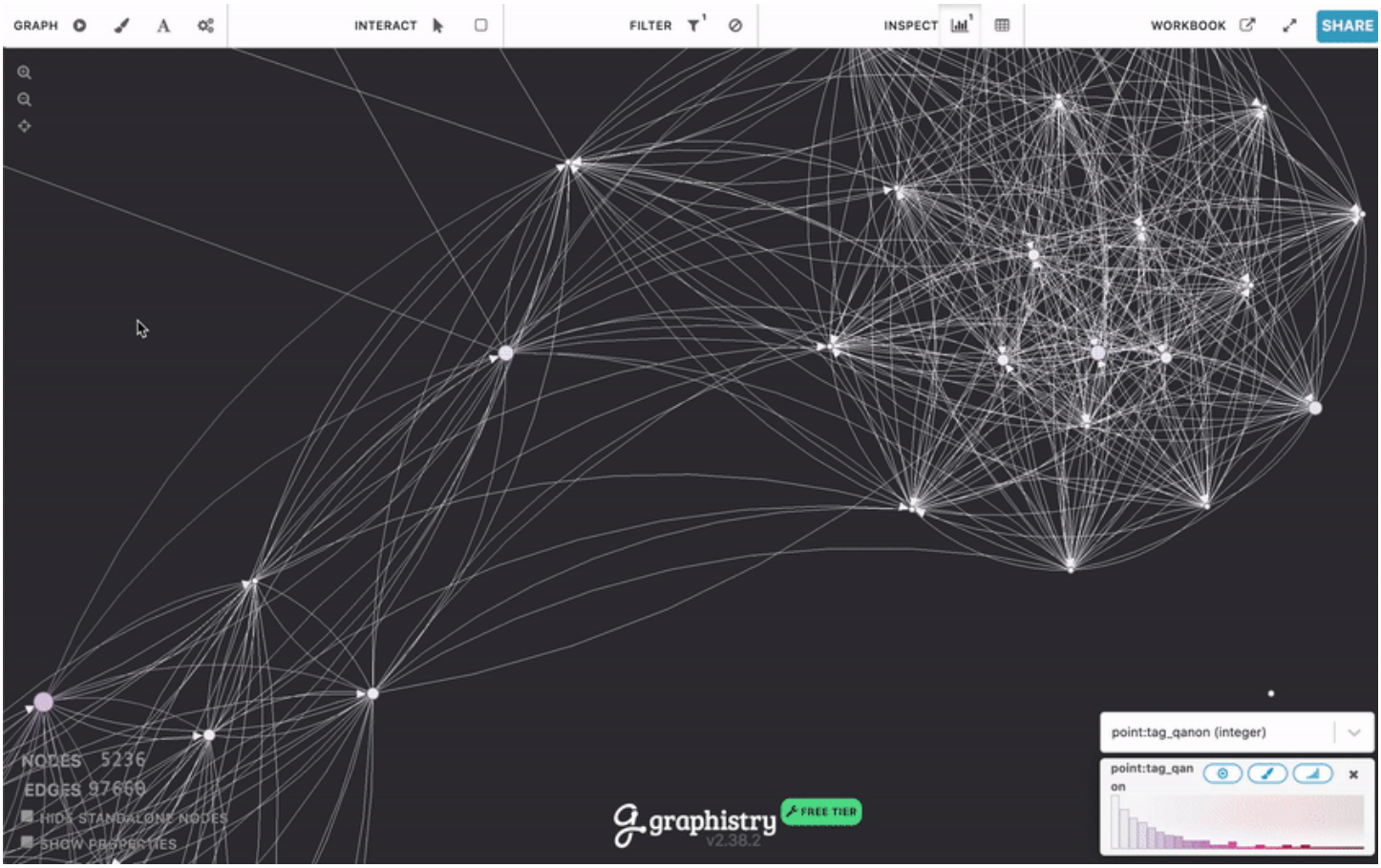


Graphistry

Graphistry 是一个视觉图形智能平台，它利用 GPU 加速来提供丰富的视觉体验。团队可以使用各种特征在 Graphistry 上进行协作，从无需代码浏览文件和数据库，到共享 Jupyter 笔记本和 Streamlit 控制面板，再到在自己的应用程序中使用嵌入式 API。

只需配置和启动 graph-app-kit 并修改几行代码，即可开始使用低编码的完全交互式控制面板。查看这篇博客文章，了解有关使用 Graphistry 和 Neptune 创建您的第一个控制面板的演练。你也可以试试 Neptune 演示PyGraphistry。 PyGraphistry 是适用于笔记本的 Python 可视化图表分析库。请查看本教程笔记本，了解 Neptune 演示 PyGraphistry 。

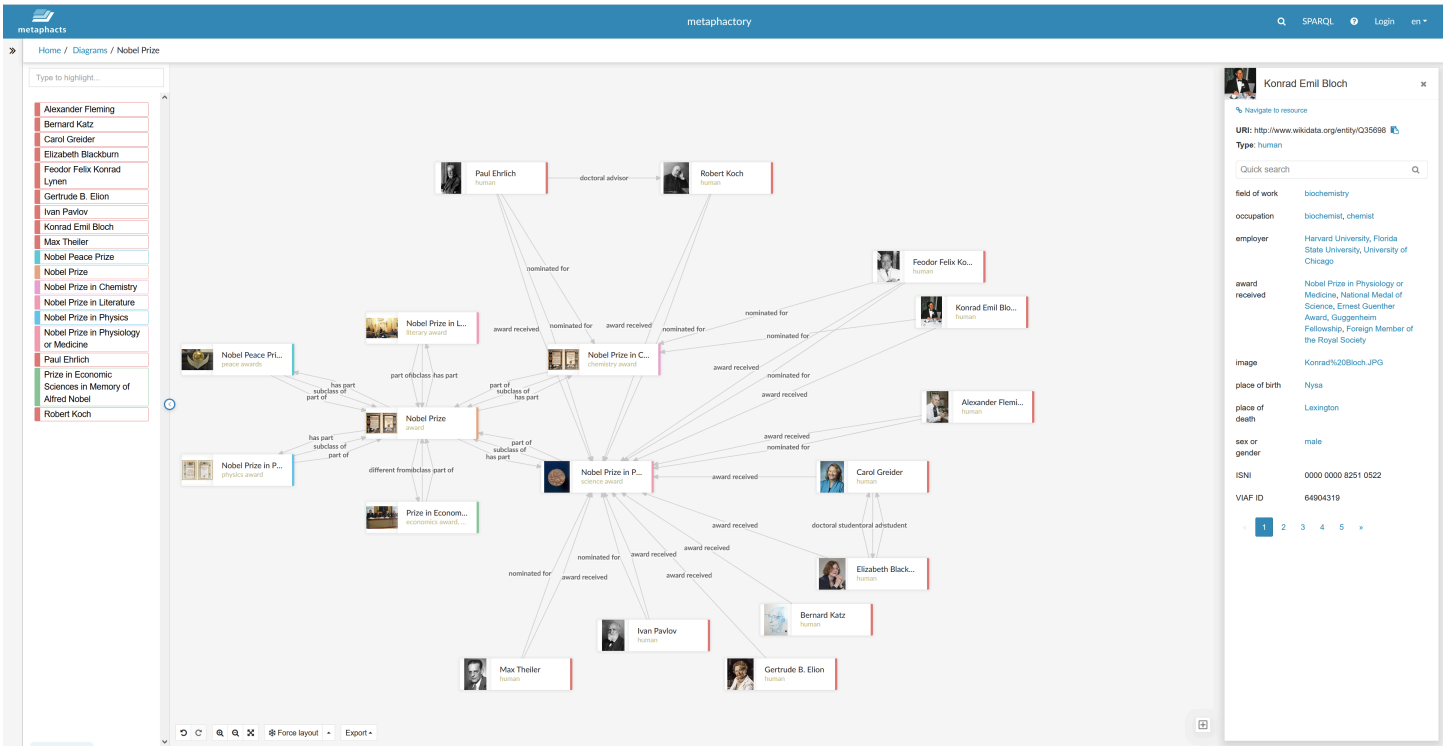
要开始使用，请访问 Marketplace AWS 中的 Graphistry。



metaphacts

metaphacts 提供了一个灵活的开放平台，用于描述和查询图形数据，并可视化知识图谱以及与之交互。借助 metaphactory，您可以使用 RDF 数据模型在 Neptune 中的知识图谱之上构建交互式 Web 应用程序，例如可视化和控制面板。Metaphactory 平台支持低代码开发体验，包括用于数据加载的用户界面、支持 OWL 和 SHACL 的可视化本体建模接口、SPARQL 查询 UI 和查询目录，以及用于图形探索、可视化、搜索和创作的一组丰富的 Web 组件。

以下是 metaphactory 可视化示例：



该平台专为工程、制造、制药、生命科学、金融、保险等领域设计并高效使用。要查看示例解决方案架构，请查看此博客文章。

要开始免费试用 metaphactory，请访问 AWS Marketplace。

G.V( )

G.V () 是一款功能强大的 Gremlin 集成开发环境 (IDE) 工具，适用于开发人员和数据分析师。使用它，您可以在 Neptune 中以交互方式查询、可视化和更新图形数据。G.V () 提供内置的 Gremlin 语言自动完成功能，该功能可在您键入查询时根据图表数据模型提供建议和文档。

还可以使用 Gremlin 查询调试特征来编写、调试、测试和深入分析图形遍历过程。

借助由 OpenAI 支持的自然语言处理，G.V () 可以根据文本提示生成与您的图形数据架构精确匹配的 Gremlin 查询，从而通过自然语言查询您的数据。

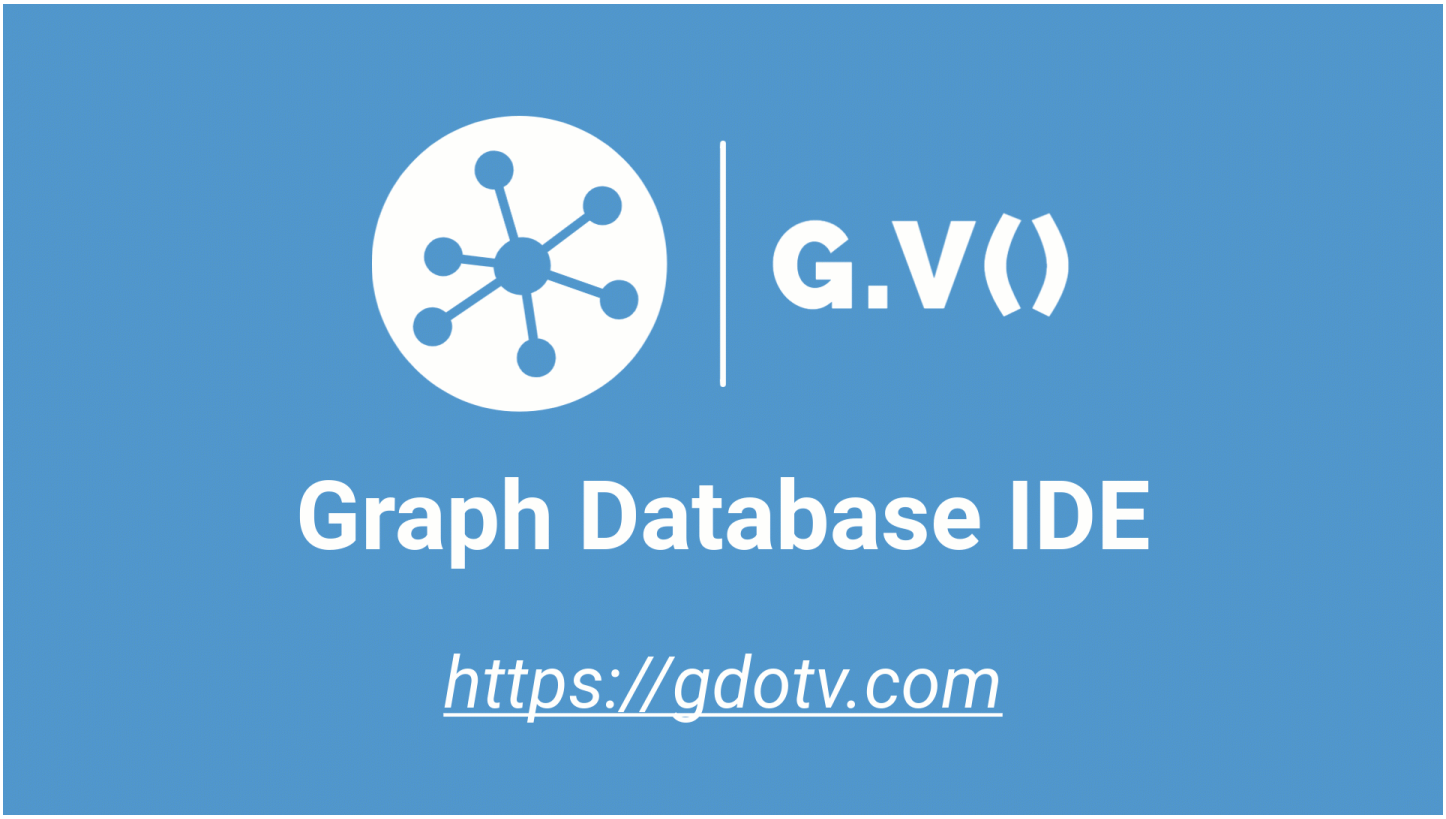
Graph Data Explorer 允许您浏览和修改图表，以快速构建新的图表结构并维护现有图表结构。

G.V () 为查询结果提供了多种可视化格式，可帮助您解释查询输出并以交互方式浏览图表。这些格式包括表、图形、JSON 和 Gremlin 控制台输出格式。

G.V () 与 Amazon Neptune 完全兼容，并提供了许多专门针对亚马逊 Neptune 的附加功能，例如慢速查询或审计日志见解以及 IAM 身份验证支持。要了解更多信息，请查看文档。

G.V () 在不断发展，并且每月都会收到新功能。要了解有关 G.V () 的更多信息，请访问 G.V () 网站开始免费试用。

请看下面的 G.V () 实际操作演示：



Linkurious

Linkurious 为技术和非技术用户以及各种用例提供不同的图形智能解决方案。

Linkurious Enterprise Explorer 是一款 off-the-shelf 图形可视化和分析软件，专为团队而设计，可以满足您的 day-to-day 活动需求，并帮助数据驱动的专业人员简单地做大事。它完全可配置且易于使用，可轻松适应您的需求，使新手或高级用户能够在 Nept AWS une 中快速可视化数据，无论数据的大小或复杂程度如何，都能直观地探索您的数据集，并在团队或企业层面进行无缝协作。

Linkurious Enterprise Watchtow er 利用了 Linkurious Enterprise Explorer 的强大功能，增加了创新的检测和案例管理功能，提供了由图形技术支持的集成检测和调查软件。一方面，它使您能够配置警报，利用海王星数据库和Neptune Analytics来自动显示复杂连接数据中的异常或模式。另一方面，它结合了案件管理和协作功能，可帮助团队高效管理其调查工作流程。

Ogma 是一个商业 JavaScript 库，可帮助您为应用程序开发强大、大规模的交互式图形可视化效果。它利用WebGL渲染和高性能布局，使用户能够在几秒钟内显示成千上万的节点和边缘并与之交互。它

还提供各种功能来定制您的应用程序并创建丰富的用户体验。最后，它还配备了全面的文档和工具，例如教程、数十个示例和一个交互式游乐场。

要开始使用，请申请 Linkurious Enterprise 或 Ogma 的 30 天免费试用。

从 Neptune 数据库集群中导出数据

从 Neptune 数据库集群导出数据有几种好方法：

- 对于少量数据，只需使用一个或多个查询的结果即可。
- 对于 RDF 数据，图形存储协议 (GSP) 可以简化导出过程。例如：

```
curl --request GET \
  'https://your-neptune-endpoint:port/sparql/gsp/?graph=http%3A//www.example.com/named/graph'
```

- 还有一个强大而灵活的开源工具，用于导出 Neptune 数据，即 `neptune-export`。以下各节介绍了此工具的特征以及如何使用它。

主题

使用 neptune-export

您可以通过两种不同的方式使用开源 neptune-export 工具：

- 作为 Neptune-Export 服务。 使用 Neptune-Export 服务从 Neptune 导出数据时，您可以通过 REST API 触发和监控导出任务。
- 作为 neptune-export Java 命令行实用程序。 要使用此命令行工具导出 Neptune 数据，您必须在可以访问 Neptune 数据库集群的环境中运行它。

Neptune-Export 服务和 neptune-export 命令行工具都将数据发布到 Amazon Simple Storage Service (Amazon S3)，数据是使用 Amazon S3 服务器端加密 (SSE-S3) 来加密的。

> **Note**
>
> 最佳做法是在所有 Amazon S3 桶上启用访问日志记录，这样您就可以审计对这些桶的所有访问。

如果您尝试从 Neptune 数据库集群导出数据，而该集群的数据在导出过程中发生变化，则无法保证导出数据的一致性。也就是说，如果您的集群在导出任务进行期间正在为写入流量提供服务，则导出的数据可能存在不一致之处。无论您是从集群中的主实例还是从一个或多个只读副本进行导出，都是如此。

为确保导出的数据一致，最好从数据库集群的克隆中导出。这既为导出工具提供了数据的静态版本，又可确保导出任务不会减慢原始数据库集群中的查询速度。

为了简化此操作，您可以在触发导出任务时表明要克隆源数据库集群。如果这样做，则导出过程会自动创建克隆，将其用于导出，然后在导出完成后将其删除。

使用 Neptune-Export 服务导出 Neptune 数据

您可以使用以下步骤，通过 Neptune-Export 服务将数据从 Neptune 数据库集群导出到 Amazon S3：

安装 Neptune-Export 服务

使用 AWS CloudFormation 模板创建堆栈：

安装 Neptune-Export 服务

1. 通过选择下表中的启动堆栈按钮之一，在 AWS CloudFormation 控制台中启动 AWS CloudFormation 堆栈：

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	Launch Stack
美国东部（俄亥俄州）	视图	在 Designer 中查看	Launch Stack
美国西部（北加利福尼亚）	视图	在 Designer 中查看	Launch Stack
美国西部（俄勒冈州）	视图	在 Designer 中查看	Launch Stack
加拿大（中部）	视图	在 Designer 中查看	Launch Stack
南美洲（圣保罗）	视图	在 Designer 中查看	Launch Stack
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	Launch Stack
欧洲地区（爱尔兰）	视图	在 Designer 中查看	Launch Stack
欧洲地区（伦敦）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
欧洲地区（巴黎）	视图	在 Designer 中查看	
欧洲地区（法兰克福）	视图	在 Designer 中查看	
中东（巴林）	视图	在 Designer 中查看	
中东（阿联酋）	视图	在 Designer 中查看	
以色列（特拉维夫）	视图	在 Designer 中查看	
非洲（开普敦）	视图	在 Designer 中查看	
亚太地区（香港）	视图	在 Designer 中查看	
亚太地区（东京）	视图	在 Designer 中查看	
亚太地区（首尔）	视图	在 Designer 中查看	
亚太地区（新加坡）	视图	在 Designer 中查看	
亚太地区（悉尼）	视图	在 Designer 中查看	
亚太地区（孟买）	视图	在 Designer 中查看	
中国（北京）	视图	在 Designer 中查看	
中国（宁夏）	视图	在 Designer 中查看	

区域	查看	在 Designer 中查看	发布
AWS GovCloud（美国西部）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国东部）	视图	在 Designer 中查看	Launch Stack

2. 在 Select Template 页面上，选择 Next。
3. 在模板的指定详细信息页面上，设置以下参数：

   - **VPC** – 设置 Neptune-Export 服务的最简单方法是将其安装在与 Neptune 数据库相同的 Amazon VPC 中。如果您想将其安装在单独的 VPC 中，则可以使用 VPC 对等连接在 Neptune 数据库集群的 VPC 和 Neptune-Export 服务 VPC 之间建立连接。
   - **Subnet1** – Neptune-Export 服务必须安装在您 VPC 的子网中，该子网允许从该子网到互联网的出站 IPv4 HTTPS 流量。这样，Neptune-Export 服务就可以调用 AWS Batch API 来创建和运行导出任务。

     如果您使用 Neptune 文档中创建数据库集群页面上的 CloudFormation 模板创建了 Neptune 集群，则可以使用该堆栈中的 `PrivateSubnet1` 和 `PrivateSubnet2` 输出来填充此参数和下一个参数。
   - **Subnet2** – VPC 中的第二个子网，允许从该子网到互联网的出站 IPv4 HTTPS 流量。
   - **EnableIAM** – 将其设置为 `true` 以使用 AWS Identity and Access Management (IAM) 保护 Neptune-Endpoint API。我们建议您这样做。

     如果您启用 IAM 身份验证，则必须对针对端点的所有 HTTPS 请求进行 `Sigv4` 签名。您可以使用诸如 awscurl 之类的工具来代表您签署请求。
   - **VPCOnly** – 将它设置为 `true` 将使导出端点成为“仅 VPC”，因此，您只能从安装了 Neptune-Export 服务的 VPC 内对其进行访问。这将 Neptune-Export API 限制为只能在该 VPC 内使用。

     建议您将 `VPCOnly` 设置为 `true`。
   - **NumOfFilesULimit** – 在 `ulimits` 容器属性中为 `nofile` 指定一个介于 10000 到 1 百万之间的值。默认值为 10000，除非您的图形包含大量唯一标签，否则我们建议保留默认值。
   - **PrivateDnsEnabled** （布尔值）– 指示是否将私有托管区与指定的 VPC 关联。默认值为 `true`。

在启用此标志的情况下创建 VPC 端点时，所有 API Gateway 流量都将通过 VPC 端点路由，并且公有 API Gateway 端点调用将被禁用。如果将 `PrivateDnsEnabled` 设置为 `false`，则启用公有 API Gateway 端点，但无法通过私有 DNS 端点连接 Neptune 导出服务。然后，您可以使用 VPC 端点的公有 DNS 端点来调用导出服务，详见此处。

4. 选择下一步。
5. 在选项页面上，选择下一步。
6. 在审核页面上，选中第一个复选框以确认 AWS CloudFormation 将创建 IAM 资源。选中第二个复选框以确认新堆栈的 `CAPABILITY_AUTO_EXPAND`。

> **Note**
>
> `CAPABILITY_AUTO_EXPAND` 明确确认在创建堆栈时将扩展宏，而无需事先审核。用户通常通过处理的模板创建更改集，以便在实际创建堆栈之前对宏所做的更改进行审核。有关更多信息，请参阅 AWS CloudFormation CreateStack API。

然后选择创建。

启用从 Neptune-Export 访问 Neptune

Neptune-Export 安装完成后，更新您的 Neptune VPC 安全组以允许从 Neptune-Export 进行访问。创建 Neptune-Export AWS CloudFormation 堆栈后，输出选项卡将包含一个 `NeptuneExportSecurityGroup` ID。更新 Neptune VPC 安全组以允许从该 Neptune 导出任务进行访问。

启用从基于 VPC 的 EC2 实例访问 Neptune-Export 端点

如果您将 Neptune-Export 端点设置为仅 VPC，则只能从安装了 Neptune-Export 服务的 VPC 内对其进行访问。要允许从 VPC 中的 Amazon EC2 实例进行连接（您可以从该实例进行 Neptune-Export API 调用），请将 AWS CloudFormation 堆栈创建的 `NeptuneExportSecurityGroup` 附加到该 Amazon EC2 示例。

使用 Neptune-Export API 运行 Neptune-Export 任务

AWS CloudFormation 堆栈的输出选项卡还包括 `NeptuneExportApiUri`。每当您向 Neptune-Export 端点发送请求时，请使用此 URI。

运行导出任务

- 请确保运行导出的用户或角色已被授予 `execute-api:Invoke` 权限。
- 如果您在安装 Neptune-Export 时在 AWS CloudFormation 堆栈中将 `EnableIAM` 参数设置为 `true`，则需要对针对 Neptune-Export API 的所有请求进行 `Sigv4` 签名。我们建议使用 awscurl 向 API 发出请求。此处的所有示例都假设已启用 IAM 身份验证。
- 如果您在安装 Neptune-Export 时在 AWS CloudFormation 堆栈中将 `VPCOnly` 参数设置为 `true`，则必须从 VPC 内调用 Neptune-Export API，通常是从位于 VPC 中的 Amazon EC2 实例调用。

要开始导出数据，请使用 `command` 和 `outputS3Path` 请求参数以及 `endpoint` 导出参数向 `NeptuneExportApiUri` 端点发送请求。

以下是从 Neptune 导出属性图数据并将其发布到 Amazon S3 的请求示例：

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-pg",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": { "endpoint": "(your Neptune endpoint DNS name)" }
      }'
```

同样，以下是将 RDF 数据从 Neptune 导出到 Amazon S3 的请求示例：

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-rdf",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": { "endpoint": "(your Neptune endpoint DNS name)" }
      }'
```

如果您省略 `command` 请求参数，则默认情况下，Neptune-Export 会尝试从 Neptune 导出属性图数据。

如果前面的命令成功运行，则输出将如下所示：

```
{
  "jobName": "neptune-export-abc12345-1589808577790",
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f"
}
```

监控您刚刚启动的导出任务

要监控正在运行的任务，请将其 jobID 附加到您的 NeptuneExportApiUri 之后，如下所示：

```
curl \
  (your NeptuneExportApiUri)(the job ID)
```

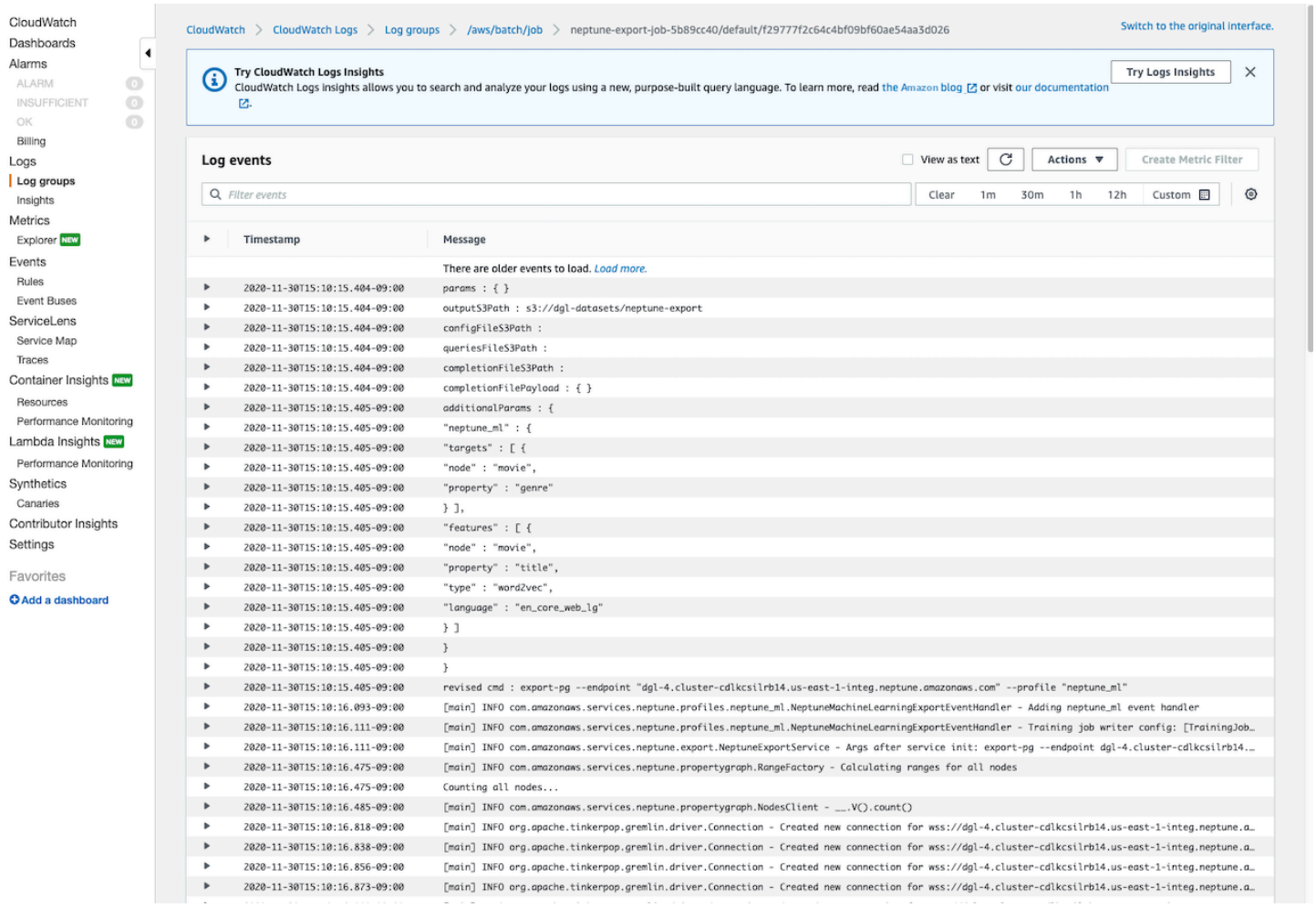
如果服务尚未启动导出任务，则响应将如下所示：

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "pending"
}
```

当您在导出任务开始后重复该命令时，响应将如下所示：

```
{
  "jobId": "c86258f7-a9c9-4f8c-8f4c-bbfe76d51c8f",
  "status": "running",
  "logs": "https://us-east-1.console.aws.amazon.com/cloudwatch/home?..."
}
```

如果您使用状态调用提供的 URI 在 CloudWatch Logs 中打开日志，则可以详细监控导出的进度：



取消正在运行的导出任务

使用 AWS Management Console 取消正在运行的导出任务

1. 打开AWS Batch控制台，地址：https://console.aws.amazon.com/batch/。
2. 选择 Jobs (作业)。
3. 根据要取消的正在运行的任务的 `jobID`，找到该任务。
4. 选择取消任务。

使用 Neptune 导出 API 取消正在运行的导出任务：

向附有 `jobID` 的 `NeptuneExportApiUri` 发送 `HTTP DELETE` 请求，如下所示：

```
curl -X DELETE \
  (your NeptuneExportApiUri) (the job ID)
```

使用 **neptune-export** 命令行工具从 Neptune 导出数据

您可以使用以下步骤，通过 `neptune-export` 命令行实用程序将数据从 Neptune 数据库集群导出到 Amazon S3：

使用 **neptune-export** 命令行实用程序的先决条件

开始之前

- 拥有 JDK 的版本 8 – 您需要安装 Java SE 开发工具包 (JDK) 的版本 8。
- 下载 neptune-export 实用程序 – 下载并安装 neptune-export.jar 文件。
- 确保 **`neptune-export`** 可以访问您的 Neptune VPC – 从可以访问您的 Neptune 数据库集群所在的 VPC 的位置运行 neptune-export。

  例如，您可以在 Neptune VPC 内的 Amazon EC2 实例上、在与 Neptune VPC 对等的独立 VPC 中或在单独的堡垒主机上运行它。
- 确保 VPC 安全组授予对 **`neptune-export`** 的访问权限 – 检查连接到 Neptune VPC 的 VPC 安全组是否允许从与 `neptune-export` 环境关联的 IP 地址或安全组访问您的数据库集群。
- 设置必要的 IAM 权限 – 如果您的数据库启用了 AWS Identity and Access Management (IAM) 数据库身份验证，请确保 `neptune-export` 运行所使用的角色与允许连接 Neptune 的 IAM policy 相关联。有关 Neptune 策略的一般信息，请参阅使用 IAM 策略。

  如果您想在查询请求中使用 `clusterId` 导出参数，则 `neptune-export` 运行所使用的角色需要以下 IAM 权限：

  - `rds:DescribeDBClusters`
  - `rds:DescribeDBInstances`
  - `rds:ListTagsForResource`

  如果要从克隆的集群中导出，则 `neptune-export` 运行所使用的角色需要以下 IAM 权限：

  - `rds:AddTagsToResource`
  - `rds:DescribeDBClusters`
  - `rds:DescribeDBInstances`
  - `rds:ListTagsForResource`
  - `rds:DescribeDBClusterParameters`
  - `rds:DescribeDBParameters`

- rds:ModifyDBParameterGroup
- rds:ModifyDBClusterParameterGroup
- rds:RestoreDBClusterToPointInTime
- rds:DeleteDBInstance
- rds:DeleteDBClusterParameterGroup
- rds:DeleteDBParameterGroup
- rds:DeleteDBCluster
- rds:CreateDBInstance
- rds:CreateDBClusterParameterGroup
- rds:CreateDBParameterGroup

  要将导出的数据发布到 Amazon S3，neptune-export 运行所使用的角色需要对 Amazon S3 位置具有以下 IAM 权限：

  - s3:PutObject
  - s3:PutObjectTagging
  - s3:GetObject

- 设置 **SERVICE_REGION** 环境变量 - 设置 SERVICE_REGION 环境变量以标识数据库集群所在的区域（有关区域标识符的列表，请参阅连接到 Neptune）。

运行 **neptune-export** 实用程序以启动导出操作

使用以下命令从命令行运行 neptune-export 并启动导出操作：

```
java -jar neptune-export.jar nesvc \
  --root-path (path to a local directory) \
  --json (the JSON file that defines the export)
```

该命令有两个参数：

开始导出时 neptune-export 的参数

- **--root-path** – 导出文件发布到 Amazon S3 之前写入的本地目录的路径。
- **--json** – 用于定义导出的 JSON 对象。

使用命令行实用程序的 **neptune-export** 命令示例

要直接从源数据库集群导出属性图数据，请执行以下操作：

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
            }
          }'
```

要直接从源数据库集群导出 RDF 数据，请执行以下操作：

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-rdf",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
            }
          }'
```

如果忽略 command 请求参数，则默认情况下，neptune-export 实用程序会从 Neptune 导出属性图数据。

要从数据库集群的克隆中导出，请执行以下操作：

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)",
              "cloneCluster" : true
            }
          }'
```

要使用 IAM 身份验证从数据库集群导出，请执行以下操作：

```
java -jar neptune-export.jar nesvc \
  --root-path /home/ec2-user/neptune-export \
  --json '{
            "command": "export-pg",
            "outputS3Path" : "s3://(your Amazon S3 bucket)/neptune-export",
            "params": {
              "endpoint" : "(your neptune DB cluster endpoint)"
              "useIamAuth" : true
            }
          }'
```

由 Neptune-Export 和 **neptune-export** 导出的文件

导出完成后，导出文件将发布到您指定的 Amazon S3 位置。将使用 Amazon S3 服务器端加密 (SSE-S3) 对发布到 Amazon S3 的所有文件进行加密。发布到 Amazon S3 的文件夹和文件会有所不同，具体取决于您是导出属性图还是 RDF 数据。如果您打开在其中发布文件的 Amazon S3 位置，则会看到以下内容：

导出文件在 Amazon S3 中的位置

- **nodes/** – 此文件夹包含逗号分隔值 (CSV) 或 JSON 格式的节点数据文件。

  在 Neptune 中，节点可以有一个或多个标签。具有不同单个标签（或多个标签的不同组合）的节点会写入不同的文件，这意味着没有单个文件包含具有不同标签组合的节点的数据。如果一个节点有多个标签，则这些标签在分配给文件之前会按字母顺序排序。
- **edges/** – 此文件夹包含逗号分隔值 (CSV) 或 JSON 格式的边缘数据文件。

  与节点文件一样，边缘数据会根据标签的组合写入不同的文件。为了进行模型训练，根据边缘的标签加上边缘的起始和结束节点的标签的组合，将边缘数据分配给不同的文件。
- **statements/** – 此文件夹包含 Turtle、N-Quads、N-Triples 或 JSON 格式的 RDF 数据文件。
- **config.json** – 此文件包含导出过程推断出的图形架构。
- **lastEventId.json** – 此文件包含数据库的 Neptune 流上最后一个事件的 `commitNum` 和 `opNum`。如果将 `includeLastEventId` 导出参数设置为 `true`，并且从中导出数据的数据库已启用 Neptune 流，则导出过程仅包括此文件。

用于控制 Neptune 导出过程的参数

无论您使用的是 Neptune-Export 服务还是 `neptune-export` 命令行实用程序，用于控制导出的参数基本相同。它们包含传递给 Neptune-Export 端点或命令行上的 `neptune-export` 的 JSON 对象。

传递到导出过程的对象最多有五个顶级字段：

```
-d '{
      "command" : "(either export-pg or export-rdf)",
      "outputS3Path" : "s3:/(your Amazon S3 bucket)/(path to the folder for exported
 data)",
      "jobsize" : "(for Neptune-Export service only)",
      "params" : { (a JSON object that contains export-process parameters) },
      "additionalParams": { (a JSON object that contains parameters for training
 configuration) }
   }'
```

目录

- namedGraph用于导出单个命名的图表

command 参数

command 顶级参数决定是导出属性图数据还是导出 RDF 数据。如果忽略 command 参数，则导出过程默认为导出属性图数据。

- **export-pg** – 导出属性图数据。
- **export-rdf** – 导出 RDF 数据。

outputS3Path 参数

outputS3Path 顶级参数是必需的，并且必须包含可将导出文件发布到的 Amazon S3 位置的 URI：

```
  "outputS3Path" : "s3://(your Amazon S3 bucket)/(path to output folder)"
```

该值必须以 s3:// 开头，后跟有效的桶名称以及（可选）桶内的文件夹路径。

jobSize 参数

jobSize 顶级参数仅用于 Neptune-Export 服务，不用于 neptune-export 命令行实用程序，并且是可选的。它允许您表征正在启动的导出任务的大小，这有助于确定专用于该任务的计算资源量及其最大并发级别。

```
  "jobsize" : "(one of four size descriptors)"
```

四个有效的大小描述符是：

- small – 最大并发度：8。适用于最大 10GB 的存储卷。
- medium – 最大并发度：32。适用于最大 100GB 的存储卷。
- large – 最大并发度：64。适用于超过 100GB 但小于 1TB 的存储卷。
- xlarge – 最大并发度：96。适用于超过 1TB 的存储卷。

默认情况下，在 Neptune-Export 服务上启动的导出作为 small 任务运行。

导出的性能不仅取决于 jobSize 设置，还取决于您要从中进行导出的数据库实例的数量、每个实例的大小以及任务的有效并发级别。

对于属性图导出，您可以使用 cloneClusterReplica计数 参数配置数据库实例的数量，也可以使用 并发 参数配置任务的有效并发级别。

`params` 对象

`params` 顶级参数是一个 JSON 对象，其中包含用于控制导出过程本身的参数，如导出 `params` 顶级 JSON 对象中的参数字段中所述。`params` 对象中的某些字段特定于属性图导出，有些字段特定于 RDF。

`additionalParams` 对象

`additionalParams` 顶级参数是一个 JSON 对象，其中包含可用于控制在导出数据后应用于数据的操作的参数。目前，`additionalParams` 仅用于导出 Neptune ML 的训练数据。

导出 **params** 顶级 JSON 对象中的参数字段

Neptune 导出 params JSON 对象允许您控制导出，包括导出数据的类型和格式。

导出参数 **params** 对象中可能的字段列表

下面列出了所有可能出现在 params 对象中的顶级字段。任何一个对象中都只显示这些字段的一个子集。

所有导出类型通用的字段列表

- cloneCluster
- cloneClusterInstanceType
- cloneClusterReplicaCount
- clusterId
- endpoint
- endpoints
- profile
- useIamAuth
- includeLastEventId

属性图导出的字段列表

- concurrency
- edgeLabels
- filter
- filterConfigFile
- gremlinFilter
- gremlinNodeFilter
- gremlinEdgeFilter
- format
- nodeLabels
- scope

RDF 导出的字段列表

- format
- rdfExportScope
- sparql
- namedGraph

所有导出类型通用的字段

params 中的 cloneCluster 字段

（可选）。默认值：false。

如果 cloneCluster 参数设置为 true，则导出过程将使用数据库集群的快速克隆：

```
"cloneCluster" : true
```

默认情况下，导出过程会从您使用 endpoint、endpoints 或 clusterId 参数指定的数据库集群中导出数据。但是，如果在导出过程中正在使用数据库集群，并且数据正在更改，则导出过程无法保证正在导出的数据的一致性。

为确保导出的数据一致，请改为使用 cloneCluster 参数从数据库集群的静态克隆中导出。

克隆的数据库集群与源数据库集群在同一 VPC 中创建，并继承源数据库集群的安全组、子网组和 IAM 数据库身份验证设置。导出完成后，Neptune 会删除克隆的数据库集群。

默认情况下，克隆的数据库集群由与源数据库集群中的主实例具有相同实例类型的单个实例组成。您可以通过使用 cloneClusterInstanceType 指定不同的实例类型，更改用于克隆的数据库集群的实例类型。

> **Note**
>
> 如果您不使用 cloneCluster 选项，而是直接从主数据库集群导出，则可能需要增加从中导出数据的实例的超时时间。对于大型数据集，应将超时设置为几个小时。

params 中的 cloneClusterInstanceType 字段

（可选）。

如果 cloneCluster 参数存在且设置为 true，则可以使用 cloneClusterInstanceType 参数指定用于克隆的数据库集群的实例类型：

默认情况下，克隆的数据库集群由与源数据库集群中的主实例具有相同实例类型的单个实例组成。

```
  "cloneClusterInstanceType" : "(for example, r5.12xlarge)"
```

params 中的 cloneClusterReplicaCount 字段

（可选）。

如果 cloneCluster 参数存在且设置为 true，则可以使用 cloneClusterReplicaCount 参数来指定在克隆的数据库集群中创建的只读副本实例的数量：

```
  "cloneClusterReplicaCount" : (for example, 3)
```

默认情况下，克隆的数据库集群由单个主实例组成。cloneClusterReplicaCount 参数允许您指定应额外创建多少只读副本实例。

params 中的 clusterId 字段

（可选）。

clusterId 参数指定要使用的数据库集群的 ID：

```
  "clusterId" : "(the ID of your DB cluster)"
```

如果您使用 clusterId 参数，则导出过程将使用该数据库集群中的所有可用实例来提取数据。

> **Note**
>
> endpoint、endpoints 和 clusterId 参数是互斥的。使用且仅使用其中一个。

params 中的 endpoint 字段

（可选）。

使用 endpoint 指定数据库集群中 Neptune 实例的端点，导出过程可以查询该端点以提取数据（请参阅终端节点连接）。这只是 DNS 名称，不包括协议或端口：

```
  "endpoint" : "(a DNS endpoint of your DB cluster)"
```

使用集群或实例端点，但不要使用主读取器端点。

> **Note**
>
> endpoint、endpoints 和 clusterId 参数是互斥的。使用且仅使用其中一个。

params 中的 **endpoints** 字段

（可选）。

使用 endpoints 指定数据库集群中端点的 JSON 数组，导出过程可以查询这些端点以提取数据（请参阅终端节点连接）。这些只是 DNS 名称，不包括协议或端口：

```
  "endpoints": [
    "(one endpoint in your DB cluster)",
    "(another endpoint in your DB cluster)",
    "(a third endpoint in your DB cluster)"
    ]
```

如果您的集群中有多个实例（一个主实例和一个或多个只读副本），则可以使用 endpoints 参数在这些端点列表间分配查询，从而提高导出性能。

> **Note**
>
> endpoint、endpoints 和 clusterId 参数是互斥的。使用且仅使用其中一个。

params 中的 **profile** 字段

（需要此字段以便为 Neptune ML 导出训练数据，除非 *additionalParams* 字段中存在 *neptune_ml* 字段）。

profile 参数为特定工作负载提供一组预配置的参数。目前，导出过程仅支持 neptune_ml 配置文件

如果要为 Neptune ML 导出训练数据，请向 params 对象添加以下参数：

```
  "profile" : "neptune_ml"
```

params 中的 useIamAuth 字段

（可选）。默认值：false。

如果您要从中导出数据的数据库启用了 IAM 身份验证，则必须包括 useIamAuth 参数（设置为 true）：

```
  "useIamAuth" : true
```

params 中的 includeLastEventId 字段

如果将 includeLastEventId 设置为 true，并且要从中导出数据的数据库启用了 Neptune Streams，则导出过程会将 lastEventId.json 文件写入您指定的导出位置。此文件包含流中最后一个事件的 commitNum 和 opNum。

```
  "includeLastEventId" : true
```

导出过程创建的克隆数据库会继承其父数据库的流设置。如果父级启用了流，则克隆同样会启用流。克隆上流的内容将反映创建克隆的时间点时父流的内容（包括相同的事件 ID）。

用于属性图导出的字段

params 中的 concurrency 字段

（可选）。默认值：4。

concurrency 参数指定导出过程应使用的并行查询数量：

```
  "concurrency" : (for example, 24)
```

一个好的指导原则是，在要从中导出数据的所有实例上，将并发级别设置为 vCPU 数量的两倍。例如，一个 r5.xlarge 实例有 4 个 vCPU。如果您要从包含 3 个 r5.xlarge 实例的集群中导出，则可以将并发级别设置为 24 (= 3 x 2 x 4)。

如果您使用的是 Neptune-Export 服务，则并发级别受到 jobSize 设置的限制。例如，小型任务支持的并发级别为 8。如果您尝试使用 concurrency 参数为小型任务指定并发级别 24，则有效级别将保持为 8。

如果您从克隆的集群中导出，则导出过程会根据克隆实例的大小和任务大小计算适当的并发级别。

params 中的 edgeLabels 字段

（可选）。

使用 edgeLabels 仅导出那些带有您指定的标签的边缘：

```
  "edgeLabels" : ["(a label)", "(another label"]
```

JSON 数组中的每个标签都必须是单个简单的标签。

scope 参数优先于 edgeLabels 参数，因此，如果 scope 值不包括边缘，则 edgeLabels 参数无效。

params 中的 filter 字段

（可选）。

使用 filter 指定仅应导出带有特定标签的节点和/或边缘，以及筛选为每个节点或边缘导出的属性。

filter 对象的一般结构，无论是内联对象还是筛选条件配置文件中的对象，如下所示：

```
  "filter" : {
    "nodes": [ (array of node label and properties objects) ],
    "edges": [ (array of edge definition an properties objects) ]
  }
```

- **nodes** – 包含节点和节点属性的 JSON 数组，格式如下：

  ```
    "nodes : [
      {
        "label": "(node label)",
        "properties": [ "(a property name)", "(another property name)", ( ... ) ]
      }
    ]
  ```

  - label – 节点的一个或多个属性图标签。

    取一个值，或者如果节点有多个标签，则取一个值数组。
  - properties – 包含要导出的节点属性的名称的数组。
- **edges** – 包含边缘定义的 JSON 数组，格式如下：

```
  "edges" : [
    {
      "label": "(edge label)",
      "properties": [ "(a property name)", "(another property name)", ( ... ) ]
    }
  ]
```

- `label` – 边缘的属性图标签。取单个值。
- `properties` – 包含要导出的边缘属性的名称的数组。

`params` 中的 `filterConfigFile` 字段

（可选）。

使用 `filterConfigFile` 指定包含筛选条件配置的 JSON 文件，其格式与 `filter` 参数采用的格式相同：

```
 "filterConfigFile" : "s3://(your Amazon S3 bucket)/neptune-export/(the name of the
JSON file)"
```

有关 `filterConfigFile` 文件的格式，请参阅filter。

`params` 中用于属性图数据的 `format` 字段

（可选）。默认值：`csv`（逗号分隔值）

`format` 参数指定导出的属性图数据的输出格式：

```
 "format" : (one of: csv, csvNoHeaders, json, neptuneStreamsJson)
```

- **`csv`** – 逗号分隔值 (CSV) 格式的输出，列标题根据 Gremlin 加载数据格式进行格式化。
- **`csvNoHeaders`** – CSV 格式的数据，没有列标题。
- **`json`** – JSON 格式的数据。
- **`neptuneStreamsJson`** – 使用 GREMLIN_JSON 更改序列化格式的 JSON 格式的数据。

`params` 中的 `gremlinFilter` 字段

（可选）。

`gremlinFilter` 参数允许您提供用于筛选节点和边缘的 Gremlin 代码段，例如 `has()` 步骤：

```
  "gremlinFilter" : (a Gremlin snippet)
```

字段名称和字符串值应用转义的双引号括起来。对于日期和时间，您可以使用 datetime 方法。

以下示例仅导出具有 date-created 属性且属性值晚于 2021-10-10 的节点和边缘：

```
  "gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
```

`params` 中的 `gremlinNodeFilter` 字段

（可选）。

`gremlinNodeFilter` 参数允许您提供用于筛选节点的 Gremlin 代码段，例如 `has()` 步骤：

```
  "gremlinNodeFilter" : (a Gremlin snippet)
```

字段名称和字符串值应用转义的双引号括起来。对于日期和时间，您可以使用 datetime 方法。

以下示例仅导出具有 `deleted` 布尔属性且属性值为 `true` 的节点：

```
  "gremlinNodeFilter" : "has(\"deleted\", true)"
```

`params` 中的 `gremlinEdgeFilter` 字段

（可选）。

`gremlinEdgeFilter` 参数允许您提供用于筛选边缘的 Gremlin 代码段，例如 `has()` 步骤：

```
  "gremlinEdgeFilter" : (a Gremlin snippet)
```

字段名称和字符串值应用转义的双引号括起来。对于日期和时间，您可以使用 datetime 方法。

以下示例仅导出具有 `strength` 数值属性且属性值为 5 的边缘：

```
  "gremlinEdgeFilter" : "has(\"strength\", 5)"
```

`params` 中的 `nodeLabels` 字段

（可选）。

使用 nodeLabels 仅导出那些带有您指定的标签的节点：

```
  "nodeLabels" : ["(a label)", "(another label"]
```

JSON 数组中的每个标签都必须是单个简单的标签。

scope 参数优先于 nodeLabels 参数，因此，如果 scope 值不包括节点，则 nodeLabels 参数无效。

params 中的 scope 字段

（可选）。默认值：all。

scope 参数指定是仅导出节点，仅导出边缘，还是同时导出节点和边缘：

```
  "scope" : (one of: nodes, edges, or all)
```

- nodes – 仅导出节点及其属性。
- edges – 仅导出边缘及其属性。
- all – 同时导出节点和边缘及其属性（默认）。

RDF 导出的字段

params 中用于 RDF 数据的 format 字段

（可选）。默认值：turtle

format 参数指定导出的 RDF 数据的输出格式：

```
  "format" : (one of: turtle, nquads, ntriples, neptuneStreamsJson)
```

- **turtle** – Turtle 格式的输出。
- **nquads** – N-Quads 格式的数据，没有列标题。
- **ntriples** – N-Triples 格式的数据。
- **neptuneStreamsJson** – 使用 SPARQL NQUADS 更改序列化格式的 JSON 格式的数据。

params 中的 rdfExportScope 字段

（可选）。默认值：graph。

rdfExportScope 参数指定 RDF 导出的范围：

```
  "rdfExportScope" : (one of: graph, edges, or query)
```

- graph – 导出所有 RDF 数据。
- edges – 仅导出表示边缘的三元组。
- query – 导出由使用 sparql 字段提供的 SPARQL 查询取回的数据。

params 中的 sparql 字段

（可选）。

sparql 参数允许您指定 SPARQL 查询来取回要导出的数据：

```
  "sparql" : (a SPARQL query)
```

如果您使用 sparql 字段提供查询，还必须将 rdfExportScope 字段设置为 query。

params 中的 namedGraph 字段

（可选）。

该namedGraph参数允许您指定 IRI 以将导出限制为单个命名图：

```
  "namedGraph" : (Named graph IRI)
```

该namedGraph参数只能在rdfExportScope字段设置为时使用graph。

筛选导出内容的示例

以下示例说明了筛选导出的数据的方法。

筛选属性图数据的导出

使用 **scope** 仅导出边缘的示例

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "scope": "edges"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

使用 **nodeLabels** 和 **edgeLabels** 仅导出带有特定标签的节点和边缘的示例

以下示例中的 nodeLabels 参数指定只应导出带有 Person 标签或 Post 标签的节点。edgeLabels 参数指定只应导出带有 likes 标签的边缘：

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "nodeLabels": ["Person", "Post"],
    "edgeLabels": ["likes"]
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

使用 **filter** 仅导出指定节点、边缘和属性的示例

此示例中的 filter 对象导出带有 type、code 和 desc 属性的 country 节点，还导出带有 dist 属性的 route 边缘。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
```

```
      "nodes": [
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ],
      "edges": [
        {
          "label": "route",
          "properties": [
            "dist"
          ]
        }
      ]
    }
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

使用 **gremlinFilter** 的示例。

此示例使用 gremlinFilter 以仅导出 2021-10-10 之后创建的节点和边缘（即其 created 属性值大于 2021-10-10）：

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinFilter" : "has(\"created\", gt(datetime(\"2021-10-10\")))"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

使用 **gremlinNodeFilter** 的示例。

此示例使用 gremlinNodeFilter 以仅导出已删除的节点（其布尔 deleted 属性的值为 true 的节点）：

```
{
```

```
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinNodeFilter" : "has(\"deleted\", true)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

使用 **gremlinEdgeFilter** 的示例。

此示例使用 gremlinEdgeFilter 以仅导出具有 strength 数值属性且属性值为 5 的边缘：

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "gremlinEdgeFilter" : "has(\"strength\", 5)"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

组合 **filter**、**gremlinNodeFilter**、**nodeLabels**、**edgeLabels** 和 **scope**

此示例中的 filter 对象导出：

- 带有其 type、code 和 desc 属性的 country 节点
- 带有其 code、icao 和 runways 属性的 airport 节点
- 带有其 dist 属性的 route 边缘

gremlinNodeFilter 参数筛选节点，以便仅导出具有 code 属性且属性值以 A 开头的节点。

nodeLabels 和 edgeLabels 参数进一步限制了输出，因此只输出 airport 节点和 route 边缘。

最后，scope 参数消除了输出中的边缘，从而在输出中只留下指定的 airport 节点。

```
{
  "command": "export-pg",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "filter": {
      "nodes": [
        {
```

```
          "label": "airport",
          "properties": [
            "code",
            "icao",
            "runways"
          ]
        },
        {
          "label": "country",
          "properties": [
            "type",
            "code",
            "desc"
          ]
        }
      ],
      "edges": [
        {
          "label": "route",
          "properties": [
            "dist"
          ]
        }
      ]
    },
    "gremlinNodeFilter": "has(\"code\", startingWith(\"A\"))",
    "nodeLabels": [
      "airport"
    ],
    "edgeLabels": [
      "route"
    ],
    "scope": "nodes"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

筛选 RDF 数据的导出

使用 **rdfExportScope** 和 **sparql** 导出特定边缘

此示例导出其谓词为 <http://kelvinlawrence.net/air-routes/objectProperty/route> 且其对象不是文本的三元组：

```
{
  "command": "export-rdf",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "rdfExportScope": "query",
    "sparql": "CONSTRUCT { ?s <http://kelvinlawrence.net/air-routes/objectProperty/
route> ?o } WHERE { ?s ?p ?o . FILTER(!isLiteral(?o)) }"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

namedGraph用于导出单个命名的图表

此示例导出属于命名图的三元组 < http://aws.amazon.com/neptune/vocab/v01/ DefaultNamedGraph **>** :

```
{
  "command": "export-rdf",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "rdfExportScope": "graph",
    "namedGraph": "http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph"
  },
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export"
}
```

对 Neptune 导出过程进行故障排除

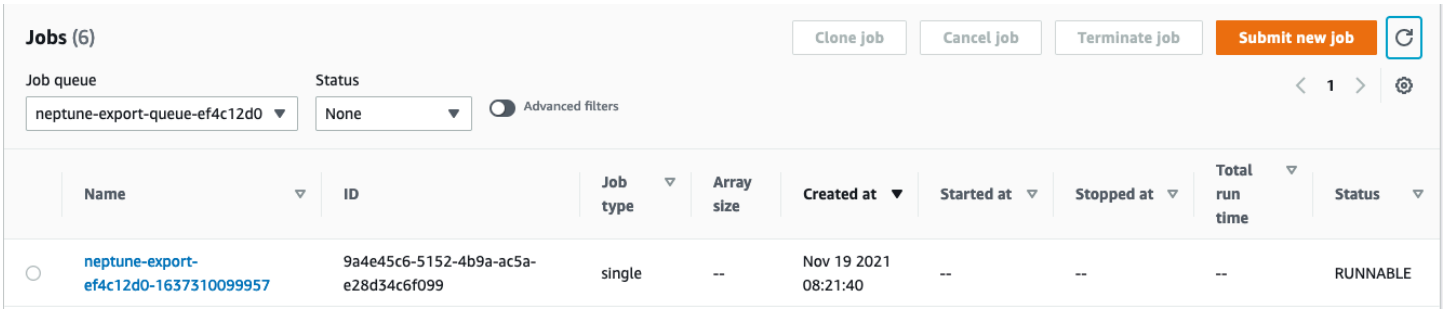
Amazon Neptune 导出流程使用 AWS Batch 来预调配导出 Neptune 数据所需的计算和存储资源。运行导出操作时，您可以使用 `logs` 字段中的链接访问导出任务的 CloudWatch 日志。

但是，执行导出的 AWS Batch 任务的 CloudWatch 日志仅在 AWS Batch 任务运行时才可用。如果 Neptune 导出报告导出处于待处理状态，则不会有日志链接可供您访问 CloudWatch 日志。如果导出任务保持 `pending` 状态的时间超过几分钟，则预调配底层 AWS Batch 资源可能会出现问题。

当导出任务离开待处理状态时，您可以按如下方式检查其状态：

检查 AWS Batch 任务的状态

1. 打开 AWS Batch 控制台，地址：https://console.aws.amazon.com/batch/。
2. 选择 neptune-export 任务队列。
3. 查找名称与您开始导出时由 Neptune 导出返回的 `jobName` 相匹配的任务。

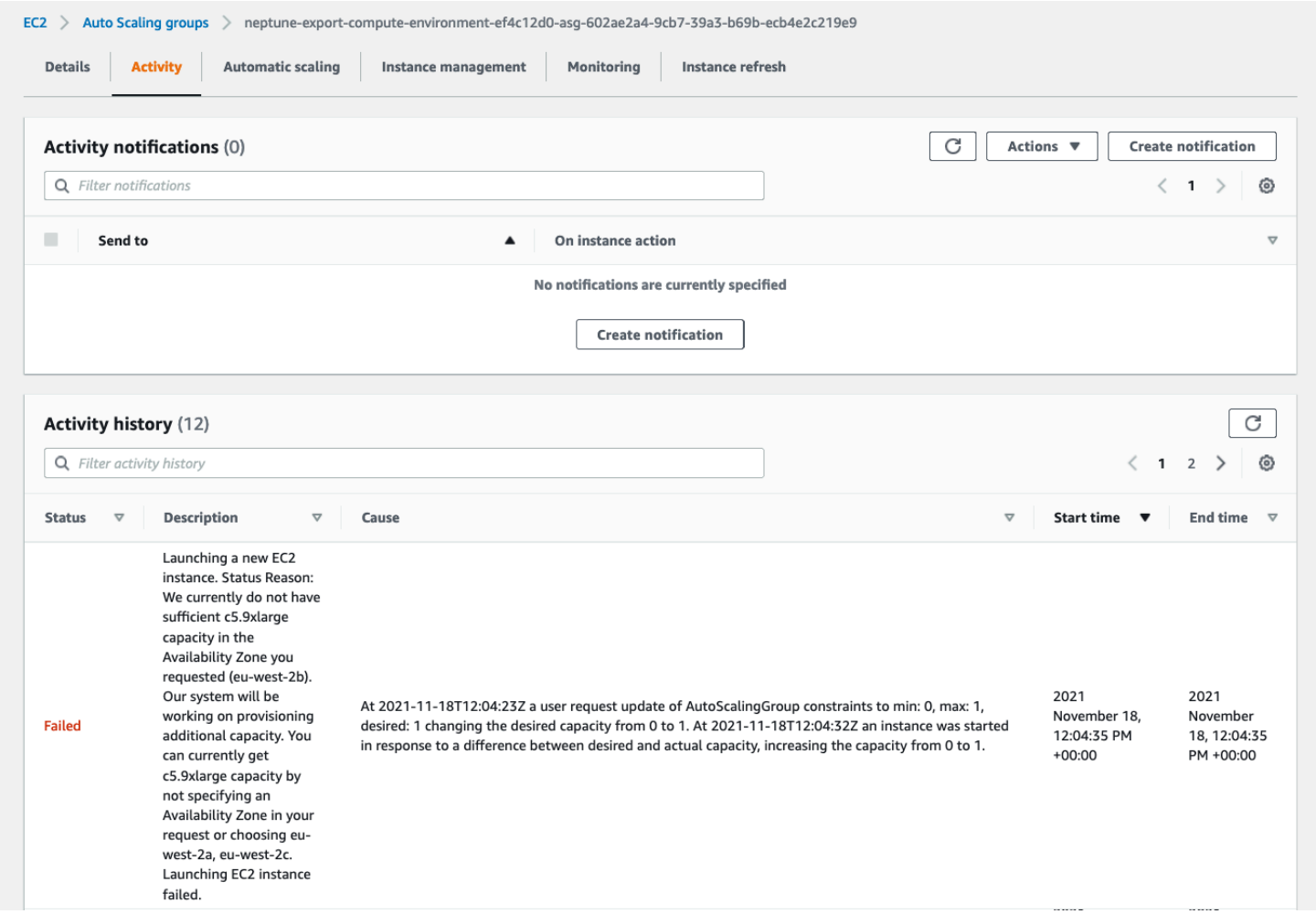


如果任务保持卡在 `RUNNABLE` 状态，则可能是因为网络或安全问题使容器实例无法加入底层的 Amazon Elastic Container Service (Amazon ECS) 集群。请参阅本支持文章中有关验证计算环境的网络和安全设置的部分。

您可以检查的另一件事是自动扩缩是否存在问题：

查看 AWS Batch 计算环境的 Amazon EC2 自动扩缩组

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 为 neptune-export 计算环境选择自动扩缩组。
3. 打开活动选项卡，并查看活动历史记录中是否存在未成功的事件。



Neptune 导出常见错误

`org.eclipse.rdf4j.query.QueryEvaluationException: Tag mismatch!`

如果 `export-rdf` 任务经常失败并引发 `Tag mismatch! QueryEvaluationException`，则 Neptune 实例的大小对于 Neptune 导出使用的大型、长时间运行的查询来说太小了。

您可以通过纵向扩展到更大的 Neptune 实例或将任务配置为从大型克隆集群中导出来避免出现此错误，如下所示：

```
'{
  "command": "export-rdf",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "cloneCluster": True,
    "cloneClusterInstanceType" : "r5.24xlarge"
```

```
  }
}'
```

管理您的 Amazon Neptune 数据库

此部分说明如何使用 AWS Management Console和 AWS CLI 来管理和维护 Neptune 数据库集群。

Neptune 在以复制拓扑形式连接的数据库服务器的集群上运行。因此，管理 Neptune 通常涉及将更改部署到多个服务器，以及确保所有 Neptune 副本与主服务器保持同步。

由于 Neptune 会随着数据增加透明地扩展底层存储，因此管理 Neptune 需要相对较少的磁盘存储管理。同样，由于 Neptune 自动执行持续备份，因此 Neptune 集群无需广泛的规划或用于执行备份的停机时间。

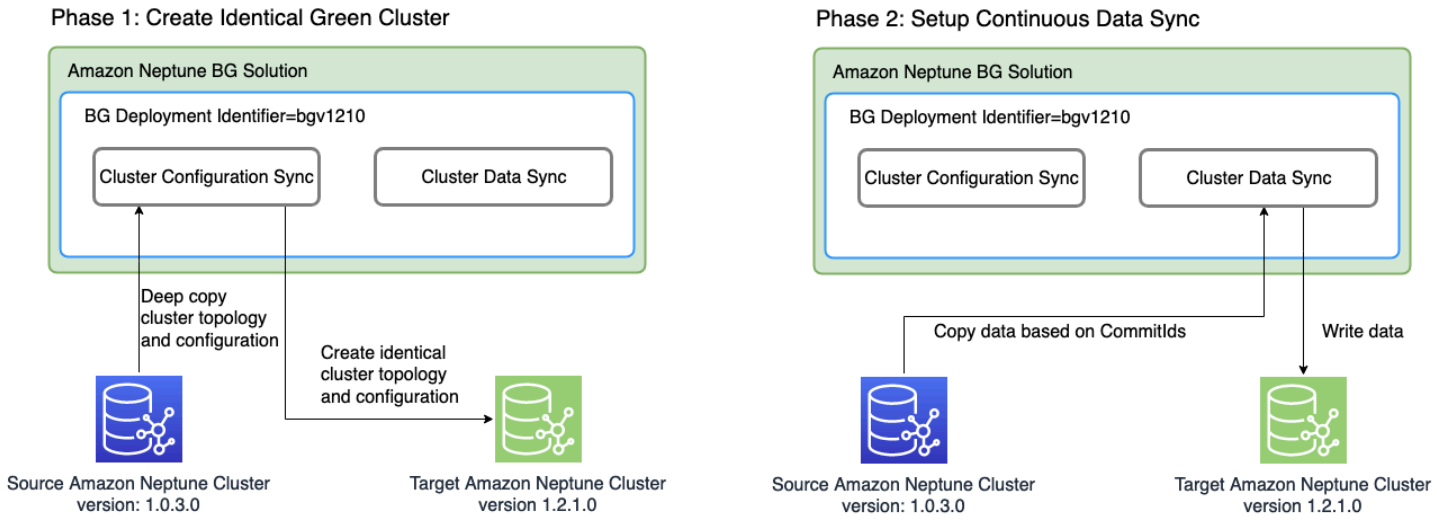
主题

使用 Neptune 蓝绿解决方案执行蓝绿更新

Amazon Neptune 引擎升级可能需要应用程序停机时间，因为在安装和验证更新时数据库不可用。无论它们是手动启动还是自动启动，都是如此。

Neptune 提供了蓝绿部署解决方案，您可以使用 AWS CloudFormation 堆栈运行该解决方案，从而大大减少此类停机时间。它创建一个与您的蓝色生产环境同步的绿色暂存环境。然后，您可以更新该暂存环境以执行次要或主要引擎版本升级、图形数据模型更改或操作系统更新，并测试结果。最后，您可以快速将其切换为生产环境，停机时间非常少。

Neptune 蓝绿解决方案分为两个阶段，如下图所示：



第 1 阶段创建一个与生产集群相同的绿色数据库集群

该解决方案使用唯一的蓝绿部署标识符创建数据库集群，其集群拓扑结构与生产集群相同。也就是说，它具有与生产（蓝色）数据库集群相同的数据库实例数量和大小、相同的参数组和所有配置，不同之处在于它已升级到您指定的目标引擎版本，该版本必须高于您的当前（蓝色）引擎版本。您可以为目标指定次要和主要引擎版本。如有必要，该解决方案将执行达到指定的目标引擎版本所需的任何中间升级。这个新集群变成了绿色暂存环境。

第 2 阶段设置连续数据同步

在完全准备好绿色环境后，该解决方案使 Neptune 流在源（蓝色）集群和目标（绿色）集群之间建立连续复制。当它们之间的复制差异达到零时，暂存环境就准备好进行测试了。此时，您必须暂停写入蓝色集群，以避免任何进一步的复制滞后。

您的目标引擎版本可能具有影响应用程序的新特征或依赖关系。检查引擎版本下的目标引擎版本页面和中间引擎版本页面，查看自当前引擎版本以来发生了什么变化。在将绿色集群提升到生产环境之前，最好在绿色集群上运行集成测试或手动验证应用程序。

在测试并验证了绿色集群中的更改后，只需将应用程序中的数据库端点从蓝色集群切换到绿色集群即可。

切换后，Neptune 蓝绿解决方案不会删除旧的蓝色生产环境。如果需要，您仍然可以访问它以进行其它验证和测试。标准账单费用适用于其实例，直到您将其删除。蓝绿解决方案还使用其它 AWS 服务，其费用按正常价格计费。清理部分介绍了有关在完成解决方案后将其删除的详细信息。

运行 Neptune 蓝绿堆栈的先决条件

在启动 Neptune 蓝绿堆栈之前：

- 请务必在生产（蓝色）集群上启用 Neptune 流。
- 蓝色集群中的所有实例都必须处于可用状态。您可以在 Neptune 控制台中或使用 describe-db-instances API 查看实例状态。
- 所有实例还必须与数据库集群参数组同步。
- Neptune 蓝绿解决方案要求您的蓝色集群所在的 VPC 中有一个 DynamoDB VPC 端点。请参阅使用 Amazon VPC 端点访问 DynamoDB。
- 选择在蓝色生产数据库集群上的写入工作负载尽可能轻的时候运行解决方案。例如，避免在将要进行批量加载或出于任何其它原因可能存在大量写入操作时运行解决方案。

使用 AWS CloudFormation 模板运行 Neptune 蓝绿解决方案

您可以使用 AWS CloudFormation 部署 Neptune 蓝绿解决方案。CloudFormation 模板在与您的蓝色源 Neptune 数据库相同的 VPC 中创建一个 Amazon EC2 实例，在那里安装解决方案并运行该实例。您可以在 CloudWatch 日志中监控其进度，如监控进度中所述。

您可以使用以下链接审核解决方案模板，也可以选择启动堆栈按钮在 AWS CloudFormation 控制台中启动该模板：

视图	在 Designer 中查看	Launch Stack

在控制台中，从窗口右上角的下拉列表中选择要运行解决方案的 AWS 区域。

按如下方式设置堆栈参数：

- **DeploymentID** – 每个 Neptune 蓝绿部署的唯一标识符。

  它用作绿色数据库集群标识符，也用作为在部署期间创建的新资源命名的前缀。
- **NeptuneSourceClusterId** – 要升级的蓝色数据库集群的标识符。
- **NeptuneTargetClusterVersion:** – 您要将蓝色数据库集群升级到的 Neptune 引擎版本。

  此值必须高于当前蓝色数据库集群的引擎版本。
- **DeploymentMode** – 表示这是新部署，还是尝试恢复先前的部署。当您使用与先前部署相同的 `DeploymentID` 时，请将 `DeploymentMode` 设置为 `resume`。

  有效值为：`new`（默认值）和 `resume`。
- **GraphQueryType** - 数据库的图形数据类型。

  有效值为：`propertygraph`（默认值）和 `rdf`。
- **SubnetId** – 来自您的蓝色数据库集群所在的同一 VPC 的子网 ID。（请参阅从同一 VPC 中的 Amazon EC2 实例连接到 Neptune 数据库集群）。

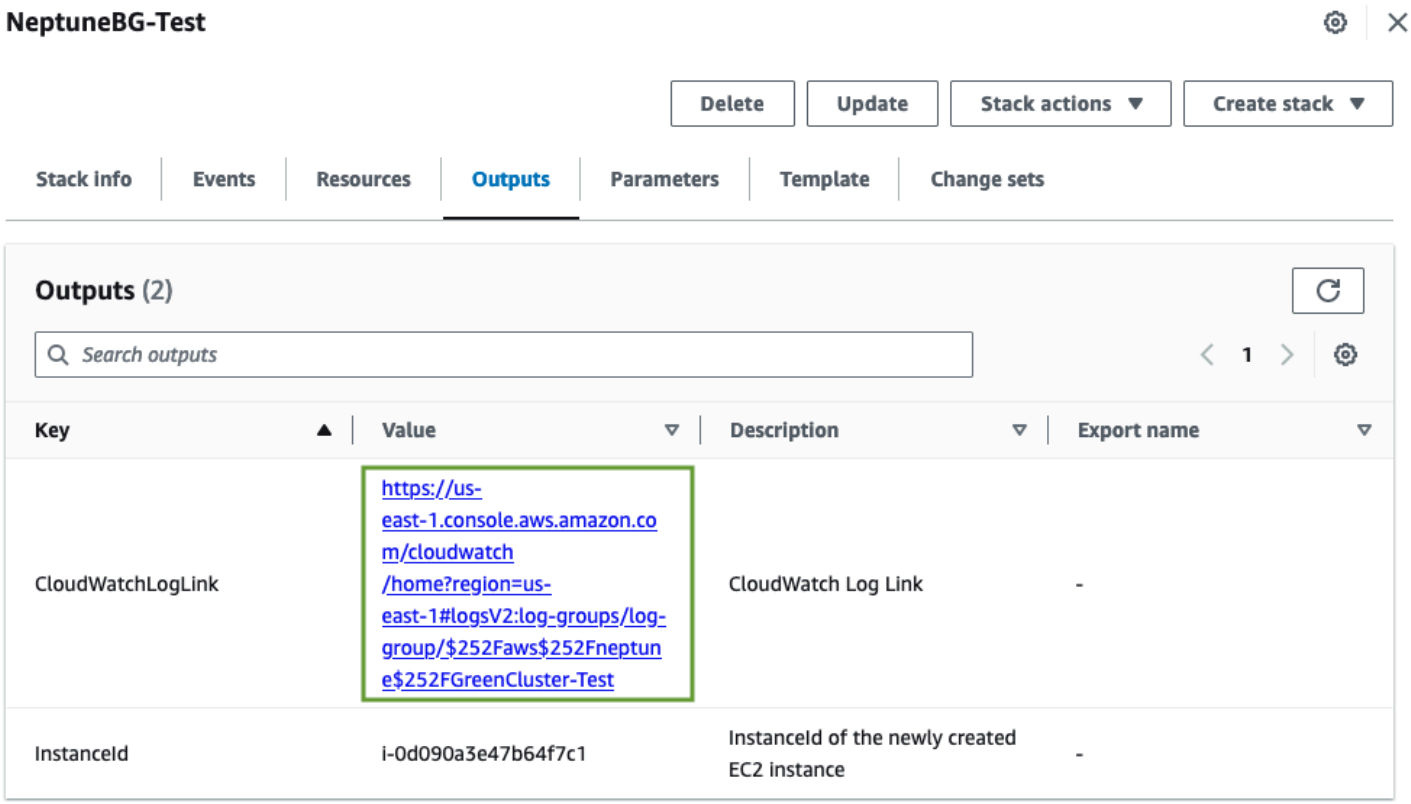
  如果您想通过 EC2 连接以 SSH 方式连接到该实例，请提供公有子网的 ID。
- **InstanceSecurityGroup** – Amazon EC2 实例的安全组。

  安全组必须有权访问您的蓝色数据库集群，并且您必须能够以 SSH 方式连接到该实例。请参阅使用 VPC 控制台创建安全组。

等到堆栈完成。一旦完成，就启动此解决方案。然后，您可以使用 CloudWatch 日志监控部署过程，如下一节所述。

监控 Neptune 蓝绿部署的进度

您可以通过前往 CloudWatch 控制台并查看 `/aws/neptune/(Neptune Blue/Green deployment ID)` CloudWatch 日志组中的日志来监控 Neptune 蓝绿解决方案的进度。您可以在解决方案的 AWS CloudFormation 堆栈的输出中找到 CloudWatch 日志的链接：



如果您提供公有子网作为堆栈参数，则还可以通过 SSH 方式连接到作为堆栈一部分创建的 Amazon EC2 实例，并参考 `/var/log/cloud-init-output.log` 中的日志。

该日志显示 Neptune 蓝绿解决方案所采取的操作，如以下屏幕截图所示：

```
============================================================

Neptune Blue Green Deployment Solution Version: 0.1.06012023

============================================================

Checking whether cluster with id = bg-06-01-14-20-29test-bg1-bgInt already exists.

BlueGreen deployment_mode = new

Didn't find any cluster with id bg-06-01-14-20-29test-bg1-bgInt

Cloned_cluster_id: bg-06-01-14-20-29test-bg1-bgInt

Replication_stack_name: bg-06-01-14-20-29test-bg1-bgInt-replication

DescribeDbClusters response for test-bg1-bgIntegTest-06-01-14-20-29: {'AllocatedStorage': 1,
'AvailabilityZones': ['us-east-1b', 'us-east-1c', 'us-east-1f'], 'BackupRetentionPeriod': 1,
'DBClusterIdentifier': 'test-bg1-bgintegtest-06-01-14-20-29', 'DBClusterParameterGroup': 'green-     -blue-
green-deployment-test-123456789012345-pg-tes710', 'DBSubnetGroup': 'default', 'Status': 'available',
'EarliestRestorableTime': datetime.datetime(2023, 6, 1, 8, 51, 23, 394000, tzinfo=tzlocal()), 'Endpoint':
'test-bg1-bgintegtest-06-01-14-20-29.cluster-critvszpmydm.us-east-1.neptune.amazonaws.com', 'ReaderEndpoint':
'test-bg1-bgintegtest-06-01-14-20-29.cluster-ro-critvszpmydm.us-east-1.neptune.amazonaws.com', 'MultiAZ':
False, 'Engine': 'neptune', 'EngineVersion': '1.2.0.0', 'LatestRestorableTime': datetime.datetime(2023, 6, 1,
8, 51, 23, 394000, tzinfo=tzlocal()), 'Port': 8182, 'MasterUsername': 'admin', 'PreferredBackupWindow':
'06:33-07:03', 'PreferredMaintenanceWindow': 'fri:09:44-fri:10:14', 'ReadReplicaIdentifiers': [],
'DBClusterMembers': [{'DBInstanceIdentifier': 'test-bg1-bgintegtest-06-01-14-20-29i-1', 'IsClusterWriter':
True, 'DBClusterParameterGroupStatus': 'in-sync', 'PromotionTier': 1}], 'VpcSecurityGroups':
```

日志消息显示蓝色和绿色集群之间的同步状态：

```
DDB checkpoint {'S': '1'}, {'S': '6'}

DDB Checkpoint: {'checkpointSubSequenceNumber': {'S': '6'}, 'lastUpdateTime': {'N': '1685611142127'},
'leaseOwner': {'S': 'nobody'}, 'checkpoint': {'S': '1'}, 'leaseKey': {'S': 'bg-anl   -234567899-replication'}}

Time difference for last checkpoint and last stream event: 5841351

Stream eventId difference for last replication checkpoint and last stream event on the Source cluster: 0:0

Found region : us-east-1

Cloudwatch Log Url for blue green solution is https://us-east-1.console.aws.amazon.com/cloudwatch
/home?region=us-east-1#logsV2:log-groups/log-group/aws/neptune/bg

Cloudwatch dashboard url for replication is https://console.aws.amazon.com/cloudwatch/home?region=us-
east-1#dashboards:name=neptune-stream-poller-bg-an   -234567899-replication

Replication poller lambda arn is arn:aws:lambda:us-east-1:451235071234:function:bg-an   -234567899-replic-
NeptuneStreamPollerLambd-B6V1ytULgmSP. Look for CW log the poller lambda for more troubleshooting.

Stream Last EventId {'commitNum': 1, 'opNum': 6} on cluster : database-d61852469-t        -experiment.cluster-
critvszpmydm.us-east-1.neptune.amazonaws.com:8182

DDB checkpoint {'S': '1'}, {'S': '6'}

DDB Checkpoint: {'checkpointSubSequenceNumber': {'S': '6'}, 'lastUpdateTime': {'N': '1685611207245'},
'leaseOwner': {'S': 'nobody'}, 'checkpoint': {'S': '1'}, 'leaseKey': {'S': 'bg-ankig-234567899-replication'}}
```

同步过程通过计算蓝色集群上的最新数据流 `eventID` 与由 Neptune 至 Neptune 复制堆栈创建的 DynamoDB 检查点表中存在的复制检查点之间的差异来检查复制滞后。使用这些消息，您可以监控当前的复制差异。

从生产蓝色集群切换到更新后的绿色集群

在将绿色集群提升到生产环境之前，请确保蓝色和绿色集群之间的提交差异为零，然后禁用蓝色集群的所有写入流量。在将数据库端点切换到绿色集群的同时继续写入蓝色集群，可能会因向两个集群写入部分数据而导致数据损坏。您可能还不需要禁用读取流量。

如果您已在源（蓝色）集群上启用了 IAM 身份验证，请务必更新应用程序中使用的所有 IAM policy 以指向绿色集群（有关此类策略的示例，请参阅此无限制访问策略）。

禁用写入流量后，等待复制完成，然后在绿色集群上（但不在蓝色集群上）启用写入流量。同时将读取流量从蓝色集群切换到绿色集群。

在 Neptune 蓝绿解决方案完成后进行清理

将暂存（绿色）集群升级到生产环境后，清理由 Neptune 蓝绿解决方案创建的资源：

- 删除为运行该解决方案而创建的 Amazon EC2 实例。
- 删除基于 Neptune 流的复制的 AWS CloudFormation 模板，这些模板使绿色集群与蓝色集群保持同步。主模板具有您之前提供的堆栈名称，一个模板名称由部署 ID 后跟“-replication”组成：即 *(DeploymentID)*-replication。

删除 AWS CloudFormation 模板并不会删除集群本身。确认绿色集群按预期运行后，您可以选择在手动删除蓝色集群之前拍摄快照。

Neptune 蓝绿解决方案最佳实践

- 在将绿色集群切换到生产环境之前，值得彻底验证其是否运行正常。检查数据的一致性和数据库的配置。某些新的引擎版本可能还需要客户端升级。请在升级之前查看引擎版本说明。在生产环境中开始蓝绿升级之前，值得在开发、测试和预生产环境中对所有这些进行测试。
- 最好在维护时段内执行从蓝色服务器到绿色服务器的切换。
- 为了确保升级和同步后一切正常运行，在删除原始集群之前，值得将其保留一段时间。如果出现不可预见的问题，它可能会很有用。
- 运行 Neptune 蓝绿解决方案时，请避免繁重的写入操作，例如批量加载，因为它们可能会导致复制滞后，从而导致大量的停机时间。理想情况下，从关闭对蓝色集群的写入到为绿色集群开启写入只有很短的时间。

对 Neptune 蓝绿解决方案进行故障排除

Neptune 蓝绿解决方案引发的错误

- **Cluster with id = *(blue_green_deployment_id)* already exists** – 具有标识符 *(blue_green_deployment_id)* 的现有集群。

  如果集群是在之前的 Neptune 蓝绿运行中创建的，请提供新的部署 ID 或将部署模式设置为 `resume`。
- **Streams should be enabled on the source Cluster for Blue Green Deployment** – 在蓝色（源）集群上启用 Neptune 流。
- **No Bulkload should be in progress on source cluster: *(cluster_id)*** – 如果 Neptune 蓝绿解决方案识别出持续的批量加载，它就会终止。

  这是为了确保同步过程能够赶上正在进行的写入操作。在启动 Neptune 蓝绿解决方案之前，请避免或取消任何正在进行的批量加载任务。

- **`Blue Green deployment requires instances to be in sync with db cluster parameter group`** – 对集群参数组的任何更改都应在整个数据库集群中同步。请参阅Amazon Neptune 参数组。
- **`Invalid target engine version for Blue Green Deployment`** – 目标引擎版本必须在Amazon Neptune 的引擎版本中列为活动的版本，并且必须高于源（蓝色）集群的当前引擎版本。

创建具有 Neptune 权限的 IAM 用户

要访问 Neptune 控制台来创建和管理 Neptune 数据库集群，您需要创建一个具有所有必要权限的 IAM 用户。

第一步是为 Neptune 创建服务相关角色策略：

为 Amazon Neptune 创建服务相关角色策略

1. 登录AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。
3. 在策略页面上，选择创建策略。
4. 在创建策略页面上，选择 JSON 选项卡并复制以下服务相关角色策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "iam:CreateServiceLinkedRole",
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
AWSServiceRoleForRDS",
      "Condition": {
        "StringLike": {
            "iam:AWSServiceName":"rds.amazonaws.com"
        }
      }
    }
  ]
}
```

5. 选择下一步：标签，然后在添加标签页面上选择下一步：审核。
6. 在审核策略页面上，将新策略命名为“NeptuneServiceLinked”。

有关服务相关角色的更多信息，请参阅 对 Neptune 使用服务相关角色。

创建具有所有必要权限的新 IAM 用户

接下来，创建附有相应托管式策略的新 IAM 用户，该策略将授予您需要的权限，以及您创建的服务相关角色策略（此处命名为 `NeptuneServiceLinked`）：

1. 登录AWS Management Console，然后通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择用户，然后在用户页面上选择添加用户。
3. 在添加用户页面上，输入新 IAM 用户的名称，为 AWS 凭证类型选择访问密钥 - 编程访问权限，然后选择下一步：权限。
4. 在设置权限页面上的筛选策略框中，键入“Neptune”。现在，从列出的策略中选择以下内容：

   - NeptuneFullAccess
   - NeptuneConsoleFullAccess
   - NeptuneServiceLinked（假设您之前创建的服务相关角色策略就是这样命名的）。
5. 接下来，在筛选策略框中键入“VPC”以代替“Neptune”。从列出的策略中选择 AmazonVPCFullAccess。
6. 选择下一步：标签，然后在添加标签页面中选择下一步：审核。
7. 在审核页面中，检查以下所有策略现在是否已附加到您的新用户：

   - NeptuneFullAccess
   - NeptuneConsoleFullAccess
   - NeptuneServiceLinked
   - AmazonVPCFullAccess

   然后，选择创建用户。
8. 最后，下载并保存新用户的访问密钥 ID 和秘密访问密钥。

要在 Amazon Simple Storage Service (Amazon S3) 之类的其它服务上进行互操作，您需要添加更多权限和信任关系。

Amazon Neptune 参数组

通过使用数据库参数组中的参数，在 Amazon Neptune 中管理数据库配置。参数组就像是引擎配置值的容器，这些值可应用于一个或多个数据库实例。

有两种类型的参数组，即数据库集群参数组和数据库参数组：

- 数据库参数组在实例级应用，通常与 Neptune 图形引擎的设置（如 neptune_query_timeout 参数）关联。
- 数据库集群参数组 应用到集群中的每个实例，通常具有更广泛的设置。每个 Neptune 集群与一个数据库集群参数组关联。该集群中的每个数据库实例继承数据库集群参数组中包含的引擎配置值。

您在数据库集群参数组中修改的任意配置值将覆盖数据库参数组中的默认值。如果您在数据库参数组中编辑对应的值，则这些值将覆盖数据库集群参数组中的设置。

如果在创建数据库实例时未指定自定义数据库参数组，则将使用默认数据库参数组。无法修改默认数据库参数组的参数设置。相反，要更改默认参数设置，则必须创建新的数据库参数组。并非所有数据库引擎参数都可在您创建的数据库参数组中进行更改。

参数组是在与不同的 Neptune 引擎版本兼容的系列中创建的。默认参数组系列是 neptune1，它与 1.2.0.0 之前的所有引擎版本兼容。从版本：1.2.0.0（2022 年 7 月 21 日）开始，必须改用 neptune1.2 参数组系列。这意味着，升级到 1.2.0.0 或更高版本时，必须先在 neptune1.2 系列中重新创建所有自定义参数组，以便在升级时可以附加它们。

一些 Neptune 参数是静态的，而另一些则是动态的。区别如下所示：

静态参数

- 静态参数是指只有在数据库实例重启后才会生效的参数。换句话说，当您更改静态参数并保存实例数据库参数组时，必须手动重启数据库实例才能使参数更改生效。当前，所有 Neptune 实例级参数（在数据库参数组而不是数据库集群参数组中）都是静态的。
- 更改集群级静态参数并保存数据库集群参数组时，参数更改将在手动重启集群中的每个数据库实例后生效。

动态参数

- 动态参数是在参数组中更新参数后几乎立即生效的参数。换句话说，更新动态参数后，无需重启数据库实例即可使参数更改生效。

- 在所有数据库实例上应用动态集群参数更改预计会有一些轻微的延迟。
- 更新后的动态参数值不适用于当前正在运行的请求，而仅适用于更改发生后提交的请求。
- 当您更改动态集群级参数时，默认情况下，参数更改将立即应用于数据库集群，而无需任何重启。要将参数更改推迟到重启集群中的数据库实例之后，可以使用 AWS CLI 将 `ApplyMethod` 设置为 `pending-reboot` 以进行参数更改。

目前，除以下新集群参数外，所有参数均为静态参数：

- `neptune_enable_slow_query_log`（集群级）
- `neptune_slow_query_log_threshold`（集群级）

以下是您在使用数据库参数组中的参数时应了解的几个要点：

- 在数据库参数组内设置参数不恰当可能会产生意外的不利影响，包括性能降低和系统不稳定。修改数据库参数时应始终保持谨慎，且在修改数据库参数组前备份数据。先对测试数据库实例试用参数组设置更改，然后再将这些更改应用于生产数据库实例。
- 当您更改与数据库实例关联的数据库参数组时，必须在数据库实例使用新的数据库参数组之前手动重启实例。

> **Note**
>
> 在版本：1.2.0.0（2022 年 7 月 21 日）之前，每当主（写入器）实例重启时，数据库集群中的所有只读副本实例都会自动重启。
> 从版本：1.2.0.0（2022 年 7 月 21 日）开始，重启主实例不会导致任何副本实例重启。这意味着，如果您要更改集群级的参数，则必须分别重启每个实例才能获得参数更改。

编辑数据库集群参数组或数据库参数组

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 选择导航窗格中的 Parameter groups (参数组)。
3. 选择要编辑的数据库参数组的 Name (名称) 链接。

   （可选）选择创建参数组以创建新的集群参数组并创建新的组。然后选择新参数组的名称。

> **Important**
>
> 如果只有默认数据库集群参数组，则此步骤为必需，因为默认数据库集群参数组无法修改。

4. 搜索参数，然后单击 “名称” 列旁边的 “值” 字段。
5. 输入允许的值，然后选择值字段旁边的复选框。
6. 选择保存更改。
7. 如果您要更改数据库集群参数，请重启 Neptune 集群中的每个数据库实例；如果您要更改数据库实例参数，则重启一个或多个特定实例。

创建数据库参数组或数据库集群参数组

您可以轻松使用 Neptune 控制台创建一个新的参数组：

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 选择左侧导航窗格中的 Parameter groups (参数组)。
3. 选择 Create DB parameter group (创建数据库参数组)。

   创建数据库参数组页面将显示。
4. 在参数组系列列表中，选择 neptune1；或者，如果您的目标是引擎版本 1.2.0.0 或更高版本，请选择 neptune1.2。
5. 在类型列表中，选择数据库参数组或数据库集群参数组。
6. 在组名框中，键入新数据库参数组的名称。
7. 在描述框中，键入新数据库参数组的描述。
8. 选择创建。

您也可以使用 AWS CLI 创建新的参数组：

```
aws neptune create-db-parameter-group \
  --db-parameter-group-name (a name for the new DB parameter group) \
  --db-parameter-group-family (either neptune1 or neptune1.2, depending on the engine
 version) \
```

```
  --description (a description for the new DB parameter group)
```

Amazon Neptune 参数

您可以使用参数组中的参数来管理 Amazon Neptune 中的数据库配置。以下参数可用于配置 Neptune 数据库：

集群级别的参数

- neptune_enable_audit_log
- neptune_enable_slow_query_log
- neptune_slow_query_log_threshold
- neptune_lab_mode
- neptune_query_timeout
- neptune_streams
- neptune_streams_expiry_days
- neptune_lookup_cache
- neptune_autoscaling_config
- neptune_ml_iam_role
- neptune_ml_endpoint

实例级参数

- neptune_dfe_query_engine
- neptune_query_timeout
- neptune_result_cache

弃用的参数

- neptune_enforce_ssl

neptune_enable_audit_log（集群级参数）

此参数切换 Neptune 的审计日志记录。

允许的值为 0（禁用）和 1（启用）。默认值为 0。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

您可以将审计日志发布到 Amazon CloudWatch，如使用 CLI 将 Neptune 审核日志发布到日志 CloudWatch 中所述。

neptune_enable_slow_query_log（集群级参数）

使用此参数启用或禁用 Neptune 的慢速查询日志记录特征。

这是一个动态参数，意味着更改其值不需要或导致数据库集群重启。

允许的值包括：

- **info** – 启用慢速查询日志记录并记录可能导致性能下降的选定属性。
- **debug** – 启用慢速查询日志记录并记录查询运行的所有可用属性。
- **disable** – 禁用慢速查询日志记录。

默认值为 disable。

您可以将慢速查询日志发布到 Amazon CloudWatch，如使用 CLI 将 Neptune 慢速查询日志发布到日志 CloudWatch 中所述。

neptune_slow_query_log_threshold（集群级参数）

此参数以毫秒为单位指定执行时间阈值，超过该阈值后，查询被视为慢速查询。如果启用慢速查询日志记录，则运行时间超过此阈值的查询将与其某些属性一起记录。

默认值为 5000 毫秒（5 秒）。

这是一个动态参数，意味着更改其值不需要或导致数据库集群重启。

neptune_lab_mode（集群级参数）

设置此项后，该参数将启用 Neptune 的特定实验性特征。有关当前可用的实验性功能，请参阅Neptune 实验室模式。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

要启用或禁用实验性功能，请在此参数中包括 *(feature name)*=enabled 或 *(feature name)*=disabled。可以通过用逗号分隔多个功能来启用或禁用它们，如下所示：

(feature #1 name)=enabled,*(feature #2 name)*=enabled

默认情况下，通常禁用实验室模式特征。一个例外是 DFEQueryEngine 特征，从 Neptune 引擎版本 1.0.5.0 开始，该特征默认处于启用状态，可以与查询提示 (DFEQueryEngine=viaQueryHint) 一起使用。从 Neptune 引擎版本 1.1.1.0 开始，DFE 引擎不再处于实验室模式，现在使用实例的数据库参数组中的 neptune_dfe_query_engine 实例参数进行控制。

neptune_query_timeout（集群级参数）

指定图形查询的特定超时持续时间（以毫秒为单位）。

允许值的范围为 10 到 2,147,483,647 (2^{31} - 1)。默认值为 120,000（2 分钟）。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

> **Note**
>
> 如果您将查询超时值设置得过高，可能会产生意想不到的成本，尤其是在无服务器实例上。如果没有合理的超时设置，您可能会无意中发出一个持续运行时间比预期长得多的查询，从而产生您意想不到的成本。在运行查询时可以纵向扩展到昂贵的大型实例类型的无服务器实例上尤其如此。
>
> 您可以使用查询超时值来避免此类意外开支，该值可以容纳您的大多数查询，并且只会导致长时间运行的查询意外超时。

neptune_streams（集群级参数）

启用或禁用 Neptune Streams。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

允许的值为 0（默认禁用）和 1（启用）。

neptune_streams_expiry_days（集群级参数）

指定服务器删除流记录之前的天数。

允许的值为 1 到 90（含）。默认为 7。

该参数是在引擎版本 1.2.0.0 中引入的。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

neptune_lookup_cache（集群级参数）

在 R5d 实例上禁用或重新启用 Neptune 查找缓存。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

允许的值包括 enabled 和 disabled。默认值为 disabled，但是每当在数据库集群中创建 R5d 实例时，neptune_lookup_cache 参数都会自动设置为 enabled，并在该实例上创建查找缓存。

neptune_autoscaling_config（集群级参数）

为 Neptune 自动扩缩创建和管理的只读副本实例设置配置参数。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

使用您设置为 neptune_autoscaling_config 参数值的 JSON 字符串，可以指定：

- Neptune 自动扩缩为它所创建的所有新的只读副本实例使用的实例类型。
- 分配给这些只读副本的维护时段。
- 要与所有新的只读副本关联的标签。

JSON 字符串的结构如下所示：

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

请注意，字符串中的引号必须全部使用反斜杠字符 (\) 进行转义。

neptune_autoscaling_config 参数中未指定的三个配置设置中的任何一个都是从数据库集群的主写入器实例的配置中复制的。

neptune_ml_iam_role（集群级参数）

指定 Neptune ML 中使用的 IAM 角色 ARN。该值可以是任何有效的 IAM 角色 ARN。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

您可以在图形上为机器学习指定默认 IAM 角色 ARN。

neptune_ml_endpoint（集群级参数）

指定用于 Neptune ML 的端点。该值可以是任何有效的 SageMaker 端点名称。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

您可以为图形上的机器学习指定默认 SageMaker 端点。

neptune_dfe_query_engine（实例级参数）

从 Neptune 引擎版本 1.1.1.0 开始，此数据库实例参数用于控制 DFE 查询引擎的使用方式。允许的值如下所示：

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

- **enabled** – 使得尽可能使用 DFE 引擎，但 `useDFE` 查询提示存在并设置为 `false` 除外。
- **viaQueryHint**（默认）– 使得 DFE 引擎仅用于显式包含设置为 `true` 的 `useDFE` 查询提示的查询。

如果未显式设置此参数，则实例启动时将使用默认值 `viaQueryHint`。

> **Note**
>
> 无论如何设置此参数，所有 openCypher 查询都由 DFE 引擎执行。

在版本 1.1.1.0 之前，这是一个实验室模式参数，而不是数据库实例参数。

neptune_query_timeout（实例级参数）

此数据库实例参数为一个实例指定图形查询的超时时间，以毫秒为单位。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

允许值的范围为 10 到 2,147,483,647 (2^{31} - 1)。默认值为 120,000（2 分钟）。

> **Note**
>
> 如果您将查询超时值设置得过高，可能会产生意想不到的成本，尤其是在无服务器实例上。如果没有合理的超时设置，您可能会无意中发出一个持续运行时间比预期长得多的查询，从而产生您意想不到的成本。在运行查询时可以纵向扩展到昂贵的大型实例类型的无服务器实例上尤其如此。
>
> 您可以使用查询超时值来避免此类意外开支，该值可以容纳您的大多数查询，并且只会导致长时间运行的查询意外超时。

neptune_result_cache（实例级参数）

neptune_result_cache – 此数据库实例参数启用或禁用 缓存查询结果。

此参数是静态的，这意味着在任何实例重启之前，对其所做的更改不会在该实例上生效。

允许的值为 0（禁用，默认值）和 1（启用）。

neptune_enforce_ssl（弃用的集群级参数）

（已弃用）过去有些区域允许 HTTP 连接到 Neptune，当此参数设置为 1 时，用于强制所有连接使用 HTTPS。但是，此参数不再相关，因为 Neptune 现在在所有区域中仅接受 HTTPS 连接。

使用 AWS Management Console启动 Neptune 数据库集群

启动新 Neptune 数据库集群的最简单方法是使用可为您创建所有必需资源的 AWS CloudFormation 模板，如创建数据库集群中所述。

如果您愿意，也可以使用 Neptune 控制台手动启动新的数据库集群，如此处所述。

在访问 Neptune 控制台创建 Neptune 集群之前，请创建一个具有执行此操作所需权限的 IAM 用户，如创建具有 Neptune 权限的 IAM 用户中所述。

然后，以该 IAM 用户的身份登录 AWS Management Console，并按照以下步骤创建新的数据库集群：

使用控制台启动 Neptune 数据库集群

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 导航到数据库页面并选择创建数据库，这将打开创建数据库页面。
3. 在引擎选项下，引擎类型为 `neptune`，您可以选择特定的引擎版本或接受默认版本。
4. 在设置下，输入新数据库集群的名称或接受此处提供的默认名称。此名称用于实例的端点地址，并且必须满足以下限制：
   - 它必须包含 1 到 63 个字母数字字符或连字符。
   - 它的第一个字符必须是字母。
   - 它不能以连字符结束或包含两个连续连字符。
   - 在给定 AWS 区域的 AWS 账户中，它必须在所有数据库实例中唯一。
5. 在模板下，选择生产或开发和测试。
6. 在数据库实例大小下，选择实例大小。这将决定新数据库集群的主写入实例的处理和内存容量。

   如果您选择了生产模板，则只能从列出的可用内存优化型类中进行选择，但如果您选择了开发和测试，则也可以从更经济的可突发类中进行选择（有关可突发类的讨论，请参阅T3 可突增实例）。

   > **Note**
   >
   > 从 Neptune 引擎版本 1.1.0.0 开始，Neptune 不再支持 R4 实例类型。

7. 在可用性和耐久性下，您可以选择是否启用多可用区部署。默认情况下，生产模板启用多可用区部署，而开发和测试模板则不启用多可用区部署。如果启用了多可用区部署，Neptune 会定位您在不同可用区 (AZ) 中创建的只读副本实例，以提高可用性。

8. 在连接下，从可用选项中选择要托管新数据库集群的虚拟私有云 (VPC)。如果您希望 Neptune 为您创建 VPC，则可以选择创建新的 VPC。您必须在同一 VPC 中创建 Amazon EC2 实例才能访问 Neptune 实例（有关更多信息，请参阅每个 Amazon Neptune 数据库集群都位于 Amazon VPC 中）。请注意，创建数据库集群后，将无法更改 VPC。

   如果需要，可以在其它连接配置下为集群进一步配置连接：

   a. 在子网组下，您可以选择要用于新数据库集群的 Neptune 数据库子网组。如果您的 VPC 还没有任何子网组，Neptune 将为您创建一个数据库子网组（请参阅每个 Amazon Neptune 数据库集群都位于 Amazon VPC 中）。
   b. 在 VPC 安全组下，选择一个或多个现有 VPC 安全组来保护对新数据库集群的网络访问，或者，如果您希望 Neptune 为您创建一个安全组，请选择新建，然后为新 VPC 安全组提供名称（请参阅使用 VPC 控制台创建安全组）。
   c. 在数据库端口下，输入数据库将用于应用程序连接的 TCP/IP 端口。Neptune 使用端口号 8182 作为默认值。

9. 如果希望 Neptune 在 Neptune Workbench 中为您创建 Jupyter 笔记本，请在笔记本配置下选择创建笔记本（请参阅使用 Neptune 图形笔记本快速入门和使用 Neptune Workbench 托管 Neptune 笔记本）。然后，您可以选择应如何配置新的笔记本：

   a. 在笔记本实例类型下，从笔记本的可用实例类中进行选择。
   b. 在笔记本名称下，输入笔记本的名称。
   c. 如果需要，还可以在描述 - 可选下输入笔记本的描述。
   d. 在 IAM 角色名称下，选择让 Neptune 为笔记本创建 IAM 角色，然后输入新角色的名称，或者进行选择，以从可用角色中选择现有 IAM 角色。
   e. 最后，选择您的笔记本是直接连接到互联网，通过 Amazon SageMaker 连接到互联网，还是通过带有 NAT 网关的 VPC 连接到互联网。有关更多信息，请参阅将笔记本实例连接到 VPC 中的资源。

10. 在标签下，您最多可以将 50 个标签与新数据库集群关联。
11. 在其它配置下，您可以为新数据库集群进行更多设置（在许多情况下，您可以跳过这些设置并暂时接受默认值）：

选项	您能做什么
数据库实例标识符	您可以为集群的写入器实例提供名称。如果不这样做，则使用基于集群名称的默认标识符。如果这样做，则指定对于您的 AWS 账户在当

选项	您能做什么
	前区域中拥有的所有数据库实例唯一的名称。数据库实例标识符不区分大小写，但它以全小写形式存储。
数据库集群参数组	选择数据库集群参数组，以定义集群中所有数据库实例的默认配置。除非您另行选择，否则 Neptune 将使用默认数据库集群参数组。有关参数组的更多信息，请参阅 Amazon Neptune 参数组。
数据库参数组	选择一个数据库参数组来定义集群中主数据库实例的配置。除非您另行选择，否则 Neptune 将使用默认参数组。有关参数组的更多信息，请参阅 参数组。
IAM 数据库身份验证	如果您选中启用 IAM 数据库身份验证，则对数据库的所有访问都将使用 AWS Identity and Access Management (IAM) 进行身份验证。 **Important** 这要求您使用 AWS 签名版本 4 签名来签署所有请求。有关更多信息，请参阅亚马逊 Neptune 中的 AWS Identity and Access Management (IAM) 概述。
Failover priority (故障转移优先级)	为失效转移选择 No preference 或优先级层。如果选择某个层，而该层中存在争用，则会选择与主实例大小相等的副本。
备份保留期	选择 Neptune 应保留此数据库实例的自动备份的时间长度（1 到 35 天）。您只能对备份保留期内的某个时间执行时间点恢复 (PITR)。

选项	您能做什么
将标签复制到快照	（默认启用）此选项将与您的数据库集群关联的所有标签复制到该集群的所有快照中。
启用加密	（默认启用）此选项会导致静态加密数据库集群中的数据。 如果这样做，请选择用来保护用于加密此数据库卷的密钥的主密钥。您可以选择默认的 `aws/rds` 密钥，从您账户的主密钥中进行选择，或输入来自其它账户的密钥的 ARN。您可以在 IAM 控制台的加密密钥选项卡上创建新的主加密密钥。有关更多信息，请参阅静态加密 Neptune 资源。
审计日志	如果您想将数据库集群的审计日志发布到 CloudWatch Logs，请勾选此选项。
启用自动次要版本升级	（默认启用）此选项使您的数据库集群在新的次要引擎版本发布后自动升级到此次要版本。自动升级会在数据库的维护时段进行。请参阅使用 `AutoMinorVersionUpgrade` 。
维护窗口	您可以选择您希望对数据库集群进行待处理修改的特定时段，例如对数据库实例类的更改或自动引擎补丁。任何此类维护操作都将在选定的时间段内开始和完成。如果您不选择时段，Neptune 会任意分配维护时段。
启用删除保护	（默认启用）删除保护机制会阻止删除您的数据库集群。您必须显式禁用它才能删除数据库集群。

12. 选择创建数据库以启动您的新 Neptune 数据库集群及其主实例。

    在 Amazon Neptune 控制台中，新数据库集群显示在数据库的列表中。数据库集群的状态为 Creating (正在创建)，直到完成创建并可供使用。当状态更改为 Available (可用) 时，您可连接到

数据库集群的主实例。根据所分配的数据库实例类和存储的不同，新实例可能需要数分钟时间才能变得可用。

要查看新创建的集群，请选择 Neptune 控制台中的数据库视图。

> **Note**
>
> 如果您使用 AWS Management Console删除数据库集群中的所有 Neptune 数据库实例，则控制台会自动删除数据库集群本身。如果您使用的是 AWS CLI 或 SDK，则必须在删除最后一个实例后手动删除数据库集群。

记下集群端点值。连接到 Neptune 数据库集群需要此值。

停止和启动 Amazon Neptune 数据库集群

停止和启动 Amazon Neptune 集群可以帮助您控制开发和测试环境的成本。您可以暂时停止集群中的所有数据库实例，而不是每次使用集群时设置和停用所有数据库实例。

主题

停止和启动 Neptune 数据库集群概述

在不需要使用 Neptune 集群的期间，您可以同时停止该集群中的所有实例。您可以在需要使用时再次启动集群。启动和停止简化了用于开发、测试或不需要持续可用性的类似活动的集群的设置和停用过程。您可以在 AWS Management Console中通过单个操作完成此操作，而不考虑集群中的实例数量。

在停止数据库集群后，您只需在指定的保留时段内为集群存储、手动快照和自动备份存储付费。您无需为任何数据库实例小时数付费。

七天后，Neptune 自动重新启动数据库集群，以确保其赶得上任何所需的维护更新。

为了最大限度减少具有较少负载的 Neptune 集群的费用，您可以停止集群，而不是删除它的所有只读副本。对于具有超过一个或两个实例的集群，只能使用 AWS CLI 或 Neptune API 频繁删除和重新创建数据库实例，而且删除也可能难以按正确顺序执行。例如，您必须在删除主实例之前删除所有只读副本，以避免激活故障转移机制。

如果您需要将数据库集群保持运行状态，但想要减少容量，请不要使用启动和停止。如果您的集群成本太高或不太繁忙，您可以删除一个或多个数据库实例，或者将数据库实例更改为使用较小的实例类，但您不能停止单个数据库实例。

停止 Neptune 数据库集群

当您一段时间不使用它时，您可以停止正在运行的 Neptune 数据库集群，然后在需要时重新启动它。在停止集群后，您需要在指定的保留时段内为集群存储、手动快照和自动备份存储付费，但不需要为数据库实例小时数付费。

停止操作会先停止所有集群的只读副本实例，然后停止主实例，以避免激活故障转移机制。

使用 AWS Management Console 停止数据库集群

使用 AWS Management Console 停止 Neptune 集群

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)，然后选择集群。您可以从该页面中执行停止操作，或者导航到要停止的数据库集群的详细信息页面。
3. 在 Actions (操作) 中，选择 Stop (停止)。

使用 AWS CLI 停止数据库集群

要使用 AWS CLI 停止数据库实例，请调用 stop-db-cluster 命令，同时使用 `--db-cluster-identifier` 参数标识要停止的数据库集群。

Example

```
aws neptune stop-db-cluster --db-cluster-identifier mydbcluster
```

使用 Neptune 管理 API 停止数据库集群

要使用 Neptune 管理 API 停止数据库实例，请调用 StopDBCluster API 命令，并使用 `DBClusterIdentifier` 参数标识要停止的数据库集群。

数据库集群停止后会发生什么

- 您可以从快照还原它（请参阅从数据库集群快照还原）。
- 您无法修改该数据库集群或其任何数据库实例的配置。
- 您无法从集群中添加或删除数据库实例。
- 如果集群仍有任何关联的数据库实例，则您无法删除此集群。
- 通常，您必须重新启动已停止的数据库集群才能执行大多数管理操作。
- 一旦再次启动停止的集群，Neptune 会将任何计划的维护应用于此集群。请记住，七天后，Neptune 自动重新启动停止的集群，以使其维护状态不会落后太多。
- Neptune 不会对已停止的数据库集群执行任何自动备份，因为在集群停止时无法更改基础数据。
- Neptune 不会延长数据库集群停止时的备份保留期。

启动已停止的 Neptune 数据库集群

您只能启动处于停止状态的 Neptune 数据库集群。在启动集群时，它的所有数据库实例将再次变得可用。集群保留其配置设置，例如，终端节点、参数组和 VPC 安全组。

使用 AWS Management Console启动已停止的数据库集群

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)，然后选择集群。您可以从该页面中执行启动操作，或者导航到该数据库集群的详细信息页面并从那里启动。
3. 在 Actions (操作) 中，选择 Start (启动)。

使用 AWS CLI启动已停止的数据库集群

要使用 AWS CLI 启动已停止的数据库集群，请使用 `--db-cluster-identifier` 参数调用 start-db-cluster 命令来指定要启动的已停止数据库集群。提供您在创建数据库集群时选择的集群名称，或使用您选择的在其末尾附加 `-cluster` 的数据库实例名称。

Example

```
aws neptune start-db-cluster --db-cluster-identifier mydbcluster
```

使用 Neptune 管理 API 启动已停止的数据库集群

要使用 Neptune 管理 API 启动 Neptune 数据库集群，请使用 `DBCluster` 参数调用 StartDBCluster API，以指定要启动的已停止数据库集群。提供您在创建数据库集群时选择的集群名称，或使用您选择的在其末尾附加 `-cluster` 的数据库实例名称。

使用快速重置 API 清空 Amazon Neptune 数据库集群

Neptune 快速重置 REST API 允许您快速轻松地重置 Neptune 图形，同时删除其所有数据。

您可以使用 %db_reset 行魔术命令在 Neptune 笔记本中执行此操作。

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.4.0 开始推出。

- 在大多数情况下，快速重置操作会在几分钟内完成。持续时间可能会有所不同，具体取决于启动操作时集群上的负载。
- 快速重置操作不会导致额外的 I/O。
- 快速重置后，存储卷大小不会缩小。取而代之的是，当插入新数据时，存储空间会被重复使用。这意味着在快速重置操作之前和之后拍摄的快照的卷大小将相同。使用在快速重置操作之前和之后创建的快照的已还原集群的卷大小也将相同。
- 作为重置操作的一部分，数据库集群中的所有实例都将重启。

> **Note**
>
> 在极少数情况下，这些服务器重启也可能导致集群的失效转移。

> **Important**
>
> 使用快速重置可能会破坏您的 Neptune 数据库集群与其它服务的集成。例如：
>
> - 快速重置会从您的数据库中删除所有流数据并完全重置流。这意味着，如果没有新的配置，流使用者可能无法再工作。
> - 快速重置会移除有关 Neptune ML 正在使用的 SageMaker 资源的所有元数据，包括任务和端点。它们继续存在于 SageMaker 中，您可以继续使用现有的 SageMaker 端点进行 Neptune ML 推理查询，但是 Neptune ML 管理 API 不再适用于它们。
> - 诸如与 ElasticSearch 的全文搜索集成之类的集成也会因快速重置而消失，必须手动重新建立这些集成，然后才能再次使用。

使用 API 从 Neptune 数据库集群中删除所有数据

1. 首先，生成一个令牌，然后可以使用该令牌来执行数据库重置。此步骤旨在帮助防止任何人意外重置数据库。

   为此，您可以向数据库集群的写入器实例上的 `/system` 端点发送 `HTTP POST` 请求以指定 `initiateDatabaseReset` 操作。

   使用 JSON 内容类型的 `curl` 命令将是：

   ```
   curl -X POST \
     -H 'Content-Type: application/json' \
         https://your_writer_instance_endpoint:8182/system \
     -d '{ "action" : "initiateDatabaseReset" }'
   ```

   或者，使用 `x-www-form-urlencoded` 内容类型：

   ```
   curl -X POST \
     -H 'Content-Type: application/x-www-form-urlencoded' \
         https://your_writer_instance_endpoint:8182/system \
     -d 'action=initiateDatabaseReset '
   ```

   `initiateDatabaseReset` 请求在其 JSON 响应中返回重置令牌，如下所示：

   ```
   {
     "status" : "200 OK",
     "payload" : {
       "token" : "new_token_guid"
     }
   }
   ```

   令牌在发出后一小时（60 分钟）内保持有效。

   如果您将请求发送到读取器实例或状态端点，Neptune 将引发 `ReadOnlyViolationException`。

   如果您发送多个 `initiateDatabaseReset` 请求，则只有最新生成的令牌将对第二步有效，即您实际执行重置操作。

如果服务器在您的 initiateDatabaseReset 请求后立即重启，则生成的令牌将失效，您需要发送新请求才能获取新令牌。

2. 接下来，您将使用从 initiateDatabaseReset 中获得的令牌，将 performDatabaseReset 请求发送到数据库集群的写入器实例上的 /system 端点。这将从数据库集群中删除所有数据。

   使用 JSON 内容类型的 curl 命令是：

```
curl -X POST \
  -H 'Content-Type: application/json' \
      https://your_writer_instance_endpoint:8182/system \
  -d '{
        "action" : "performDatabaseReset",
        "token" : "token_guid"
      }'
```

   或者，使用 x-www-form-urlencoded 内容类型：

```
curl -X POST \
  -H 'Content-Type: application/x-www-form-urlencoded' \
      https://your_writer_instance_endpoint:8182/system \
  -d 'action=performDatabaseReset&token=token_guid'
```

   该请求会返回 JSON 响应。如果请求被接受，则响应为：

```
{
  "status" : "200 OK"
}
```

   如果您发送的令牌与发出的令牌不匹配，则响应如下所示：

```
{
  "code" : "InvalidParameterException",
  "requestId":"token_guid",
  "detailedMessage" : "System command parameter 'token' : 'token_guid' does not
 match database reset token"
}
```

   如果请求被接受并且重置开始，则服务器将重启并删除数据。在数据库集群重置期间，您无法向其发送任何其它请求。

使用带有 IAM-Auth 的快速重置 API

如果您在数据库集群上启用了 IAM-Auth，则可以使用 awscurl 发送使用 IAM-Auth 进行身份验证的快速重置命令：

使用 awscurl 通过 IAM-Auth 发送快速重置请求

1. 正确设置 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量（如果您使用的是临时凭证，也要正确设置 AWS_SECURITY_TOKEN）。
2. initiateDatabaseReset 请求类似于以下内容：

```
awscurl -X POST --service neptune-db "$SYSTEM_ENDPOINT" \
  -H 'Content-Type: application/json' --region us-west-2 \
  -d '{ "action" : "initiateDatabaseReset" }'
```

3. performDatabaseReset 请求类似于以下内容：

```
awscurl -X POST --service neptune-db "$SYSTEM_ENDPOINT" \
  -H 'Content-Type: application/json' --region us-west-2 \
  -d '{ "action" : "performDatabaseReset" }'
```

使用 Neptune Workbench **%db_reset** 行魔术命令重置数据库集群

Neptune Workbench 支持 %db_reset 行魔术命令，可让您在 Neptune 笔记本中快速重置数据库。

如果您在没有任何参数的情况下调用魔术命令，则会看到一个屏幕，询问您是否要删除集群中的所有数据，并显示一个复选框，要求您确认在删除集群数据后，这些数据将不再可用。此时，您可以选择继续删除数据，也可以取消操作。

更危险的选项是使用 --yes 或 -y 选项调用 %db_reset，这会导致在没有进一步提示的情况下执行删除。

您也可以分两步执行重置，就像使用 REST API 一样：

```
%db_reset --generate-token
```

响应如下：

```
{
```

```
  "status" : "200 OK",
  "payload" : {
    "token" : "new_token_guid"
  }
}
```

然后执行以下操作：

```
%db_reset --token new_token_guid
```

响应如下：

```
{
  "status" : "200 OK"
}
```

快速重置操作的常见错误代码

Neptune 错误代码	HTTP 状态	消息	示例
InvalidParameterException	400	系统命令参数“*action*”具有不支持的值“*XXX*”	参数无效
InvalidParameterException	400	为 *action* 提供的值过多	发送了一个包含多个操作的快速重置请求，其标头为“Content-type:application/x-www-form-urlencoded”
InvalidParameterException	400	重复字段“action”	发送了一个包含多个操作的快速重置请求，其标头为“Content-Type: application/json”

Neptune 错误代码	HTTP 状态	消息	示例
`MethodNotAllowedException`	400	错误路由：*/bad_endpoint*	请求发送到错误的端点
`MissingParameterException`	400	缺少必需参数：[action]	快速重置请求不包含所需的“action”参数
`ReadOnlyViolationException`	400	不允许对只读副本实例进行写入操作	快速重置请求已发送到读取器或状态端点
`AccessDeniedException`	403	身份验证令牌缺失	向启用了 IAM-Auth 的数据库端点发送了没有正确签名的快速重置请求
`ServerShutdownException`	500	数据库重置正在进行中。请在集群可用后重试查询。	当快速重置开始时，现有和传入的 Gremlin/Sparql 查询会失败。

将 Neptune 读取器实例添加到数据库集群

在 Neptune 数据库集群中，有一个主数据库实例和最多 15 个 Neptune 读取器实例。主数据库实例支持读取和写入操作，并执行针对集群卷的所有数据修改。Neptune 读取器实例连接到与主数据库实例相同的存储卷，并且仅支持读取操作。

使用读取器副本从主数据库实例中分载读取工作负载。

我们建议您将数据库集群中的主实例和 Neptune 读取器分配到多个可用区，以提高数据库集群的可用性。

以下章节介绍如何在数据库集群中创建读取器实例。

使用控制台创建 Neptune 读取器实例

为 Neptune 数据库集群创建主实例后，您可以使用 Neptune 控制台添加其它 Neptune 读取器实例。

使用 AWS Management Console创建 Neptune 阅读器实例

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要在其中创建读取器实例的数据库集群。
4. 选择操作，然后选择添加读取器。
5. 在创建副本数据库实例页面上，指定 Neptune 副本的选项。下表显示 Neptune 只读副本的设置。

对于此选项...	请执行此操作
数据库实例类	选择定义 Neptune 副本的处理和内存要求的数据库实例类。有关 Neptune 在不同区域中提供的数据库实例类的当前列表，请参阅 Neptune 定价页面。
可用区	指定可用区。选择与主数据库实例不同的区域。该列表仅包括数据库集群的数据库子网组映射的那些可用区。
加密	启用或禁用加密。
只读副本源	选择要为其创建 Neptune 副本的主实例的标识符。
数据库实例标识符	为该实例输入一个名称，该名称在您所选区域中对于您的账户是唯一的。您可选择对该名称进行一些巧妙处理，例如将所选的可用区包含在名称中（如 `neptune-us-east-1c` ）。
数据库端口	数据库实例接受连接的端口号。
数据库参数组	此实例的参数组。
日志导出	选择要发布的日志（如果有）。
Auto Minor Version Upgrade	如果您希望在 Neptune 数据库引擎的次要版本升级可用时，让 Neptune 副本自动接收这些升级，请选择是。

对于此选项...	请执行此操作
	Auto Minor Version Upgrade (自动次要版本升级) 选项仅适用于次要升级。它不适用于引擎维护补丁，这些补丁始终会自动应用以保持系统稳定性。

6. 选择创建只读副本以创建 Neptune 副本实例。

要从数据库集群中移除 Neptune 读取器实例，请按照在 Amazon Neptune 中删除数据库实例中的说明进行操作。

使用控制台修改 Neptune 数据库集群

当您使用 AWS Management Console修改数据库实例时，您可以选择立即应用来立即应用更改。如果您选择立即应用更改，将立即应用您的新更改以及等待修改队列中的所有更改。

如果您没有选择立即应用更改，更改将被放置在等待修改队列中。队列中的所有等待更改都将在下一维护时段应用。

> **Important**
>
> 如果任何等待修改需要停机，选择立即应用更改可导致相应的数据库实例意外停机。数据库集群中的其他数据库实例则无停机。

> **Note**
>
> 在 Neptune 中修改数据库集群时，立即应用设置仅影响对数据库集群标识符、IAM 数据库身份验证的更改。所有其他修改会立即应用，而不管立即应用设置的值如何。

使用控制台修改数据库集群

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择集群，然后选择要修改的数据库集群。
3. 选择操作，然后选择修改集群。此时会显示修改数据库集群页面。
4. 根据需要更改任意设置。

   > **Note**
   >
   > 在控制台中，某些实例级别更改仅适用于当前数据库实例，而其他更改适用于整个数据库集群。要在控制台中更改设置，而该设置在实例级别修改整个数据库集群，请按照修改数据库集群中的数据库实例中的说明操作。

5. 当所有更改都达到您的要求时，选择 Contiune (继续) 并查看摘要。
6. 要立即应用更改，请选择立即应用。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择修改集群以保存更改。

要编辑您的更改，请选择 Back (返回)，要取消更改，请选择 Cancel (取消)。

修改数据库集群中的数据库实例

使用控制台修改数据库集群中的数据库实例

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择实例，然后选择要修改的数据库实例。
3. 选择实例操作，然后选择修改。将显示 Modify DB Instance (修改数据库实例) 页面。
4. 根据需要更改任意设置。

   > **Note**
   >
   > 某些设置应用于整个数据库集群，并且必须在集群级别进行更改。要更改这些设置，请按照使用控制台修改 Neptune 数据库集群中的说明操作。
   > 在 AWS Management Console中，某些实例级别更改仅适用于当前数据库实例，而其他更改适用于整个数据库集群。

5. 当所有更改都达到您的要求时，选择 Contiune (继续) 并查看摘要。
6. 要立即应用更改，请选择立即应用。
7. 在确认页面上，检查您的更改。如果更改正确无误，请选择 Modify DB Instance (修改数据库实例) 保存更改。

   要编辑您的更改，请选择 Back (返回)，要取消更改，请选择 Cancel (取消)。

Amazon Neptune 中的性能和扩展

Neptune 数据库集群和实例可以在三个不同级别上进行扩展：

- 存储扩展
- 实例扩展；
- 读取扩展

Neptune 中的存储扩展

Neptune 存储会自动根据集群卷中的数据进行扩展。随着数据的增长，您的集群卷存储也在增长，除中国和 GovCloud（存储容量仅限于 64TiB）外，所有受支持区域的存储容量都高达 128TiB。

每小时检查一次集群卷的大小，以确定您的存储成本。

Neptune 数据库消耗的存储空间按每 GB 月增量计费，消耗的 I/O 按每百万请求增量计费。您只需为您的 Neptune 数据库消耗的存储空间和 I/O 付费，无需提前预调配。

有关定价信息，请参阅 Neptune 产品页。

在 Neptune 中扩展实例

可通过修改数据库集群中每个数据库实例的数据库实例类来按需扩展 Neptune 数据库集群。Neptune 支持多种优化型数据库实例类。

Neptune 中的读取扩展

可通过在数据库集群中创建最多 15 个 Neptune 副本来实现针对 Neptune 数据库集群的读取扩展。每个 Neptune 副本均返回集群卷中的相同数据，且副本滞后时间最短（通常大大少于主实例写入更新后的 100 毫秒）。当读取流量增大时，可创建额外 Neptune 副本并直接连接到这些副本，为您的数据库集群分配读取负载。Neptune 副本不必具有与主实例相同的数据库实例类。

有关将 Neptune 副本添加到数据库集群的信息，请参阅添加读取器实例。

自动扩缩 Amazon Neptune 数据库集群中的副本数量

您可以使用 Neptune 自动扩缩来自动调整数据库集群中 Neptune 副本的数量，以满足您的连接和工作负载要求。自动扩缩可让您的 Neptune 数据库集群处理工作负载的增加，然后，当工作负载减少时，自动扩缩会删除不必要的副本，这样您就无需为未使用的容量付费。

只能对已经有一个主写入器实例和至少一个只读副本实例的 Neptune 数据库集群使用自动扩缩（请参阅Amazon Neptune 数据库集群和实例）。此外，集群中的所有只读副本实例都必须处于可用状态。如果任何只读副本处于除可用状态以外的状态，则在集群中的每个只读副本都可用之前，Neptune 自动扩缩不会执行任何操作。

如果您需要创建新的集群，请参阅创建数据库集群。

使用 AWS CLI，您可以定义扩展策略并将其应用于数据库集群。您也可以使用 AWS CLI 来编辑或删除您的自动缩放策略。该策略指定以下自动扩缩参数：

- 集群中要具有的最小和最大副本数。
- 副本添加扩展活动之间的间`ScaleOutCooldown`隔和副本删除扩展活动之间的`ScaleInCooldown`间隔。
- 用于向上或向下扩展的 CloudWatch 指标和指标触发值。

可以通过多种方式减小 Neptune 自动扩缩操作的频率：

- 最初，为自动扩缩以添加或删除读取器，必须突破 `CPUUtilization` 高警报达至少 3 分钟，或者必须突破低警报达至少 15 分钟。
- 在首次添加或删除之后，后续 Neptune 自动扩缩操作的频率受到自动扩缩策略中 `ScaleOutCooldown` 和 `ScaleInCooldown` 设置的限制。

如果您使用的 CloudWatch 指标达到了您在策略中指定的高阈值，并且自上次自动扩展操作以来已经过了`ScaleOutCooldown`间隔，并且如果您的数据库集群还没有您设置的最大副本数，则 Neptune auto-scaling 会使用与数据库集群的主实例相同的实例类型创建一个新的副本。

同样，如果指标达到您指定的低阈值，自上次自动扩缩操作以来经过了 `ScaleInCooldown` 间隔，并且，如果您的数据库集群的副本数量超过了您指定的最小副本数，Neptune 自动扩缩会删除其中一个副本。

> **Note**
>
> Neptune 自动扩缩仅移除它创建的副本。它不会移除先前存在的副本。

使用 neptune_autoscaling_config 数据库集群参数，您还可以指定 Neptune 自动扩缩创建的新只读副本的实例类型、这些只读副本的维护时段以及要与每个新只读副本关联的标签。您可以在 JSON 字符串中提供这些配置设置作为 neptune_autoscaling_config 参数的值，如下所示：

```
"{
  \"tags\": [
    { \"key\" : \"reader tag-0 key\", \"value\" : \"reader tag-0 value\" },
    { \"key\" : \"reader tag-1 key\", \"value\" : \"reader tag-1 value\" },
  ],
  \"maintenanceWindow\" : \"wed:12:03-wed:12:33\",
  \"dbInstanceClass\" : \"db.r5.xlarge\"
}"
```

请注意，JSON 字符串中的引号必须全部使用反斜杠字符 (\) 进行转义。像往常一样，字符串中的所有空格都是可选的。

neptune_autoscaling_config 参数中未指定的三个配置设置中的任何一个都是从数据库集群的主写入器实例的配置中复制的。

当自动扩缩添加新的只读副本实例时，它会在数据库实例 ID 前加上autoscaled-reader 前缀（例如 autoscaled-reader-7r7t7z3lbd-20210828）。它还会为它创建的每个只读副本添加一个标签，其键为 autoscaled-reader，值为 TRUE。您可以在 AWS Management Console中的数据库实例详细信息页面的标签选项卡上看到此标签。

```
 "key" : "autoscaled-reader",  "value" : "TRUE"
```

通过自动扩缩创建的所有只读副本实例的提升层为最低优先级，默认情况下为 15。这意味着在失效转移期间，任何优先级更高的副本（如手动创建的副本）会先提升。请参阅 Neptune 数据库集群的容错能力。

Neptune 自动缩放是使用 Application Auto Scaling 实现的，其目标跟踪扩展策略使用海王星CPUUtilization CloudWatch 指标作为预定义指标。

在 Neptune 无服务器数据库集群中使用自动扩缩

当需求超过实例的容量时，Neptune 无服务器的响应速度比 Neptune 自动扩缩的速度快得多，并可以纵向扩展实例而不是添加另一个实例。自动扩缩旨在适应相对稳定的工作负载增加或减少，而无服务器则擅长处理需求中的快速峰值和抖动。

了解它们的优势，您可以将自动扩缩和无服务器结合起来，以创建一个灵活的基础设施，该基础设施将高效地处理工作负载的变化并满足需求，同时最大限度地降低成本。

为了让自动扩缩与无服务器有效协作，务必将无服务器集群的 maxNCU 设置配置得足够高，以适应需求的峰值和短暂变化。否则，瞬态更改不会触发无服务器扩展，这可能会导致自动扩缩启动许多不必要的额外实例。如果 maxNCU 设置得足够高，则无服务器扩展可以更快、以更低的成本处理这些变化。

如何为 Amazon Neptune 启用自动扩缩

只能使用 AWS CLI为 Neptune 数据库集群启用自动扩缩。您无法使用 AWS Management Console启用自动扩缩。

此外，以下 Amazon 区域不支持自动扩缩：

- 非洲（开普敦）：af-south-1
- 中东（阿联酋）：me-central-1
- AWS GovCloud （美国东部）：us-gov-east-1
- AWS GovCloud （美国西部）：us-gov-west-1

为 Neptune 数据库集群启用自动扩缩涉及三个步骤：

1. 向 Application Auto Scaling 注册数据库集群

为 Neptune 数据库集群启用自动扩缩的第一步是使用 AWS CLI 或其中一个 Application Auto Scaling SDK 向 Application Auto Scaling 注册该集群。集群必须已经有一个主实例和至少一个只读副本实例：

例如，要注册一个集群以自动扩展并添加一个至八个副本，可以按如下方式使用 AWS CLI register-scalable-target命令：

```
aws application-autoscaling register-scalable-target \
  --service-namespace neptune \
  --resource-id cluster:(your DB cluster name) \
  --scalable-dimension neptune:cluster:ReadReplicaCount \
  --min-capacity 1 \
```

```
  --max-capacity 8
```

这等同于使用 RegisterScalableTarget Application Auto Scaling API 操作。

AWS CLI register-scalable-target 命令使用以下参数：

- **service-namespace** – 设置为 neptune。

  此参数与 Application Auto Scaling API 中的 ServiceNamespace 参数相同。
- **resource-id** – 将其设置为 Neptune 数据库集群的资源标识符。资源类型为 cluster，后跟冒号 (':')，然后是数据库集群的名称。

  此参数与 Application Auto Scaling API 中的 ResourceID 参数相同。
- **scalable-dimension** – 在本例中，可扩展维度是数据库集群中副本实例的数量，因此您可以将此参数设置为 neptune:cluster:ReadReplicaCount。

  此参数与 Application Auto Scaling API 中的 ScalableDimension 参数相同。
- **min-capacity** – 由 Application Auto Scaling 管理的最小读取器数据库副本实例数。该值应设置在 0 到 15 的范围内，并且必须等于或小于在 max-capacity 中为最大 Neptune 副本数量指定的值。数据库集群中必须至少有一个读取器才能进行自动扩缩。

  此参数与 Application Auto Scaling API 中的 MinCapacity 参数相同。
- **max-capacity** – 数据库集群中读取器数据库副本实例的最大数量，包括预先存在的实例和由 Application Auto Scaling 管理的新实例。此值必须设置在 0 到 15 的范围内，并且必须等于或大于在 min-capacity 中为最小 Neptune 副本数量指定的值。

  该max-capacity AWS CLI 参数等同于 Application Auto Scaling API 中的MaxCapacity参数。

注册数据库集群时，Application Auto Scaling 会创建一个 AWSServiceRoleForApplicationAutoScaling_NeptuneCluster 服务相关角色。有关更多信息，请参阅《Application Auto Scaling 用户指南》中的 Application Auto Scaling 服务相关角色。

2. 定义用于数据库集群的自动扩缩策略

目标跟踪扩展策略定义为一个 JSON 文本对象，也可以将其保存在文本文件中。对于 Neptune，此策略目前只能使用 Neptune CPUUtilization CloudWatch 指标作为名为的预定义指标。NeptuneReaderAverageCPUUtilization

以下是 Neptune 的目标跟踪扩展配置策略示例：

```
{
  "PredefinedMetricSpecification": { "PredefinedMetricType":
 "NeptuneReaderAverageCPUUtilization" },
  "TargetValue": 60.0,
  "ScaleOutCooldown" : 600,
  "ScaleInCooldown" : 600
}
```

此处的 **TargetValue** 元素包含 CPU 利用率的百分比，高于该百分比的自动扩缩会横向扩展（即添加更多副本），低于该百分比则会横向缩减（即删除副本）。在这种情况下，触发扩展的目标百分比为 60.0%。

ScaleInCooldown 元素指定在一个横向缩减活动完成后，另一个横向缩减可以开始前的时长（秒）。默认值为 300 秒。这里，值为 600 指定一个副本删除完成与开始另一个副本删除之间必须经过至少十分钟。

ScaleOutCooldown 元素指定在一个横向扩展活动完成后，另一个横向扩展可以开始前的时长（秒）。默认值为 300 秒。这里，值为 600 指定一个副本添加完成与开始另一个副本添加之间必须经过至少十分钟。

DisableScaleIn 元素是一个布尔值，如果存在并设置为 `true`，则完全禁用横向缩减，这意味着自动扩缩可能会添加副本，但从不会删除任何副本。默认情况下，横向缩减处于启用状态，并且 `DisableScaleIn` 为 `false`。

向 Application Auto Scaling 注册 Neptune 数据库集群并在文本文件中定义 JSON 扩展策略后，接下来将扩展策略应用于已注册的数据库集群。你可以使用 AWS CLI [put-scaling-policy](#)命令来执行此操作，参数如下所示：

```
aws application-autoscaling put-scaling-policy \
  --policy-name (name of the scaling policy) \
  --policy-type TargetTrackingScaling \
  --resource-id cluster:(name of your Neptune DB cluster) \
  --service-namespace neptune \
  --scalable-dimension neptune:cluster:ReadReplicaCount \
  --target-tracking-scaling-policy-configuration file://(path to the JSON configuration
 file)
```

应用自动扩缩策略后，将在数据库集群上启用自动扩缩。

您也可以使用 AWS CLI [put-scaling-policy](#)命令更新现有的自动缩放策略。

另请参阅《App lication Auto Scaling API 参考》中的 “PutScaling政策”。

从 Neptune 数据库集群中移除自动扩缩

要从 Neptune 数据库集群中删除自动缩放，请使用 AWS CLI删除扩展策略和取消注册可扩展目标命令。

维护 Amazon Neptune 数据库集群

Neptune 会定期对其使用的所有资源执行维护，包括：

- 必要时更换底层硬件。这种情况发生在后台，您无需采取任何行动，而且通常不会影响您的操作。
- 更新底层操作系统。数据库集群中实例的操作系统升级是为了提高性能和安全性，因此，通常应尽快完成升级。通常而言，更新大约需要花费 10 分钟。操作系统更新不会更改数据库实例的数据库引擎版本或数据库实例类。

  通常最好先更新数据库集群中的读取器实例，然后更新写入器实例。在发生失效转移时，同时更新读取器和写入器可能会导致停机。注意，在操作系统更新之前，数据库实例不会自动备份，因此，请务必在应用操作系统更新之前进行手动备份。
- 更新 Neptune 数据库引擎。Neptune 定期发布各种引擎更新，以引入新特征和改进并修复错误。

引擎版本号

引擎版本 1.3.0.0 之前的版本编号

在 2019 年 11 月之前，Neptune 一次仅支持一个引擎版本，引擎版本号均采用 `1.0.1.0.200<xxx>` 形式，其中 `xxx` 为补丁号。所有新引擎版本均作为早期版本的补丁发布。

自 2019 年 11 月起，Neptune 开始支持多个版本，使客户能够更好地控制他们的升级路径。因此，引擎版本编号发生了变化。

从 2019 年 11 月到引擎版本 1.3.0.0，引擎版本号分为 5 个部分。以版本号 `1.0.2.0.R2` 为例：

- 第一部分始终是 1。
- 第二部分 `0`（在 `1.0.2.0.R2` 中）是数据库的主要版本号。
- 第三部分和第四部分 `2.0`（在 `1.0.2.0.R2` 中）都是次要版本号。
- 第五部分 `R2`（在 `1.0.2.0.R2` 中）是补丁号。

大多数更新都是补丁更新，补丁和次要版本更新之间的区别并不总是很清楚。

自引擎版本 1.3.0.0 起的版本编号

从引擎版本 1.3.0.0 开始，Neptune 改变了引擎更新的编号和管理方式。

引擎版本号现在有四个部分，每个部分对应于一种版本类型，如下所示：

####.####.####.####

以前作为补丁发布的那种不间断的更改现在作为次要版本发布，您可以使用 `AutoMinorVersionUpgrade` 实例设置对其进行管理。

这意味着，如果您愿意，则每次发布新的次要版本时，都可以通过订阅 `RDS-EVENT-0156` 事件来接收通知（请参阅订阅 Neptune 事件通知）。

补丁版本现在保留用于紧急的定向修复，并使用版本号的最后一部分（`*.*.*.1`、`*.*.*.2` 等）进行编号。

Amazon Neptune 中不同类型的引擎版本

与引擎版本号的四个部分相对应的四种引擎版本如下所示：

- 产品版本 - 只有当产品在功能或界面上发生彻底的根本性变化时，才会更改此版本。当前 Neptune 产品版本为 1。
- 主要版本 - 主要版本引入了重要的新特征和重大更改，其使用寿命通常至少为两年。
- 次要版本 - 次要版本可以包含新特征、改进和错误修复，但不包含任何重大更改。您可以选择是否在下一个维护时段内自动应用它们，也可以选择在发布时接收通知。
- 补丁版本 - 发布补丁版本仅用于解决紧急错误修复或关键安全更新。它们很少包含重大更改，并且会在发布后的下一个维护时段内自动应用。

Amazon Neptune 主要版本更新

主要版本更新通常会引入一个或多个重要的新特征，且通常包含重大更改。它的支持寿命通常约为两年。引擎版本中列出了 Neptune 的主要版本，及其发布日期和预计的使用寿命。

在您使用的主要版本到期之前，主要版本更新完全是可选的。如果您选择升级到新的主要版本，则必须使用 AWS CLI 或 Neptune 控制台自行安装新版本，如主要版本升级。中所述。

但是，如果您使用的主要版本已过期，则系统会通知您需要升级到最新的主要版本。然后，如果您没有在通知后的宽限期内进行升级，则系统会自动安排在下一个维护时段内升级到最新的主要版本。请参阅引擎版本的使用寿命了解更多信息。

Amazon Neptune 次要版本更新

大多数 Neptune 引擎更新都是次要版本更新。它们经常发生，并且不包含重大更改。

如果您在数据库集群的写入器（主）实例中将 AutoMinorVersionUpgrade 字段设置为 true，则次要版本更新将在数据库集群中的所有实例发布后的下一个维护时段内自动应用于这些实例。

如果您在数据库集群的写入器实例中将 AutoMinorVersionUpgrade 字段设置为 false，则只有在您明确安装时才会应用这些更新。

> **Note**
>
> 次要版本更新是独立的（它们不依赖于以前对同一主要版本的次要版本更新）累积更新（它们包含先前次要版本更新中引入的所有特征和修复）。这意味着，无论您是否安装了以前的次要版本更新，您都可以安装任何给定的次要版本更新。

通过订阅 RDS-EVENT-0156 事件，可以轻松跟踪次要版本的发布（参见订阅 Neptune 事件通知）。然后，每次发布新的次要版本时，您都会收到通知。

此外，无论您是否订阅通知，您都可以随时查看哪些更新有待处理。

Amazon Neptune 补丁版本更新

如果出现影响实例可靠性的安全问题或其他严重缺陷，则 Neptune 会部署强制补丁。它们将在下一个维护时段应用于数据库集群中的所有实例，而无需您进行任何干预。

只有在不部署补丁版本的风险超过与部署补丁版本相关的风险和停机时间时，才会部署补丁版本。补丁版本很少发生（通常几个月一次），并且几乎不会需要过长的维护时段。

规划 Amazon Neptune 主要引擎版本的使用寿命

Neptune 引擎版本几乎总是在日历季度末达到使用寿命的尽头。只有在出现重大的安全或可用性问题时，才会出现例外。

当引擎版本达到其使用寿命结束时，您需要将 Neptune 数据库升级到较新的版本。

通常，Neptune 引擎版本将继续可用，如下所示：

- 次要引擎版本：次要引擎版本在发布后的至少 6 个月内保持可用。
- 主要引擎版本：主要引擎版本在发布后的至少 12 个月内保持可用。

在引擎版本使用寿命结束前至少 3 个月，AWS 将向与您的 AWS 账户关联的电子邮件地址发送自动电子邮件通知，并将相同的邮件发布到您的 AWS Health Dashboard。这将使您有时间计划和准备升级。

当引擎版本达到其使用寿命结束时，您将无法再使用该版本创建新的集群或实例，自动缩放功能也无法使用该版本创建实例。

实际达到使用寿命结束的引擎版本将在维护时段期间自动升级。在引擎版本使用寿命结束前 3 个月发送给您的邮件将包含有关此自动更新将涉及的内容的详细信息，包括您将自动升级到的版本、对数据库集群的影响以及我们建议的操作。

> **Important**
>
> 您有责任使数据库引擎版本保持最新。AWS 敦促所有客户将其数据库升级到最新的引擎版本，以便从最新的安全、隐私和可用性保护措施中受益。如果您在弃用日期之后在不受支持的引擎或软件（“旧版引擎”）上运行数据库，则更有可能面临安全、隐私和运营风险，包括停机事件。
>
> 在任何引擎上运行您的数据库均受管理您使用 AWS 服务的协议的约束。旧版引擎并非普遍可用。AWS 不再为旧版引擎提供支持，如果 AWS 确定旧版引擎对服务、AWS、其关联公司或任何第三方构成安全或责任风险或损害风险，则 AWS 可以随时限制对任何旧版引擎的访问或使用。您决定继续在旧版引擎中运行您的内容可能会导致您的内容不可用、损坏或无法恢复。
>
> 在旧版引擎上运行的数据库受服务水平协议 (SLA) 例外情况的约束。
>
> 在旧版引擎上运行的数据库和相关软件包含漏洞、错误、缺陷和/或有害组件。因此，无论协议或服务条款中存在任何相反之处，AWS 均“按原样”提供旧版引擎。

管理 Neptune 数据库集群的引擎更新

> **Note**
>
> 更新将同时应用于数据库集群中的所有实例。更新操作需要在这些实例上重启数据库，因此，会出现从 20 或 30 秒到几分钟的停机，之后您可以继续使用数据库集群。在极少数情况下，可能需要多可用区失效转移才能完成实例的维护更新。
>
> 对于可能需要更长时间才能应用的主要版本升级，您可以使用蓝绿部署策略，最大限度地减少停机时间。

确定您当前使用的引擎版本

您可以使用 AWS CLI `get-engine-status` 命令来检查您的数据库集群当前使用的引擎发布版本：

```
aws neptunedata get-engine-status
```

JSON 输出包含 "dbEngineVersion" 字段，如下所示：

```
  "dbEngineVersion": "1.3.0.0",
```

查看哪些更新有待处理且可用

您可以使用 Neptune 控制台检查数据库集群的待处理更新。在左列中选择数据库，然后在数据库窗格中选择您的数据库集群。待处理的更新列在维护列中。如果您依次选择操作和维护，则有三种操作选择：

- 立即升级。
- 在下一个时段升级。
- 推迟升级。

您可以使用 AWS CLI 列出待处理的引擎更新，如下所示：

```
aws neptune describe-pending-maintenance-actions \
  --resource-identifier (ARN of your DB cluster)
  --region (your region) \
  --engine neptune
```

您可以使用 AWS CLI 列出可用的引擎更新，如下所示：

```
aws neptune describe-db-engine-versions \
  --region (your region) \
  --engine neptune
```

可用的引擎版本列表仅包括版本号高于当前版本且已定义升级路径的那些版本。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。次要版本升级可能引入会影响代码的新特征或行为，即使没有任何重大更改也如此。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是使用 Neptune 蓝绿部署解决方案。这样，您就可以在新版本上运行应用程序和查询，而不会影响您的生产数据库集群。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

Neptune 维护时段

每周维护时段为 30 分钟，在此期间将应用定期引擎更新和其他系统更改。大部分维护事件将在 30 分钟的维护时段内完成，但较大的维护事件有时可能需要更长时间才能完成。

每个数据库集群都有每周 30 分钟的维护时段。如果在创建数据库集群时未为其指定首选时段，则 Neptune 随机选择一周中的某一天，然后从这一天 8 小时的时间块中随机分配 30 分钟的时段（随区域的不同而变化）。

例如，以下是多个 AWS 区域使用的维护时段的 8 小时时间块：

区域	时间段
美国西部（俄勒冈州）区域	06:00–14:00 UTC
美国西部（北加利福尼亚）区域	06:00–14:00 UTC
美国东部（俄亥俄州）区域	03:00–11:00 UTC
欧洲地区（爱尔兰）区域	22:00–06:00 UTC

维护时段决定待处理操作何时开始，大多数维护操作都是在该时段内完成的，但是较大的维护任务可以在该时段结束之后继续进行。

移动数据库集群维护时段

理想情况下，您的维护时段应在集群使用率最低的时候。如果您的当前时段不在这时候，则您可以将其移到更好的时间，如下所示：

更改数据库集群维护时段

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择数据库。
3. 选择要更改维护时段的数据库集群。
4. 选择修改。
5. 在修改集群页面底部选择显示更多。
6. 在首选维护时段部分，根据需要设置维护时段的日期、时间和持续时间。
7. 选择下一步。

   在确认页面上，检查您的更改。
8. 要立即应用对维护时段的更改，请选择立即应用。
9. 选择提交以应用更改。

   要编辑您的更改，请选择以前，要取消更改，请选择取消。

使用 AutoMinorVersionUpgrade 控制次要版本的自动更新

> **Important**
>
> AutoMinorVersionUpgrade 仅对引擎版本 1.3.0.0 以上的次要版本升级有效。

如果您在数据库集群的写入器（主）实例中将 AutoMinorVersionUpgrade 字段设置为 true，则次要版本更新将在数据库集群中的所有实例发布后的下一个维护时段内自动应用于这些实例。

如果您在数据库集群的写入器实例中将 AutoMinorVersionUpgrade 字段设置为 false，则只有在您明确安装时才会应用这些更新。

> **Note**
>
> 无论 AutoMinorVersionUpgrade 参数的设置方式如何，补丁版本（*.*.*.1、*.*.*.2 等）总是在下一个维护时段内自动安装。

您可以使用 AWS Management Console 设置 AutoMinorVersionUpgrade，如下所示：

使用 Neptune 控制台设置 **AutoMinorVersionUpgrade**

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要为其设置 AutoMinorVersionUpgrade 的数据库集群的主（写入器）实例。
4. 选择 Modify(修改)。
5. 在修改集群页面底部选择显示更多。
6. 在展开页面的底部，选择开启自动次要版本升级或关闭自动次要版本升级。
7. 选择下一步。

   在确认页面上，检查您的更改。
8. 要应用对自动次要版本升级的更改，请选择立即应用。
9. 选择提交以应用更改。

   要编辑您的更改，请选择以前，要取消更改，请选择取消。

您也可以使用 AWS CLI 来设置 AutoMinorVersionUpgrade 字段。例如，要将其设置为 true，您可以使用如下命令：

```
aws neptune modify-db-instance \
  --db-instance-identifier (the ID of your cluster's writer instance) \
  --auto-minor-version-upgrade \
  --apply-immediately
```

同样，要将其设置为 false，可使用如下命令：

```
aws neptune modify-db-instance \
  --db-instance-identifier (the ID of your cluster's writer instance) \
  --no-auto-minor-version-upgrade \
  --apply-immediately
```

手动安装 Neptune 引擎的更新

安装主要版本引擎升级

必须始终手动安装主要引擎版本。为了最大限度地减少停机时间并留出充足的时间进行测试和验证，安装新的主要版本的最佳方法通常是使用 Neptune 蓝绿部署解决方案。

在某些情况下，您还可以使用创建数据库集群时使用的 AWS CloudFormation 模板来安装主要版本升级（请参阅 使用 AWS CloudFormation 模板更新 Neptune 数据库集群的引擎版本）。

如果要立即安装主要版本更新，则可以使用如下所示的 CLI 命令：

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (identifier for your neptune cluster) \
  --engine neptune \
  --engine-version (the new engine version) \
  --apply-immediately
```

确保指定要升级的引擎版本。如果不这样做，您的引擎可能会升级到非最新版本或不是您期望的版本。

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。

如果集群使用自定义集群参数组，请确保指定使用此参数：

```
  --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保指定它使用此参数：

```
  ---db-instance-parameter-group-name (name of the custom instance parameter group)
```

使用 AWS Management Console 安装次要版本引擎升级

使用 Neptune 控制台执行次要版本升级

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择数据库，然后选择您希望修改的数据库集群。
3. 选择 Modify(修改)。
4. 在实例规格下，选择要升级的新版本。

5. 选择下一步。
6. 如果要立即应用更改，请选择立即应用。
7. 选择提交以更新您的数据库集群。

使用 AWS CLI 安装次要版本引擎升级

您可以使用如下命令来执行次要版本升级，而无需等待下一个维护时段：

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (your-neptune-cluster) \
  --engine-version (new-engine-version) \
  --apply-immediately
```

如果要使用 AWS CLI 手动升级，请确保包括要升级到的引擎版本。如果不这样做，您的引擎可能会升级到非最新版本或不是您期望的版本。

从 1.2.0.0 之前的版本升级到引擎版本 1.2.0.0 或更高版本

引擎版本 1.2.0.0 引入了几项重大更改，这些更改可能会使从早期版本升级变得更加复杂：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志的过程完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所

示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

使用 AWS CloudFormation 模板更新 Neptune 数据库集群的引擎版本

您可以重复使用用于创建 Nept AWS CloudFormation une 数据库集群的 Neptune 模板来更新其引擎版本。

Neptune 引擎版本升级可以是次要版本升级，也可以是主要版本升级。使用 AWS CloudFormation 模板可以帮助进行主要版本升级，其中通常包含重大更改。由于主要版本升级可能包含不与现有应用程序向后兼容的数据库更改，因此在升级时您可能还需要对应用程序进行更改。请务必在升级之前进行测试，我们强烈建议您在升级之前，始终创建数据库集群的手动快照。

请注意，您必须为每个主要版本单独进行引擎升级。您不能跳过某个主要版本并直接升级到以下主要版本。

在 2023 年 5 月 17 日之前，如果您使用 Neptune AWS CloudFormation 堆栈升级引擎版本，它只会创建一个新的空数据库集群来代替当前的数据库集群。但是，从2023年5月17日起，Neptune AWS CloudFormation 堆栈现在支持就地升级引擎，以保留您的现有数据。

对于主要版本升级，您的模板应在 `DBCluster` 中设置以下属性：

- `DBClusterParameterGroup`（自定义或默认）
- `DBInstanceParameterGroupName`
- `EngineVersion`

同样，对于附加到数据库集群的数据库实例，您应该设置：

- `DBParameterGroup`（自定义/默认）

确保在模板中定义了所有参数组，无论它们是默认参数组还是自定义参数组。

对于自定义参数组，请确保现有自定义参数组系列与新的引擎版本兼容。1.2.0.0 之前的引擎版本使用参数组系列 `neptune1`，而 1.2.0.0 以后的引擎版本需要参数组系列 `neptune1.2`。请参阅Amazon Neptune 参数组了解更多信息。

对于主要引擎版本升级，请在 `DBCluster` `DBInstanceParameterGroupName` 字段中指定具有相应系列的参数组。

应将默认参数组升级为与新引擎版本兼容的参数组。

请注意，Neptune 会在引擎升级后自动重启数据库实例。

主题

示例：从 1.2.0.1 到 1.2.0.2 的次要版本引擎升级

找到要升级的数据库集群，以及用于创建数据库集群的模板。例如：

```
Description: Base Template to create Neptune Stack with Engine Version 1.2.0.1 using
 custom Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-cluster-parameter-group-description
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-parameter-group-description
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.1
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
```

```
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

将 EngineVersion 属性从 1.2.0.1 更新为 1.2.0.2：

```
Description: Template to upgrade minor engine version to 1.2.0.2
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-cluster-parameter-group-description
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: test-cfn-neptune-db-parameter-group-description
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
```

```
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

现在使用 AWS CloudFormation 来运行修改后的模板。

示例：使用默认参数组将主要版本从 1.1.1.0 升级到 1.2.0.2

找到要升级的 DBCluster，以及用于创建数据库集群的模板。例如：

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 default Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
```

```
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- 将默认 DBClusterParameterGroup 更新为新引擎版本（此处为 default.neptune1.2）使用的参数组系列中的参数组。
- 对于附加到 DBCluster 的每个 DBInstance，请将默认 DBParameterGroup 更新为新引擎版本（此处为 default.neptune1.2）使用的系列中的参数组。
- 将 DBInstanceParameterGroupName 属性设置为该系列中的默认参数组（此处为 default.neptune1.2）。
- 将 EngineVersion 属性从 1.1.0.0 更新为 1.2.0.2：

模板应该如下所示：

```
Description: Template to upgrade major engine version to 1.2.0.2 by using upgraded
 default parameter groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName: default.neptune1.2
      DBInstanceParameterGroupName: default.neptune1.2
  NeptuneDBInstance:
```

```
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
```

现在使用 AWS CloudFormation 来运行修改后的模板。

示例：使用自定义参数组将主要版本从 1.1.1.0 升级到 1.2.0.2

找到要升级的 DBCluster，以及用于创建数据库集群的模板。例如：

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 custom Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Name: engineupgradetestcpg
      Family: neptune1
      Description: 'NeptuneDBClusterParameterGroup with family neptune1'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Name: engineupgradetestpg
      Family: neptune1
      Description: 'NeptuneDBParameterGroup1 with family neptune1'
      Parameters:
```

```
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- 将自定义DBClusterParameterGroup系列更新为新引擎版本所使用的系列default.neptune1.2（此处）。
- 对于DBInstance附加到的每个自定义系列DBCluster，请将自定义DBParameterGroup系列更新为新引擎版本所使用的系列（此处default.neptune1.2）。
- 将 DBInstanceParameterGroupName 属性设置为该系列中的参数组（此处为 default.neptune1.2）。
- 将 EngineVersion 属性从 1.1.0.0 更新为 1.2.0.2：

模板应该如下所示：

```
Description: Template to upgrade major engine version to 1.2.0.2 by modifying existing
 custom parameter groups
Parameters:
```

```
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Name: engineupgradetestcpgnew
      Family: neptune1.2
      Description: 'NeptuneDBClusterParameterGroup with family neptune1.2'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Name: engineupgradetestpgnew
      Family: neptune1.2
      Description: 'NeptuneDBParameterGroup1 with family neptune1.2'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
      DBInstanceParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  NeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
Outputs:
```

```
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

现在使用 AWS CloudFormation 来运行修改后的模板。

示例：混合使用默认参数组和自定义参数组，将主要版本从 1.1.1.0 升级到 1.2.0.2

找到要升级的 DBCluster，以及用于创建数据库集群的模板。例如：

```
Description: Base Template to create Neptune Stack with Engine Version 1.1.1.0 using
 custom Parameter Groups
Parameters:
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1
      Description: 'NeptuneDBClusterParameterGroup with family neptune1'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1
      Description: 'NeptuneDBParameterGroup with family neptune1'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.1.1.0
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
    DependsOn:
      - NeptuneDBClusterParameterGroup
  CustomNeptuneDBInstance:
```

```
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
  DefaultNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

- 对于自定义集群参数组，请将 DBClusterParameterGroup 系列更新为对应于新引擎版本（即 neptune1.2）的系列。
- 对于默认集群参数组，请将 DBClusterParameterGroup 更新为与新引擎版本（即 default.neptune1.2）相对应的默认参数组。
- 对于附加到 DBCluster 的每个 DBInstance，将默认 DBParameterGroup 更新为新引擎版本（此处为 default.neptune1.2）所使用的系列中的参数组，并将自定义参数组更新为使用由新引擎版本（此处为 neptune1.2）支持的系列的参数组。
- 将 DBInstanceParameterGroupName 属性设置为新引擎版本支持的系列中的参数组。

模板应该如下所示：

```
Description: Template to update Neptune Stack to Engine Version 1.2.0.1 using custom
 and default Parameter Groups
Parameters:
```

```
  DbInstanceType:
    Description: Neptune DB instance type
    Type: String
    Default: db.r5.large
Resources:
  NeptuneDBClusterParameterGroup:
    Type: 'AWS::Neptune::DBClusterParameterGroup'
    Properties:
      Family: neptune1.2
      Description: 'NeptuneDBClusterParameterGroup with family neptune1.2'
      Parameters:
        neptune_enable_audit_log: 0
  NeptuneDBParameterGroup:
    Type: 'AWS::Neptune::DBParameterGroup'
    Properties:
      Family: neptune1.2
      Description: 'NeptuneDBParameterGroup1 with family neptune1.2'
      Parameters:
        neptune_query_timeout: 20000
  NeptuneDBCluster:
    Type: 'AWS::Neptune::DBCluster'
    Properties:
      EngineVersion: 1.2.0.2
      DBClusterParameterGroupName:
        Ref: NeptuneDBClusterParameterGroup
      DBInstanceParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBClusterParameterGroup
  CustomNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName:
        Ref: NeptuneDBParameterGroup
    DependsOn:
      - NeptuneDBCluster
      - NeptuneDBParameterGroup
  DefaultNeptuneDBInstance:
    Type: 'AWS::Neptune::DBInstance'
    Properties:
      DBClusterIdentifier:
```

```
        Ref: NeptuneDBCluster
      DBInstanceClass:
        Ref: DbInstanceType
      DBParameterGroupName: default.neptune1.2
    DependsOn:
      - NeptuneDBCluster
Outputs:
  DBClusterId:
    Description: Neptune Cluster Identifier
    Value:
      Ref: NeptuneDBCluster
```

现在使用 AWS CloudFormation 来运行修改后的模板。

Neptune 中的数据库克隆

使用数据库克隆，可以快速而经济高效地在 Amazon Neptune 中创建您的所有数据库的克隆。克隆数据库在首次创建时只需要很少的额外空间。数据库克隆使用写入时复制协议。在源数据库或克隆数据库上的数据发生变化时会复制数据。您可以多次克隆同一数据库集群。还可以基于其他克隆来创建更多克隆。有关写入时复制协议在 Neptune 存储上下文中的工作原理的更多信息，请参阅写入时复制。

您可以在很多情况下使用数据库克隆，特别是在您不希望影响生产环境时，例如：

- 测试并评估更改的影响，例如架构更改或参数组更改。
- 执行工作负载密集型操作，例如导出数据或运行分析查询。
- 在非生产环境中为进行开发或测试创建生产数据库集群的副本。

使用 AWS Management Console创建数据库集群的克隆

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择实例。选择要创建克隆的数据库集群的主实例。
3. 选择 Instance actions (实例操作)，然后选择 Create clone (创建克隆)。
4. 在 Create Clone (创建克隆) 页面上，输入克隆数据库集群的主实例名称作为 DB instance identifier (数据库实例标识符)。

   如果需要，配置克隆数据库集群的任何其他设置。有关不同的数据库集群设置的信息，请参阅使用控制台启动。
5. 选择 Create Clone 启动克隆数据库集群。

使用 AWS CLI创建数据库集群的克隆

- 调用 Neptune restore-db-cluster-to-point-in-time AWS CLI 命令并提供以下值：
  - `--source-db-cluster-identifier` – 要创建克隆的源数据库集群的名称。
  - `--db-cluster-identifier` – 克隆数据库集群的名称。
  - `--restore-type copy-on-write` – `copy-on-write` 值指示应创建克隆数据库集群。
  - `--use-latest-restorable-time` – 它指定应使用最新的可恢复备份时间。

> **Note**
>
> restore-db-cluster-to-point-in-time AWS CLI 命令仅克隆数据库集群，而不克隆该数据库集群的数据库实例。

以下 Linux/UNIX 示例从 `source-db-cluster-id` 数据库集群创建一个克隆并将该克隆命名为 `db-clone-cluster-id`。

```
aws neptune restore-db-cluster-to-point-in-time \
  --region us-east-1 \
  --source-db-cluster-identifier source-db-cluster-id \
  --db-cluster-identifier db-clone-cluster-id \
  --restore-type copy-on-write \
  --use-latest-restorable-time
```

如果将 \ 行尾转义字符替换为 Windows 上的 ^ 等效字符，则该示例也适用于 Windows：

```
aws neptune restore-db-cluster-to-point-in-time ^
  --region us-east-1 ^
  --source-db-cluster-identifier source-db-cluster-id ^
  --db-cluster-identifier db-clone-cluster-id ^
  --restore-type copy-on-write ^
  --use-latest-restorable-time
```

限制

Neptune 中的数据库克隆具有以下限制：

- 不能跨 AWS 区域创建克隆数据库。克隆数据库必须在与源数据库相同的区域中创建。
- 克隆的数据库始终使用从中克隆的源数据库所用 Neptune 引擎版本的最新补丁。即使源数据库尚未升级到该补丁版本也是如此。不过，引擎版本本身不会改变。
- 目前，Neptune 数据库集群的每个副本最多只能有 15 个克隆，包括基于其它克隆的克隆。达到该限制后，必须制作数据库的另一个副本，而不是克隆它。但是，如果您制作一个新副本，它也可以有多达 15 个克隆。
- 目前不支持跨账户数据库克隆。

- 您可以为克隆提供不同的 Virtual Private Cloud (VPC)。不过，在这些 VPC 中的子网必须映射到同一组可用区。

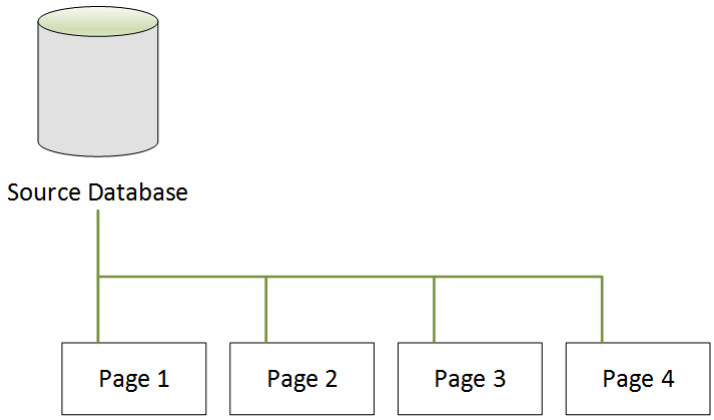
数据库克隆的写入时复制协议

以下场景说明了写入时复制协议的工作原理。

- 克隆之前的 Neptune 数据库
- 克隆之后的 Neptune 数据库
- 当对源数据库进行更改时
- 当对克隆数据库进行更改时

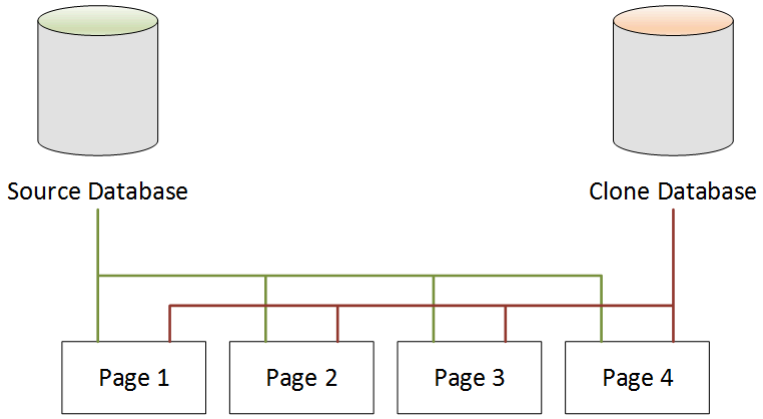
克隆之前的 Neptune 数据库

源数据库中的数据以页的形式存储。在下图中，源数据库有四个页面。



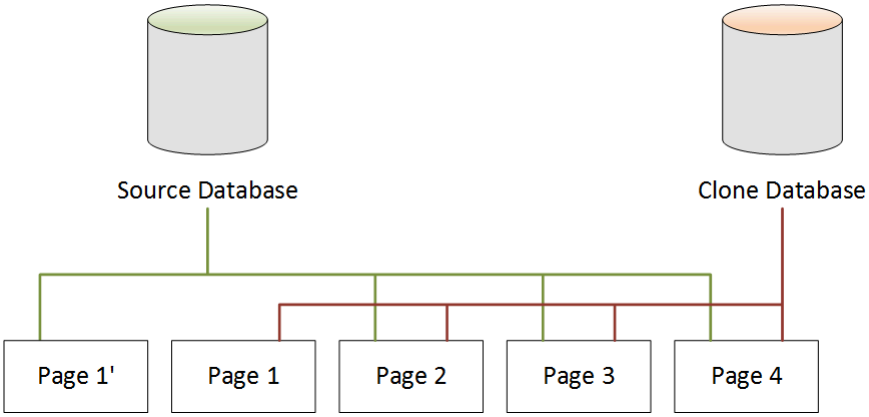
克隆之后的 Neptune 数据库

如下图所示，在克隆数据库之后，源数据库中没有更改。源数据库和克隆数据库均指向相同的四个页面。没有物理复制任何页面，因此无需额外的存储。



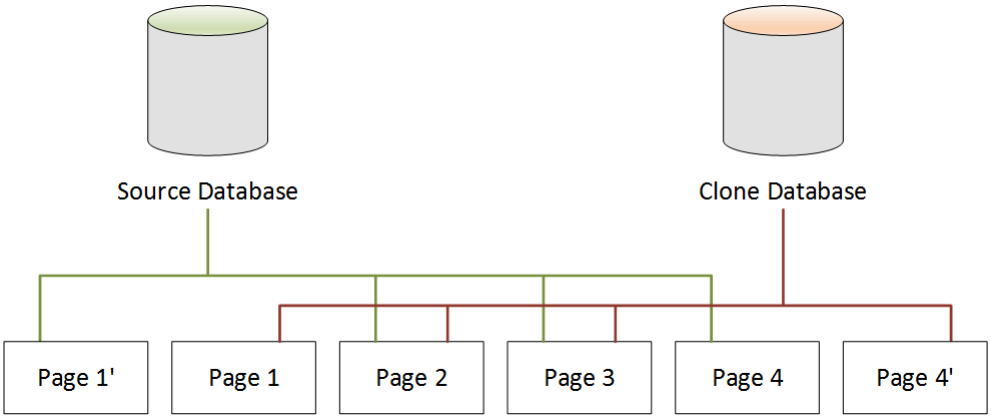
当对源数据库进行更改时

在下面的示例中，源数据库对 Page 1 中的数据进行了更改。它使用额外的存储创建名为 Page 1 的新页，而不是写入原始 Page 1'。源数据库现在指向新 Page 1'，也指向Page 2、Page 3 和 Page 4。克隆数据库继续指向 Page 1 到 Page 4。



当对克隆数据库进行更改时

在下图中，克隆数据库还进行了一项更改，这次是在 Page 4 中。使用额外的存储创建名为 Page 4 的新页，而不是写入原始 Page 4'。源数据库继续指向 Page 1'，以及 Page 2 到 Page 4，但克隆数据库现在指向 Page 1 到 Page 3，以及 Page 4'。



如第二种情况中所示，在数据库克隆之后，创建克隆时无需额外的存储。不过，如第三和第四种情况所示，源数据库和克隆数据库中发生更改之后，将仅创建更改的页面。随着时间推移，当源数据库和克隆数据库上出现了更多更改时，逐渐需要更多存储来捕获和存储更改。

删除源数据库

删除源数据库不会影响与其关联的克隆数据库。克隆数据库继续指向以前由源数据库拥有的页面。

管理 Amazon Neptune 实例

以下各部分介绍有关实例级别操作的信息。

主题

Neptune T3 可突增实例类

除固定性能的实例类（例如 R5 和 R6）之外，Amazon Neptune 还向您提供了使用可突增性能 T3 实例的选项。在开发图形应用程序时，您希望数据库既快速又具有响应能力，但您不需要一直使用它。在这种情况下，Neptune 的 db.t3.medium 实例类正是您应该使用的，其成本要比最便宜的固定性能实例类低得多。

可突增实例在 CPU 性能的基准级别运行，直到工作负载有更多的需求，然后在工作负载需要的时间内突增到远超基准的水平。它的每小时价格涵盖了突增，前提是平均 CPU 利用率在 24 小时内不超过基准。对于大多数开发和测试情况，这意味着以低成本获得良好的性能。

如果您从 T3 实例类开始，则以后在要转入生产阶段时，可以使用 AWS Management Console、AWS CLI 或其中一个 AWS SDK 轻松切换到固定性能的实例类。

T3 突增由 CPU 积分控制

一个 CPU 积分表示在一分钟内的完全利用了一个虚拟 CPU 核心 (vCPU)。这也可以转化为一个 vCPU 在两分钟内的 50% 使用率，或两个 vCPU 在两分钟内 25% 的使用率，依此类推。

T3 实例在空闲时可累积 CPU 积分，并在处于活动状态使用这些积分，两者的测量精度均为毫秒。db.t3.medium 实例类具有两个 vCPU，在空闲时每小时可获得 12 个 CPU 积分。这意味着每个 vCPU 的 20% 利用率可以实现零 CPU 积分余额。获得的 12 个 CPU 积分将按照 vCPU 的 20% 使用率支出（因为 20% 乘以 60 分钟也就是 12）。因此，这个 20% 的利用率是基准 利用率，可以得到正好为零的 CPU 积分余额。

空闲时间（CPU 使用率低于总共可用的 20%）可以生成在积分存储桶中存储的 CPU 积分，一个 db.t3.medium 实例类的上限为 576（24 小时内可累积的最大 CPU 积分数，即 2 x 12 x 24）。超过该限制后将直接丢弃 CPU 积分。

在必要时，只要工作负载需要，CPU 利用率可以突增至 100%，即使 CPU 积分余额降至零以下。如果实例的负余额持续 24 小时，则该时间段内产生的每 -60 个 CPU 积分会导致 0.05 美元的额外费用。但是，对于大多数开发和测试工作负载，突增前后通常都是空闲时间。

> **Note**
>
> Neptune 的 T3 实例类配置方式类似于 Amazon EC2 无限模式。

使用 AWS Management Console 创建 T3 可突增实例

在 AWS Management Console 中，您可以创建使用 db.t3.medium 实例类的主数据库集群实例或只读副本实例，也可以修改现有实例以使用 db.t3.medium 实例类。

例如，要在 Neptune 控制台中创建新的数据库集群主实例，请执行以下操作：

- 选择 Create Database（创建数据库）。
- 选择等于或高于 1.0.2.2 的数据库引擎版本。
- 在 Purpose (用途) 下，选择 Development and Testing (开发和测试)。
- 对于 DB instance class (数据库实例类)，接受默认值：db.t3.medium — 2 vCPU, 4 GiB RAM。

使用 AWS CLI 创建 T3 可突增实例

您也可以使用 AWS CLI 执行相同的操作：

```
aws neptune create-db-cluster \
    --db-cluster-identifier (name for a new DB cluster) \
    --engine neptune \
    --engine-version "1.0.2.2"

aws neptune create-db-instance \
    --db-cluster-identifier (name of the new DB cluster) \
    --db-instance-identifier (name for the primary writer instance in the cluster) \
    --engine neptune \
    --db-instance-class db.t3.medium
```

修改 Neptune 数据库实例（并立即应用）

您可将大多数更改立即应用到 Amazon Neptune 数据库实例，或者延后直至下一个维护时段。一些修改（例如，参数组更改）需要您手动重新启动数据库实例才能使更改生效。

> **Important**
>
> 如果 Neptune 必须重启您的数据库实例才能应用更改，则修改会导致中断。在修改数据库实例设置之前，请先审核对数据库和应用程序的影响。

常见设置和停机时间影响

下表包含有关哪些设置可更改、何时可应用更改、以及更改是否会导致数据库实例停机的详细信息。

数据库实例设置	停机说明	
数据库实例类	无论是立即应用还是在下一个维护时段内应用，此更改期间都会发生中断。	
数据库实例标识符	无论是立即应用还是在下一个维护时段内应用，数据库实例都将重启并在该更改期间发生中断。	
子网组	无论是立即应用还是在下一个维护时段内应用，数据库实例都将重启并在该更改期间发生中断。	
安全组	无论您指定何时进行更改，更改都会尽快异步应用，不会导致中断。	–
证书颁发机构	默认情况下，当您分配新的证书颁发机构时，数据库实例会重启。	

数据库实例设置	停机说明
Database Port	更改始终立即发生，从而导致数据库实例重启并发生中断。
数据库参数组	更改此设置不会导致中断。参数组名称本身将立即发生更改，但在您重新启动实例而不进行故障转移之前，不会应用实际参数更改。在这种情况下，数据库实例不会自动重启，参数更改也不会在下一个维护时段中应用。但是，如果修改新关联的数据库参数组中的动态参数，这些更改将立即得到应用，而无需重新启动。 有关更多信息，请参阅在 Amazon Neptune 中重启数据库实例。
数据库集群参数组	数据库参数组名称将立即更改。
备份保留期	如果您指定更改应立即发生，则此更改将立即发生。否则，如果您将该设置从一个非零值更改为另一个非零值，则将尽快地异步应用更改。任何其它更改都在下一维护时段内发生。如果从零改为非零值或从非零值改为零，则会发生服务中断。

数据库实例设置	停机说明
审计日志	如果您想通过 CloudWatch Logs 使用审计日志记录，请选择审计日志。您还必须将数据库集群参数组中的 `neptune_enable_audit_log` 参数设置为 `enable` (1)，才能启用审计日志记录。
自动次要版本升级	如果要在次要引擎版本升级可用时让 Neptune 数据库集群自动接收这些升级，请选择启用自动次要版本升级。 自动次要版本升级选项仅适用于升级到 Amazon Neptune 数据库集群的次要引擎版本。它不适用于应用于维持系统稳定性的常规修补程序。

重命名 Neptune 数据库实例

您可以使用 AWS Management Console重命名 Amazon Neptune 数据库实例。重命名数据库实例会产生深远的影响。下面是您在重命名数据库实例前应知道的事情的列表。

- 当您重命名数据库实例时，数据库实例的终端节点会发生更改，因为 URL 包含了您分配给数据库实例的名称。您应当始终将流量从旧的 URL 重定向到新的。
- 当您重命名数据库实例时，数据库实例使用的旧 DNS 名称会立刻被删除，但它可能会在缓存中保留几分钟。已重命名的数据库实例的新 DNS 名称大约在 10 分钟后生效。重命名的数据库实例在新名称生效前不可用。
- 重命名实例时，不能使用现有的数据库实例名称。
- 重命名后，所有与数据库实例关联的只读副本会保持与该实例的关联。例如，假设您有一个为生产数据库服务的数据库实例，而该实例有多个关联的只读副本。如果您重命名该数据库实例，然后在生产环境中将其替换为数据库快照，则您重命名的数据库实例仍会拥有与它关联的只读副本。
- 如果您重复使用数据库实例的名称，则与数据库实例的名称关联的指标和事件保持不变。例如，如果提升一个只读副本，然后将其重命名为以前主实例的名称，则与该主实例关联的事件和指标将与重命名的实例关联。
- 无论是否重命名，数据库实例标签会始终与数据库实例关联在一起。
- 对于重命名的数据库实例，数据库快照也会保留下来。

使用 Neptune 控制台重命名数据库实例

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要重命名的数据库实例旁边的单选按钮。
4. 在 Instance actions (实例操作) 菜单中，选择 Modify (修改)。
5. 在 DB Instance Identifier (数据库实例标识符) 文本框中，输入一个新名称。选择立即应用，然后选择继续。
6. 选择修改数据库实例以完成更改。

在 Amazon Neptune 中重启数据库实例

在某些情况下，如果修改 Amazon Neptune 数据库实例，更改与该实例关联的数据库参数组，或更改实例使用的参数组中的静态数据库参数，则必须重启该实例以应用更改。

重启数据库实例会重新启动数据库引擎服务。此外，重启还会将相关数据库参数组的任何挂起的更改应用于数据库实例。重新启动数据库实例会导致实例短暂中断，在此期间此数据库实例的状态会设置为 rebooting。如果为多可用区配置了 Neptune 实例，则可能会通过失效转移进行重启。重启完成后，即会创建 Neptune 事件。

如果您的数据库实例为多可用区部署，则可在选择重启选项时强制从一个可用区故障转移到另一个可用区。当强制失效转移数据库实例时，Neptune 自动切换到另一个可用区中的备用副本。然后，它会更新数据库实例的 DNS 记录，使其指向备用数据库实例。因此，您必须清除并重新建立到您的数据库实例的任何现有连接。

如果需要模拟数据库实例故障转移以进行测试，或是在故障转移进行之后将操作还原到原始可用区，通过故障转移重启十分有用。有关更多信息，请参阅《Amazon RDS 用户指南》中的高可用性（多可用区）。当您重启数据库集群时，它将故障转移到备用副本。重启 Neptune 副本不会初始化失效转移。

重启所需的时间是崩溃恢复过程的函数。为优化重新启动的时间，建议您在重启过程中尽可能减少数据库活动，以减少中转事务的回滚活动。

在控制台上，如果数据库实例未处于可用状态，则重启选项可能处于禁用状态。这可能是多个原因导致的，例如，正在进行的备份、客户请求的修改或维护时段操作。

> **Note**
>
> 在版本：1.2.0.0（2022 年 7 月 21 日）之前，每当主（写入器）实例重启时，数据库集群中的所有只读副本都会自动重启。
> 从版本：1.2.0.0（2022 年 7 月 21 日）开始，重启主实例不会导致任何副本重启。这意味着，如果您要更改集群参数，则必须单独重启每个实例才能获得参数更改（请参阅参数组）。

使用 Neptune 控制台重启数据库实例

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 选择要重启的数据库实例。

4. 选择实例操作，然后选择重启。
5. 要强制从一个可用区失效转移到另一个可用区，请在重启数据库实例对话框中选择是否通过失效转移重启？。
6. 选择 Reboot。要取消重启，请改为选择取消。

在 Amazon Neptune 中删除数据库实例

您可以随时删除处于任何状态的 Amazon Neptune 数据库实例，前提是实例已启动并对该实例禁用了删除保护。

如果已启用删除保护，则无法删除数据库实例

您只能删除禁用了删除保护的数据库实例。Neptune 强制执行删除保护，无论您是使用控制台、AWS CLI 还是 API 来删除数据库实例。

在使用 AWS Management Console创建生产数据库实例时，将默认启用删除保护。

如果您使用 AWS CLI 或 API 命令创建数据库实例，则默认情况下将禁用删除保护。

要删除已启用删除保护的数据库实例，请先修改实例以将其 `DeletionProtection` 字段设置为 `false`。

启用或禁用删除保护不会导致中断。

在删除数据库实例之前为其拍摄最终快照

要删除数据库实例，您必须指定该实例的名称以及是否要为该实例拍摄一个最终的数据库快照。如果要删除的数据库实例的状态为正在创建，则无法拍摄最终数据库快照。如果数据库实例处于状态为失败、不兼容恢复或不兼容网络的失败状态，则只能在 `SkipFinalSnapshot` 参数设置为 `true` 时删除该实例。

如果您使用 AWS Management Console删除数据库集群中的所有 Neptune 数据库实例，则会自动删除整个数据库集群。如果您使用的是 AWS CLI 或开发工具包，则必须在删除最后一个实例后手动删除数据库集群。

> **Important**
>
> 如果您删除整个数据库集群，则会同时删除其所有自动备份，并且无法恢复。这意味着，除非您选择手动创建最终数据库快照，否则您以后无法将数据库实例还原到其最终状态。在删除集群时，不会删除实例的手动快照。

如果要删除的数据库实例有一个只读副本，应提升该只读副本或将其删除。

在以下示例中，介绍了如何删除一个数据库实例，包括有最终数据库快照和无最终数据库快照两种情况。

删除无最终快照的数据库实例

如果要快速删除数据库实例，则可以跳过创建最终数据库快照的步骤。删除数据库实例时，会删除所有的自动备份，且无法恢复。手动快照不会删除。

使用 Neptune 控制台删除数据库实例且不制作最终数据库快照

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 在实例列表中，选择要删除的数据库实例旁边的单选按钮。
4. 选择实例操作，然后选择删除。
5. 在是否创建最终快照? 框中选择否。
6. 选择 Delete (删除)。

删除有最终快照的数据库实例

如果希望以后能还原所删除的数据库实例，您可以创建最终数据库快照。所有自动备份也会被删除，且无法恢复。手动快照不会删除。

使用 Neptune 控制台删除数据库实例但制作最终数据库快照

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 在实例列表中，选择要删除的数据库实例旁边的单选按钮。
4. 选择实例操作，然后选择删除。
5. 在是否创建最终快照? 框中选择是。
6. 在 Final snapshot name (最终快照名称) 框中，键入最终数据库快照的名称。
7. 选择 Delete。

您可以检查实例的运行状况，确定实例的类型，查明当前安装的引擎版本，并使用实例状态 API 获取有关实例的其他信息。

Amazon Neptune 无服务器

Amazon Neptune 无服务器是一种按需自动扩缩配置，旨在根据需要扩展数据库集群，以满足处理需求的大幅增长，然后在需求减少时再次缩减。它有助于自动执行旨在监控工作负载和调整 Neptune 数据库容量的过程。因为容量是根据应用程序的需求自动调整的，所以您只需为应用程序实际需要的资源付费。

Neptune 无服务器的应用场景

Neptune 无服务器支持很多类型的工作负载。它适用于要求苛刻、高度可变的工作负载，如果您的数据库使用量通常在短时间内很高，然后是长时间的轻度活动或根本没有活动，它会非常有用。Neptune 无服务器对于以下应用场景特别有用：

- 可变工作负载 – CPU 活动突然出现无法预测的增加的工作负载。有了 Neptune 无服务器，您的图形数据库可自动扩展容量以满足工作负载需求，然后在活动涌现结束时再缩减。您不再需要为峰值或平均容量进行预调配。您可以指定容量上限来应对峰值工作负载，除非确实需要该容量，否则不使用该容量。

  Neptune 无服务器提供的扩展粒度帮助您使容量与工作负载的需求紧密匹配。Neptune 无服务器可以根据需要，以细粒度增量添加或移除容量。当只需要多一点容量时，它只需要增加半个 Neptune 容量单位 (NCU)。
- 多租户应用程序 – 通过利用 Neptune 无服务器，您可以为需要运行的每个应用程序创建单独的数据库集群，而不必单独管理这些租户集群。根据多种因素，每个租户集群可能有不同的繁忙和闲置期，但是 Neptune 无服务器可以在没有您干预的情况下高效地扩展它们。
- 新应用程序 - 部署新应用程序时，通常不确定它需要多少数据库容量。使用 Neptune 无服务器，您可以设置一个数据库集群，该集群可以自动扩展以满足新应用程序开发时的容量需求。
- 容量规划 – 假设您通常通过修改集群中所有数据库实例的数据库实例类来调整数据库容量，或者验证工作负载的最佳数据库容量。借助 Neptune 无服务器，您可以避免产生此管理开销。相反，您无需创建新的数据库集群或实例，即可将现有数据库实例从预调配改为无服务器，或者从无服务器修改为预调配。
- 开发和测试 – Neptune 无服务器也非常适合开发和测试环境。借助 Neptune 无服务器，您可以创建最大容量足够高的数据库实例来测试要求最苛刻的应用程序，而对于在测试之间系统可能处于空闲状态的所有其它时间，则可以创建最小容量较低的数据库实例。

Neptune 无服务器只能扩展计算容量。您的存储量保持不变，并且不受无服务器扩展的影响。

> **Note**
>
> 还可以使用 Neptune 自动扩缩和 Neptune 无服务器来处理不同类型的工作负载变化。

Amazon Neptune 无服务器约束

- 并非在所有区域都可用 – Neptune 无服务器仅在以下区域可用：
  - 美国东部（弗吉尼亚州北部）：us-east-1
  - 美国东部（俄亥俄州）：us-east-2
  - 美国西部（北加利福尼亚）：us-west-1
  - 美国西部（俄勒冈州）：us-west-2
  - 加拿大（中部）：ca-central-1
  - 欧洲地区（斯德哥尔摩）：eu-north-1
  - 欧洲地区（爱尔兰）：eu-west-1
  - 欧洲地区（伦敦）：eu-west-2
  - 欧洲地区（法兰克福）：eu-central-1
  - 亚太地区（东京）：ap-northeast-1
  - 亚太地区（新加坡）：ap-southeast-1
  - 亚太地区（悉尼）：ap-southeast-2
- 在早期引擎版本中不可用 – Neptune 无服务器仅在引擎版本 1.2.0.1 或更高版本中可用。
- 与 Neptune 查找缓存不兼容 – 查找缓存不适用于无服务器数据库实例。
- 无服务器实例中的最大内存为 256GB – 将 MaxCapacity 设置为 128 个 NCU（支持的最高设置）允许 Neptune 无服务器实例扩展到 256GB 的内存，相当于 R6g.8XL 预调配实例类型的内存。

Neptune 无服务器数据库集群中的容量扩展

设置 Neptune 无服务器数据库集群与设置普通预调配集群类似，需要对最小和最大扩展单位进行额外配置，并将实例类型设置为 db.serverless。扩展配置以 Neptune 容量单位 (NCU) 定义，每个容量单位由 2GiB 内存 (RAM) 以及相关的虚拟处理器容量 (vCPU) 和网络组成。它设置为 ServerlessV2ScalingConfiguration 对象的一部分，用 JSON 表示，如下所示：

```
"ServerlessV2ScalingConfiguration": {
```

```
  "MinCapacity": (minimum NCUs, a floating-point number such as   1.0),
  "MaxCapacity": (maximum NCUs, a floating-point number such as 128.0)
}
```

在任何时候，每个 Neptune 写入器或读取器实例都有一个由浮点数衡量的容量，该浮点数表示该实例当前使用的 NCU 数量。您可以使用实例级别的 CloudWatch ServerlessDatabase容量指标来了解给定数据库实例当前正在使用多少 NCU，并使用 ncuU ssionation 指标来了解该实例正在使用的最大容量占其最大容量的百分比。这两个指标也适用于数据库集群级，用于显示整个数据库集群的平均资源利用率。

创建 Neptune 无服务器数据库集群时，您可以为所有无服务器实例设置最小和最大 Neptune 容量单位 (NCU) 数量。

您指定的最小 NCU 值可设置数据库集群中无服务器实例可以缩小到的最小大小，同样，最大 NCU 值确定了无服务器实例可以增长到的最大大小。您可以设置的最高的最大 NCU 值为 128.0 NCU，最低的最小值为 1.0 NCU。

Neptune 通过监控 CPU、内存和网络等资源的利用率，持续跟踪每个 Neptune 无服务器实例上的负载。负载由应用程序的数据库操作、服务器的后台处理以及其它管理任务产生。

当无服务器实例上的负载达到当前容量限制时，或者当 Neptune 检测到任何其它性能问题时，该实例会自动扩展。当实例上的负载下降时，容量会缩减到配置的最小容量单位，CPU 容量将在内存之前释放。这种架构允许以可控的逐步减少的方式释放资源，并有效地处理需求波动。

您可以将读取器实例与写入器实例一起扩展，也可以通过设置其提升层来独立扩展。提升层 0 和 1 中的读取器实例与写入器同时扩展，这使它们的大小保持在适当的容量，以便在失效转移情况下快速接管来自写入器的工作负载。提升层 2 到 15 的读取器独立于写入器实例且也独立于彼此进行扩展。

如果您将 Neptune 数据库集群创建为多可用区集群以确保高可用性，则 Neptune 无服务器会根据数据库负载纵向扩展和缩减所有可用区中的实例。您可以将辅助可用区中读取器实例的提升层设置为 0 或 1，这样它就可以随主可用区中写入器实例的容量一起纵向扩展和缩减，以便它随时可以接管当前的工作负载。

> **Note**
>
> Neptune 数据库集群的存储由您所有数据的六个副本组成，分布在三个可用区中，无论您是否将集群创建为多可用区集群。存储复制由存储子系统处理，不受 Neptune 无服务器的影响。

为 Neptune 无服务器数据库集群选择最小容量值

您可以为最小容量设置的最小值为 1.0 NCU。

请务必不要将最小值设置为低于应用程序高效运行所需的值。将其设置得过低可能导致某些内存密集型工作负载的超时率更高。

将最小值设置得尽可能低可以节省资金，因为当需求较低时，您的集群将使用最少的资源。但是，如果您的工作负载往往会大幅波动，从非常低到非常高，则可能需要将最小值设置为较高，因为较高的最小值可以使您的 Neptune 无服务器实例更快地纵向扩展。

其原因是 Neptune 根据当前容量选择扩展增量。如果当前容量较低，Neptune 最初将缓慢纵向扩展。如果最小值较高，Neptune 从较大的扩展增量开始，因此可以更快地纵向扩展，以应对工作负载的突然大幅增加。

为 Neptune 无服务器数据库集群选择最大容量值

您可以为最大容量设置的最大值是 128.0 NCU，可以为最大容量设置的最小值是 2.5 NCU。无论您设置的最大容量值是多少，都必须至少与您设置的最小容量值一样大。

通常，请将最大值设置得足够高，以应对应用程序可能遇到的峰值负载。将其设置得过低可能导致某些内存密集型工作负载的超时率更高。

将最大值设置得尽可能高的好处是，即使是最意想不到的工作负载，您的应用程序也可能能够处理。缺点是您失去了一些预测和控制资源成本的能力。需求的意外激增最终可能会使成本远远超过您预期的预算。

精心设定的最大值的好处是，它可以让您满足峰值需求，同时也为 Neptune 计算成本设定了上限。

> **Note**
>
> 更改 Neptune 无服务器数据库集群的容量范围，会导致某些配置参数的默认值发生更改。Neptune 可以立即应用其中一些新的默认值，但是某些动态参数更改只有在重启后才会生效。pending-reboot 状态表示需要重启才能应用一些参数更改。

使用您的现有配置来估算无服务器需求

如果您通常修改预调配数据库实例的数据库实例类以满足异常高或异常低的工作负载，则可以利用这个经验粗略估计等效的 Neptune 无服务器容量范围。

估计最佳最小容量设置

您可以应用您对现有 Neptune 数据库集群的了解来估算最有效的无服务器最小容量设置。

例如，如果您的预调配工作负载的内存要求对于 T3 或 T4g 等小型数据库实例类而言过高，请选择可提供与 R5 或 R6g 数据库实例类的内存相当的最低 NCU 设置。

或者，假设您在集群的工作负载较低时使用 db.r6g.xlarge 数据库实例类。该数据库实例类具有 32GiB 的内存，因此，您可以将最小 NCU 设置指定为 16，从而创建可以缩减至大约是这个相同容量的无服务器实例（每个 NCU 对应于大约 2GiB 的内存）。如果 db.r6g.xlarge 实例有时未得以充分利用，您可以指定一个较低的值。

如果您的应用程序在数据库实例可以在内存或缓冲区缓存中保存给定数量的数据时工作效率最高，请考虑指定一个足够大的最小 NCU 设置，以便为此类情况提供足够的内存。否则，当无服务器实例缩减时，数据可能会被从缓冲区缓存中逐出，并且当实例再次纵向扩展时，数据必须随着时间的推移读回到缓冲区缓存。如果将数据带回缓冲区缓存的 I/O 量很大，那么选择更高的最小 NCU 值可能是值得的。

如果您发现您的无服务器实例大部分时间都在特定容量下运行，则最好将最小容量设置为略低于该容量。如果当前容量没有远低于所需容量，则 Neptune 无服务器可以高效地估计纵向扩展的规模和速度。

在混合配置中，使用预调配的写入器和 Neptune 无服务器读取器，读取器不会随写入器一起扩展。由于它们是独立扩展的，因此为它们设置较低的最小容量会导致复制滞后过大。当写入密集度很高的工作负载时，它们可能没有足够的容量来跟上写入器所做的更改。在这种情况下，请设置与写入器容量相当的最小容量。尤其是，如果在提升层 2-15 的读取器中观察到复制滞后，请增大集群的最小容量设置。

估计最佳的最大容量设置

您还可以应用您对现有 Neptune 数据库集群的了解来估算最有效的无服务器最大容量设置。

例如，假设您在集群的工作负载较高时使用 db.r6g.4xlarge 数据库实例类。该数据库实例类有 128GiB 的内存，因此您可以将最大 NCU 设置指定为 64，以设置等效的 Neptune 无服务器实例（每个 NCU 对应大约 2GiB 的内存）。如果 db.r6g.4xlarge 实例无法始终处理工作负载，您可以指定一个更高的值，以便进一步纵向扩展数据库实例。

如果您的工作负载很少出现意想不到的峰值，那么将最大容量设置得足够高，以便即使在这些峰值期间也能保持应用程序性能，这可能是有意义的。另一方面，您可能需要设置较低的最大容量，这样可以在异常峰值期间降低吞吐量，但这样可以让 Neptune 毫无问题地处理预期的工作负载，从而限制成本。

Neptune 无服务器数据库集群和实例的其它配置

除了为 Neptune 无服务器数据库集群设置最小和最大容量外，还有其它一些配置选项需要考虑。

在数据库集群中组合无服务器实例和预调配实例

数据库集群不必仅是无服务器的，您可以创建无服务器实例和预调配实例的组合（混合配置）。

例如，假设您需要的写容量比无服务器实例中可用的写容量更多。在这种情况下，您可以使用非常大的预调配写入器来设置集群，但对于读取器仍然使用无服务器实例。

或者，假设集群上的写入工作负载会变化，但读取工作负载保持稳定。在这种情况下，您可以为集群设置一个无服务器写入器和一个或多个预调配读取器。

有关如何创建混合配置数据库集群的信息，请参阅使用 Amazon Neptune 无服务器。

为 Neptune 无服务器实例设置提升层

对于包含多个无服务器实例或混用预调配实例和无服务器实例的集群，请注意每个无服务器实例的提升层设置。此设置控制的无服务器数据库实例行为比预调配数据库实例更多。

在中 AWS Management Console，您可以使用 “创建数据库”、“修改实例” 和 “添加读者” 页面上的 “其他配置” 下的 “故障转移” 优先级来指定此设置。您可以在数据库页面上可选的优先级层中看到现有实例的这个属性。还可以在数据库集群或实例的详细信息页面上看到此属性。

对于预调配实例，0–15 层的选择仅决定在失效转移操作期间 Neptune 选择将读取器实例提升为写入器的顺序。对于 Neptune 无服务器读取器实例，层编号还决定实例是纵向扩展以匹配写入器实例的容量，还是仅根据自己的工作负载独立于写入器实例的容量进行扩展。

第 0 层或第 1 层的 Neptune 无服务器读取器实例的最小容量至少与写入器实例一样高，这样它们就可以在失效转移时从写入器手中接管。如果写入器是预调配实例，Neptune 估计等效的无服务器容量，并将该估计值用作无服务器读取器实例的最小容量。

第 2-15 层中的 Neptune 无服务器读取器实例对其最小容量没有这样的限制，并且独立于写入器进行扩展。当它们处于空闲状态时，它们缩减到在集群的容量范围中指定的最小 NCU 值。但是，如果读取工作负载迅速激增，这可能会导致问题。

使读取器容量与写入器容量保持一致

要记住的一件重要事情是，您要确保读取器实例能够跟上写入器实例的步伐，以防止复制延迟过大。在以下两种情况下，这尤其令人担忧，其中无服务器读取器实例不会自动横向缩减以与写入器实例同步：

- 当您的写入器为预调配，而读取器为无服务器时。
- 当写入器为无服务器，而无服务器读取器处于提升层 2-15 时。

在这两种情况下，都应将最小无服务器容量设置为与预期的写入器容量相匹配，以确保读取器操作不会超时和可能导致重启。对于预调配的写入器实例，请将最小容量设置为与预调配实例的最小容量相匹配。对于无服务器写入器，可能更难预测最佳设置。

由于实例容量范围是在集群级设置的，因此所有无服务器实例都由相同的最小和最大容量设置控制。第 0 层和第 1 层中的读取器实例的扩展与写入器实例同步，但提升层 2-15 中的实例彼此独立且独立于写入器实例进行扩展，具体取决于它们的工作负载。如果您将最小容量设置得过低，则第 2 层至第 15 层的空闲实例可能会缩减得太低，无法以足够快的速度再次纵向扩展以应对写入器活动的突然激增。

避免将超时值设置得过高

如果您在无服务器实例上设置的查询超时值过高，可能会产生意想不到的成本。

如果没有合理的超时设置，您可能会无意中发出一个查询，该查询需要强大、昂贵的实例类型，并且保持运行很长时间，从而产生您意想不到的成本。您可以通过使用查询超时值来避免这种情况，该值可以容纳您的大多数查询，并且只会导致长时间运行的查询意外超时。

对于使用参数设置的常规查询超时值和使用查询提示设置的每个查询的超时值，都是如此。

优化 Neptune 无服务器配置

如果 Neptune 无服务器数据库集群未根据其正在运行的工作负载进行调整，您可能会注意到它没有以最佳方式运行。您可以调整最小和/或最大容量设置，使其可以在不遇到内存问题的情况下进行扩展。

- 增加集群的最小容量设置。这可以纠正空闲实例缩减到内存少于应用程序和所启用的特征所需容量的情况。
- 增加集群的最大容量设置。这可以纠正繁忙的数据库无法纵向扩展到具有足够内存的容量来处理工作负载和任何启用的内存密集型特征的情况。
- 更改相关实例上的工作负载。例如，您可以将读取器数据库实例添加到集群中，以便将读取负载分布到更多数实例中。
- 调整应用程序的查询，使其使用更少的资源。
- 尝试使用大于 Neptune 无服务器中可用的最大 NCU 的预调配实例，看看它是否更适合工作负载的内存和 CPU 要求。

使用 Amazon Neptune 无服务器

您可以将新的 Neptune 数据库集群创建为无服务器集群，或者在某些情况下，可以将现有数据库集群转换为使用无服务器。也可以将无服务器数据库集群中的数据库实例与无服务器实例相互转换。你只能在支持 Neptune Serverless AWS 区域 的地方使用 Neptune Serverless，还有其他一些限制（参见）。Amazon Neptune 无服务器约束

也可以使用 Neptune AWS CloudFormation 堆栈创建 Neptune 无服务器数据库集群。

创建使用无服务器的新数据库集群

要创建使用无服务器的 Neptune 数据库集群，您可以通过 AWS Management Console，使用与创建预调配集群相同的方法来创建。不同之处在于，在数据库实例大小下，您需要将数据库实例类设置为无服务器。当您这样做时，您需要为集群设置无服务器容量范围。

您也可以使用如下所 AWS CLI 示的 with 命令创建无服务器数据库集群（在 Windows 上，将 '\' 替换为 '^'）：

```
aws neptune create-db-cluster \
  --region (an AWS ## region that supports serverless) \
  --db-cluster-identifier (ID for the new serverless DB cluster) \
  --engine neptune \
  --engine-version (optional: 1.2.0.1 or above) \
  --serverless-v2-scaling-configuration "MinCapacity=1.0, MaxCapacity=128.0"
```

也可以指定 `serverless-v2-scaling-configuration` 参数，如下所示：

```
  --serverless-v2-scaling-configuration '{"MinCapacity":1.0, "MaxCapacity":128.0}'
```

然后，您可以对 `ServerlessV2ScalingConfiguration` 属性运行 `describe-db-clusters` 命令，该命令应返回您指定的容量范围设置：

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": (the specified minimum number of NCUs),
    "MaxCapacity": (the specified maximum number of NCUs)
}
```

将现有数据库集群或实例转换为无服务器

如果您的 Neptune 数据库集群使用引擎版本 1.2.0.1 或更高版本，则可以将其转换为无服务器。此过程确实会导致一些停机时间。

第一步是向现有集群添加容量范围。你可以使用或使用这样的 AWS CLI 命令来做到这一点（在 Windows 上，用 '^' 替换 '\'）： AWS Management Console

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (your DB cluster ID) \
  --serverless-v2-scaling-configuration \
      MinCapacity=(minimum number of NCUs, such as  2.0), \
      MaxCapacity=(maximum number of NCUs, such as 24.0)
```

下一步是创建一个新的无服务器数据库实例，以替换集群中现有的主实例（写入器）。同样，您可以使用 AWS Management Console 或来执行此操作以及所有后续步骤 AWS CLI。在任一种情况下，将数据库实例类指定为无服务器。 AWS CLI 命令看起来像这样（在 Windows 上，将 '\' 替换为 '^'）：

```
aws neptune create-db-instance \
  --db-instance-identifier (an instance ID for the new writer instance) \
  --db-cluster-identifier (ID of the DB cluster) \
  --db-instance-class db.serverless
  --engine neptune
```

当新的写入器实例变为可用时，执行失效转移以使其成为集群的写入器实例：

```
aws neptune failover-db-cluster \
  --db-cluster-identifier (ID of the DB cluster) \
  --target-db-instance-identifier (instance ID of the new serverless instance)
```

接下来，删除旧的写入器实例：

```
aws neptune delete-db-instance \
  --db-instance-identifier (instance ID of the old writer instance) \
  --skip-final-snapshot
```

最后，执行同样的操作，创建一个新的无服务器实例，以取代您想要变为无服务器实例的每个现有预调配读取器实例，然后删除现有的预调配实例（读取器实例不需要失效转移）。

修改现有无服务器数据库集群的容量范围

您可以像这样使用 AWS CLI 来更改 Neptune 无服务器数据库集群的容量范围（在 Windows 上，将 '\' 替换为 '^'）：

```
aws neptune modify-db-cluster \
  --region (an AWS region that supports serverless) \
  --db-cluster-identifier (ID of the serverless DB cluster) \
  --apply-immediately \
  --serverless-v2-scaling-configuration MinCapacity=4.0, MaxCapacity=32
```

更改容量范围会导致某些配置参数的默认值发生更改。Neptune 可以立即应用其中一些新的默认值，但是某些动态参数更改只有在重启后才会生效。pending-reboot 状态表示需要重启才能应用一些参数更改。

将无服务器数据库实例更改为预调配

要将 Neptune 无服务器实例转换为预调配实例，您只需将其实例类更改为预调配的实例类之一。请参阅 修改 Neptune 数据库实例（并立即应用）。

使用 Amazon 监控无服务器容量 CloudWatch

您可以使用 CloudWatch 来监控数据库集群中 Neptune 无服务器实例的容量和利用率。有两个 CloudWatch 指标可以让您跟踪集群级别和实例级别的当前无服务器容量：

- **ServerlessDatabaseCapacity** – 作为实例级指标，ServerlessDatabaseCapacity 报告当前实例容量（以 NCU 为单位）。作为集群级指标，它报告集群中所有数据库实例的所有 ServerlessDatabaseCapacity 值的平均值。
- **NCUUtilization** – 此指标报告可能使用的容量百分比。计算方法为：当前容量 ServerlessDatabaseCapacity（实例级或集群级）除以数据库集群的最大容量设置。

  如果该指标在集群级接近 100%，这意味着集群已尽可能高地扩展，则可以考虑增加最大容量设置。

  如果读取器实例的容量接近 100%，而写入器实例的容量未接近最大容量，则可以考虑添加更多读取器实例来分配读取工作负载。

请注意，对于无服务器实例，CPUUtilization 和 FreeableMemory 指标的含义与对于预调配实例的含义略有不同。在无服务器环境中，CPUUtilization 是一个百分比，其计算方法是当前使用的

CPU 量除以在最大容量基础上提供的 CPU 量。同样，FreeableMemory 报告实例达到最大容量时可用的可释放内存量。

以下示例说明如何 AWS CLI 在 Linux 上使用来检索给定数据库实例的最小、最大和平均容量值，这些值在一小时内每 10 分钟测量一次。Linux date 命令指定相对于当前日期和时间的开始时间和结束时间。--query 参数中的 sort_by 函数根据 Timestamp 字段按时间顺序对结果进行排序：

```
aws cloudwatch get-metric-statistics \
  --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" \
  --end-time "$(date -d 'now')" \
  --period 600 \
  --namespace "AWS/Neptune"
  --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=(instance ID) \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' \
  --output table
```

使用 Neptune Streams 实时捕获图形更改

Neptune Streams 以完全托管式方式按发生顺序即时记录对图形所做的每一个更改。启用 Streams 后，Neptune 将负责管理可用性、备份、安全性和到期等事宜。

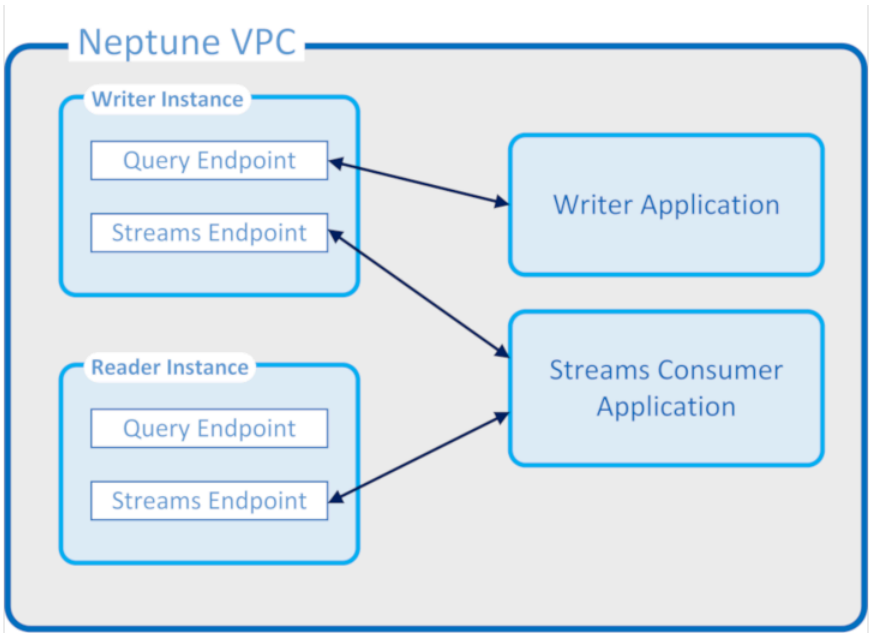
> **Note**
>
> 该特征从版本 1.0.1.0.200463.0（2019 年 10 月 15 日）开始在实验室模式下可用，从 Neptune 引擎版本 1.0.2.2.R2 开始可用于生产用途。

在许多使用案例中，都可能需要即时捕获对图的更改，下面是一些示例：

- 您可能希望在发生某些更改时应用程序能够自动通知用户。
- 您可能还想在其他数据存储中维护图表数据的最新版本，例如亚马逊 OpenSearch 服务、亚马逊或亚马逊简单存储服务 (Amazon S3) Simple Storage Service。 ElastiCache

Neptune 对更改日志流使用与图形数据相同的原生存储。它随进行更改的事务同步写入更改日志条目。您可以使用 HTTP REST API 从日志流检索这些更改记录。（有关信息，请参阅 调用 Streams API。）

下图显示了如何从 Neptune Streams 检索更改日志数据。



Neptune Streams 保证

- 事务一旦完成，其所做的更改将立即可供写入器和读取器读取（读取器仍有正常的复制滞后）。
- 更改记录严格按照发生顺序显示（这包括在事务中进行的更改）。
- 更改流不包含重复项。每个更改仅记录一次。
- 更改流是完整的。不会丢失或忽略任何更改。
- 更改流包含确定数据库本身在任意时间点的完整状态所需的全部信息，前提是起始状态已知。
- 可以随时打开或关闭 Streams。

Neptune Streams 操作属性

- 更改日志流是完全托管的。
- 更改日志数据作为进行更改的同一事务的一部分同步写入。
- 启用 Neptune Streams 后，将产生与更改日志数据相关的 I/O 和存储费用。
- 默认情况下，更改记录在创建一周后自动清除。从引擎版本 1.2.0.0 开始，可以使用 neptune_streams_expiry_days 数据库参数将此保留期更改为 1 到 90 之间的任意天数。
- 流的读取性能随实例扩展。
- 您可以使用只读副本实现高可用性和高读取吞吐量。您可以并发创建和使用任意数量的流读取器。
- 更改日志数据在多个可用区之间复制，因此具有高持久性。
- 日志数据与图形数据本身一样安全。可以在静态和传输过程中加密。可以使用 IAM、Amazon VPC 和 AWS Key Management Service (AWS KMS) 控制访问权限。与图表数据一样，可以对其进行备份，然后使用恢复 (PITR) 进行 point-in-time 恢复。
- 流数据作为每个事务的一部分同步写入，因此会导致总体写入性能稍微下降。
- 流数据不进行分片，因为 Neptune 在设计上是单分片的。
- 日志流 `GetRecords` API 使用的资源与所有其它 Neptune 图形操作相同。这意味着客户端需要在流请求和其他数据库请求之间进行负载平衡。
- 禁用流后，所有日志数据将立即变得不可访问。这意味着您必须在禁用日志记录之前读取所有感兴趣的日志数据。
- 目前没有与的本机集成 AWS Lambda。日志流不会生成可触发 Lambda 函数的事件。

主题

- 使用 Neptune Streams

- Neptune Streams 中的序列化格式
- Neptune Streams 示例
- 使用在 AWS CloudFormation Streams 使用者应用程序中设置海王星到海王星的复制
- 使用 Neptune 流跨区域复制进行灾难恢复

使用 Neptune Streams

借助 Neptune Streams 特征，您可以生成完整的更改日志条目序列，以即时记录对图形数据所做的每一个更改。有关该功能的概述，请参阅 使用 Neptune Streams 实时捕获图形更改。

主题

- 启用 Neptune Streams
- 禁用 Neptune Streams
- 调用 Neptune Streams REST API
- Neptune Streams API 响应格式
- Neptune Streams API 异常

启用 Neptune Streams

您可以通过设置 `neptune_streams` 数据库集群参数随时启用或禁用 Neptune Streams。将参数设置为 `1` 会启用流，将其设置为 `0` 会禁用流。

> **Note**
>
> 更改 `neptune_streams` 数据库集群参数后，您必须重启集群中的所有数据库实例才能使更改生效。

您可以设置 `neptune_streams_expiry_days` 数据库集群参数，以控制流记录在删除之前保留在服务器上的天数（从 1 到 90）。默认值为 7。

Neptune Streams 最初是作为一项实验性特征引入的，您可以在实验室模式下使用数据库集群 `neptune_lab_mode` 参数启用或禁用该特征（请参阅Neptune 实验室模式）。使用实验室模式启用流的功能现在已弃用，未来会禁用。

禁用 Neptune Streams

您随时可以关闭正在运行的 Neptune Streams。

要关闭流，请更新数据库集群参数组，以便将 neptune_streams 参数的值设置为 0。

> **Important**
>
> 关闭 Streams 后，您无法再访问更改日志数据。请确保在关闭 Streams 前读取您感兴趣的内容。

调用 Neptune Streams REST API

您可以使用 REST API 访问 Neptune Streams，该 API 将向以下其中一个本地端点发送 HTTP GET 请求：

- 对于 SPARQL 图形数据库：`https://Neptune-DNS:8182/sparql/stream`。
- 对于 Gremlin 或 openCypher 图形数据库：`https://Neptune-DNS:8182/propertygraph/stream` 或 `https://Neptune-DNS:8182/pg/stream`。

> **Note**
>
> 从引擎版本 1.1.0.0 开始，Gremlin 流端点 (`https://Neptune-DNS:8182/gremlin/stream`) 及其关联的输出格式 (GREMLIN_JSON) 已被弃用。为了向后兼容，它仍然受支持，但可能会在将来版本中移除。

仅允许进行 HTTP GET 操作。

如果 HTTP 请求包含 Accept-Encoding 标头并将 gzip 指定为可接受的压缩格式（即 "Accept-Encoding: gzip"），则 Neptune 支持对响应进行 gzip 压缩。

参数

- limit – long，可选。范围：1–100000。默认值：10。

  指定要返回的最大记录数。响应还受 10 MB 大小的限制，该大小限制不能修改，并且优先级高于 limit 参数中指定的记录数。如果达到了 10 MB 限制，响应中将包含一条超出阈值记录。

- iteratorType – 字符串，可选。

  该参数可接受以下值：

  - AT_SEQUENCE_NUMBER（默认值）– 指示读取应该从由 commitNum 和 opNum 参数共同指定的事件序列号开始。
  - AFTER_SEQUENCE_NUMBER – 指示读取应该从紧接在由 commitNum 和 opNum 参数共同指定的事件序列号之后的事件序列号开始。
  - TRIM_HORIZON – 指示读取应该从系统中最后一条未剪裁的记录开始，该记录是更改日志流中时间最久的未过期（尚未删除）记录。当您没有特定的开始事件序列号时，该模式在应用程序启动期间非常有用。
  - LATEST – 指示读取应该从系统中最近的记录开始，该记录是更改日志流中最新的未过期（尚未删除）记录。当需要从当前的流顶部读取记录以免处理较旧的记录时，例如在灾难恢复或零停机时间升级期间，这很有用。请注意，在此模式下，最多只返回一条记录。
- commitNum – long，当 iteratorType 为 AT_SEQUENCE_NUMBER 或 AFTER_SEQUENCE_NUMBER 时是必需的。

  从更改日志流中读取的起始记录的提交编号。

  如果 iteratorType 为 TRIM_HORIZON 或 LATEST，则忽略此参数。
- opNum – long，可选（默认值为 1）。

  从更改日志流数据中开始读取的指定提交中的操作序列号。

对于更改 SPARQL 图形数据的操作，通常是每个操作仅生成一条更改记录。但是，更改 Gremlin 图形数据的操作可以是每个操作生成多条更改记录，如以下示例所示：

- INSERT – Gremlin 顶点可以有多个标签，而 Gremlin 元素可以有多个属性。插入元素时，将为每个标签和属性生成单独的更改记录。
- UPDATE – 更改 Gremlin 元素属性时，将生成两条更改记录：第一条为删除先前值的记录，第二条为插入新值的记录。
- DELETE – 为每个删除的元素属性生成一条单独的更改记录。例如，当删除具有属性的 Gremlin 边缘时，将为每个属性生成一条更改记录，然后，生成一条删除边缘标签的更改记录。

  删除 Gremlin 顶点时，将首先删除所有入边和出边属性，然后删除边缘标签、顶点属性，最后是顶点标签。上述每个删除操作都会生成一条更改记录。

Neptune Streams API 响应格式

Neptune Streams REST API 请求的响应包含以下字段：

- `lastEventId` – 流响应中最后一次更改的序列标识符。事件 ID 由两个字段组成：`commitNum` 标识更改图形的事务，opNum 标识该事务中的特定操作。如以下示例所示。

```
  "eventId": {
    "commitNum": 12,
    "opNum": 1
  }
```

- `lastTrxTimestamp` – 请求提交事务的时间（使用 Unix 纪元时间表示，单位为毫秒）。
- `format` – 返回的更改记录的序列化格式。对于 Gremlin 或 openCypher 更改记录，可能的值为 `PG_JSON`；对于 SPARQL 更改记录，可能的值为 `NQUADS`。
- `records` – 包含在响应中的序列化更改日志流记录的数组。`records` 数组中的每条记录都包含以下字段：
  - `commitTimestamp` – 请求提交事务的时间（使用 Unix 纪元时间表示，单位为毫秒）。
  - `eventId` – 流更改记录的序列标识符。
  - `data`— 序列化的 Gremlin、SPARQL 或变更记录。 OpenCypher 下一部分Neptune Streams 中的序列化格式中更详细地介绍了每个记录的序列化格式。
  - `op` – 造成更改的操作。
  - `isLastOp` – 仅当此操作是其事务中的最后一个操作时才存在。如果存在，则它设置为 `true`。有助于确保整个事务都被消耗掉。
- `totalRecords` – 响应中的记录总数。

例如，对于包含多个操作的事务，以下响应返回 Gremlin 更改数据：

```
{
  "lastEventId": {
    "commitNum": 12,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
```

```
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD"
    }
  ],
  "totalRecords": 1
}
```

以下响应返回事务中最后一个操作的 SPARQL 更改数据（该操作在事务编号 97 中由 EventId(97, 1) 标识）。

```
{
  "lastEventId": {
    "commitNum": 97,
    "opNum": 1
  },
  "lastTrxTimestamp": 1561489355102,
  "format": "NQUADS",
  "records": [
    {
      "commitTimestamp": 1561489355102,
      "eventId": {
        "commitNum": 97,
        "opNum": 1
      },
      "data": {
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
```

```
  "totalRecords": 1
}
```

Neptune Streams API 异常

下表描述了 Neptune Streams 异常。

错误代码	HTTP 代码	可以重试？	消息
`InvalidParameterException`	400	否	输入参数中提供了无效的或 out-of-range 值。
`ExpiredStreamException`	400	否	所有请求的记录均超出了允许的最长使用期限并且已经过期。
`ThrottlingException`	500	是	请求速率超出吞吐量上限。
`StreamRecordsNotFoundException`	404	否	找不到请求的资源。可能指定了错误的流。
`MemoryLimitExceededException`	500	是	由于缺少内存，请求不成功，但可在服务器较不忙时重试。

Neptune Streams 中的序列化格式

Amazon Neptune 使用两种不同的格式来将图形更改数据序列化到日志流，具体取决于该图形是使用 Gremlin 还是 SPARQL 创建的。

这两种格式共享一种通用的记录序列化格式，如Neptune Streams API 响应格式中所述，该格式包含以下字段：

- `commitTimestamp` – 请求提交事务的时间（使用 Unix 纪元时间表示，单位为毫秒）。
- `eventId` – 流更改记录的序列标识符。

- `data`— 序列化的 Gremlin、SPARQL 或变更记录。 OpenCypher 下面的各部分中更详细地介绍了每个记录的序列化格式。
- `op` – 造成更改的操作。

主题

PG_JSON 更改序列化格式

> **Note**
>
> 从引擎版本 1.1.0.0 开始，Gremlin 流端点 (`https://Neptune-DNS:8182/gremlin/stream`) 输出的 Gremlin 流输出格式 (`GREMLIN_JSON`) 已被弃用。它被 PG_JSON 取代，PG_JSON 当前与 `GREMLIN_JSON` 相同。

日志流响应的 `data` 字段中包含的 Gremlin 或 openCypher 更改记录具有以下字段：

- `id` – 字符串，必需。

  Gremlin 或 openCypher 元素的 ID。
- `type` – 字符串，必需。

  Gremlin 或 openCypher 元素的类型。必须是以下类型之一：
  - `vl` – Gremlin 的顶点标签；openCypher 的节点标签。
  - `vp` – Gremlin 的顶点属性；openCypher 的节点属性。
  - `e` – Gremlin 的边缘和边缘标签；openCypher 的关系和关系类型。
  - `ep` – Gremlin 的边缘属性；openCypher 的关系属性。
- `key` – 字符串，必需。

  属性名称。对于元素标签，此为“label”。
- `value` – `value` 对象，必需。

  这是一个 JSON 对象，其中包含 `value` 字段（存储值本身）和 `datatype` 字段（存储该值的 JSON 数据类型）。

```
  "value": {
    "value": "the new value",
    "dataType": "the JSON datatype of the new value"
  }
```

- `from` – 字符串，可选。

  如果这是一个边缘（类型 = e），则为相应的 from 顶点或源节点的 ID。

- `to` – 字符串，可选。

  如果这是一个边缘（类型 = e），则为相应的 to 顶点或目标节点的 ID。

Gremlin 示例

- 以下是 Gremlin 顶点标签的示例。

```
{
  "id": "an ID string",
  "type": "vl",
  "key": "label",
  "value": {
    "value": "the new value of the vertex label",
    "dataType": "String"
  }
}
```

- 以下是 Gremlin 顶点属性的示例。

```
{
  "id": "an ID string",
  "type": "vp",
  "key": "the property name",
  "value": {
    "value": "the new value of the vertex property",
    "dataType": "the datatype of the vertex property"
  }
}
```

- 以下是 Gremlin 边缘的示例。

```
{
```

```
  "id": "an ID string",
  "type": "e",
  "key": "label",
  "value": {
    "value": "the new value of the edge",
    "dataType": "String"
  },
  "from": "the ID of the corresponding "from" vertex",
  "to": "the ID of the corresponding "to" vertex"
}
```

openCypher 示例

- 以下是 openCypher 节点标签的示例。

```
{
  "id": "an ID string",
  "type": "vl",
  "key": "label",
  "value": {
    "value": "the new value of the node label",
    "dataType": "String"
  }
}
```

- 以下是 openCypher 节点属性的示例。

```
{
  "id": "an ID string",
  "type": "vp",
  "key": "the property name",
  "value": {
    "value": "the new value of the node property",
    "dataType": "the datatype of the node property"
  }
}
```

- 以下是 openCypher 关系的示例。

```
{
  "id": "an ID string",
  "type": "e",
```

```
  "key": "label",
  "value": {
    "value": "the new value of the relationship",
    "dataType": "String"
  },
  "from": "the ID of the corresponding source node",
  "to": "the ID of the corresponding target node"
}
```

SPARQL NQUADS 更改序列化格式

Neptune 使用 W3C RDF 1.1 N-Quads 规范中定义的资源描述框架 (RDF) N-QUADS 语言在图形中记录对 SPARQL 四元组的更改。

更改记录中的 data 字段仅包含一个 stmt 字段，该字段包含表示更改后的四元组的 N-QUADS 语句，如以下示例所示。

```
  "stmt" : "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
```

Neptune Streams 示例

以下示例显示如何访问 Amazon Neptune 中的更改日志流数据。

主题

- AT_SEQUENCE_NUMBER 更改日志
- AFTER_SEQUENCE_NUMBER 更改日志
- TRIM_HORIZON 更改日志
- LATEST 更改日志
- 压缩更改日志

AT_SEQUENCE_NUMBER 更改日志

以下示例显示了 Gremlin 或 openCypher AT_SEQUENCE_NUMBER 更改日志。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq
{
```

```
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "commitTimestamp": 1560011610678,
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

此示例显示 AT_SEQUENCE_NUMBER 更改日志的 SPARQL 示例。

```
curl -s "https://localhost:8182/sparql/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AT_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1571252030566,
  "format": "NQUADS",
  "records": [
    {
      "eventId": {
```

```
        "commitNum": 1,
        "opNum": 1
      },
      "commitTimestamp": 1571252030566,
      "data": {
        "stmt": "<https://test.com/s> <https://test.com/p> <https://test.com/o> .\n"
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

AFTER_SEQUENCE_NUMBER 更改日志

以下示例显示了 Gremlin 或 openCypher AFTER_SEQUENCE_NUMBER 更改日志。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&commitNum=1&opNum=1&iteratorType=AFTER_SEQUENCE_NUMBER" |jq
{
  "lastEventId": {
    "commitNum": 2,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011633768,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011633768,
      "eventId": {
        "commitNum": 2,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
```

```
      "op": "REMOVE",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

TRIM_HORIZON 更改日志

以下示例显示了 Gremlin 或 openCypher TRIM_HORIZON 更改日志。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?
limit=1&iteratorType=TRIM_HORIZON" |jq
{
  "lastEventId": {
    "commitNum": 1,
    "opNum": 1
  },
  "lastTrxTimestamp": 1560011610678,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1560011610678,
      "eventId": {
        "commitNum": 1,
        "opNum": 1
      },
      "data": {
        "id": "d2b59bf8-0d0f-218b-f68b-2aa7b0b1904a",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "vertex",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

LATEST 更改日志

以下示例显示了 Gremlin 或 openCypher LATEST 更改日志。请注意，API 参数 `limit`、`commitNum` 和 opNum 是完全可选的。

```
curl -s "https://Neptune-DNS:8182/propertygraph/stream?iteratorType=LATEST" | jq
{
  "lastEventId": {
    "commitNum": 21,
    "opNum": 4
  },
  "lastTrxTimestamp": 1634710497743,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1634710497743,
      "eventId": {
        "commitNum": 21,
        "opNum": 4
      },
      "data": {
        "id": "24be4e2b-53b9-b195-56ba-3f48fa2b60ac",
        "type": "e",
        "key": "label",
        "value": {
          "value": "created",
          "dataType": "String"
        },
        "from": "4",
        "to": "5"
      },
      "op": "REMOVE",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

压缩更改日志

以下示例显示了 Gremlin 或 openCypher 压缩更改日志。

```
curl -sH \
  "Accept-Encoding: gzip" \
  "https://Neptune-DNS:8182/propertygraph/stream?limit=1&commitNum=1" \
  -H "Accept-Encoding: gzip" \
  -v |gunzip -|jq
> GET /propertygraph/stream?limit=1 HTTP/1.1
> Host: localhost:8182
> User-Agent: curl/7.64.0
> Accept: /
> Accept-Encoding: gzip
*> Accept-Encoding: gzip*
>
< HTTP/1.1 200 OK
< Content-Type: application/json; charset=UTF-8
< Connection: keep-alive
*< content-encoding: gzip*
< content-length: 191
<
{ [191 bytes data]
Connection #0 to host localhost left intact
{
  "lastEventId": "1:1",
  "lastTrxTimestamp": 1558942160603,
  "format": "PG_JSON",
  "records": [
    {
      "commitTimestamp": 1558942160603,
      "eventId": "1:1",
      "data": {
        "id": "v1",
        "type": "vl",
        "key": "label",
        "value": {
          "value": "person",
          "dataType": "String"
        }
      },
      "op": "ADD",
      "isLastOp": true
    }
  ],
  "totalRecords": 1
}
```

使用在 AWS CloudFormation Streams 使用者应用程序中设置海王星到海王星的复制

您可以使用 AWS CloudFormation 模板设置 Neptune 流使用者应用程序，以支持海王星到海王星的复制。

主题

- 为你所在的地区选择一个 AWS CloudFormation 模板
- 添加有关您正在创建的 Neptune 流使用者堆栈的详细信息
- 运行 AWS CloudFormation 模板
- 使用最新的 Lambda 构件更新流轮询器

为你所在的地区选择一个 AWS CloudFormation 模板

要在 AWS CloudFormation 控制台上启动相应的 AWS CloudFormation 堆栈，请根据要使用的 AWS 区域，选择下表中的启动堆栈按钮之一。

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	Launch Stack
美国东部（俄亥俄州）	视图	在 Designer 中查看	Launch Stack
美国西部（加利福尼亚北部）	视图	在 Designer 中查看	Launch Stack
美国西部（俄勒冈州）	视图	在 Designer 中查看	Launch Stack
加拿大（中部）	视图	在 Designer 中查看	Launch Stack
南美洲（圣保罗）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	Launch Stack
欧洲地区（爱尔兰）	视图	在 Designer 中查看	Launch Stack
欧洲地区（伦敦）	视图	在 Designer 中查看	Launch Stack
欧洲地区（巴黎）	视图	在 Designer 中查看	Launch Stack
欧洲地区（法兰克福）	视图	在 Designer 中查看	Launch Stack
中东（巴林）	视图	在 Designer 中查看	Launch Stack
中东（阿联酋）	视图	在 Designer 中查看	Launch Stack
以色列（特拉维夫）	视图	在 Designer 中查看	Launch Stack
非洲（开普敦）	视图	在 Designer 中查看	Launch Stack
Asia Pacific (Tokyo)	视图	在 Designer 中查看	Launch Stack
亚太地区（香港）	视图	在 Designer 中查看	Launch Stack
亚太地区（首尔）	视图	在 Designer 中查看	Launch Stack
亚太地区（新加坡）	视图	在 Designer 中查看	Launch Stack
亚太地区（悉尼）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
亚太地区（孟买）	视图	在 Designer 中查看	Launch Stack
中国（北京）	视图	在 Designer 中查看	Launch Stack
中国（宁夏）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国西部）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国东部）	视图	在 Designer 中查看	Launch Stack

在 Create stack (创建堆栈) 主页上，选择 Next (下一步)。

添加有关您正在创建的 Neptune 流使用者堆栈的详细信息

Specify Stack Details (指定堆栈详细信息) 页面提供了属性和参数，可用于控制应用程序的设置。

堆栈名称-您正在创建的新 AWS CloudFormation 堆栈的名称。通常可以使用默认值 `NeptuneStreamPoller`。

在 Parameters (参数) 下，提供以下内容：

流使用者运行的 VPC 的网络配置

- **`VPC`** – 提供将运行轮询 Lambda 函数的 VPC 的名称。
- **`SubnetIDs`** – 与其建立网络接口的子网。添加与您的 Neptune 集群对应的子网。
- **`SecurityGroupIds`** – 提供向源 Neptune 数据库集群授予写入入站访问权限的安全组的 ID。
- **`RouteTableIds`** – 如果您还没有 Amazon DynamoDB 端点，则需要在 Neptune VPC 中创建一个。您必须提供与子网关联的路由表 ID 的逗号分隔列表。
- **`CreateDDBVPCEndPoint`** – 一个布尔值，默认为 `true`，表示是否需要创建 DynamoDB VPC 端点。如果您已在 VPC 中创建 DynamoDB 端点，则只需将其更改为 `false`。
- **`CreateMonitoringEndPoint`** – 一个布尔值，默认为 `true`，表示是否需要创建监控 VPC 端点。如果您已在 VPC 中创建镜像终端节点，则只需将其更改为 `false`。

流轮询器

- **ApplicationName** – 您通常可以将此设置保留为默认值 (NeptuneStream)。如果您使用不同的名称，则该名称必须是唯一的。
- **LambdaMemorySize** – 用于设置 Lambda 轮询器函数可用的内存大小。默认值为 2,048 MB。
- **LambdaRuntime** – 从 Neptune 流中检索项目的 Lambda 函数使用的语言。您可以将其设置为 `python3.9` 或 `java8`。
- **LambdaS3Bucket** – 包含 Lambda 代码构件的 Amazon S3 桶。除非您使用从其它 Amazon S3 桶加载的自定义 Lambda 轮询函数，否则将此设置保留为空。
- **LambdaS3Key** – 与您的 Lambda 代码构件对应的 Amazon S3 键。除非您使用自定义 Lambda 轮询函数，否则将此设置保留为空。
- **LambdaLoggingLevel** – 通常，将此设置保留为默认值，即 `INFO`。
- **ManagedPolicies** – 列出用于执行 Lambda 函数的托管式策略。通常，除非您使用自定义 Lambda 轮询函数，否则将此设置保留为空。
- **StreamRecordsHandler** – 通常，除非您为 Neptune 流中的记录使用自定义处理程序，否则将此设置保留为空。
- **StreamRecordsBatchSize** – 要从流中提取的最大记录数。您可以使用此参数来优化性能。默认值 (`5000`) 是一个很好的开始。允许的最大值为 10,000。数字越大，从流读取记录所需的网络调用就越少，但处理记录所需的内存越多。此参数的值越低，吞吐量就越低。
- **MaxPollingWaitTime** – 两次轮询之间的最长等待时间（以秒为单位）。确定调用 Lambda 轮询器轮询 Neptune 流的频率。将此值设置为 0 以进行连续轮询。最大值为 3600 秒（1 小时）。默认值（60 秒）是一个很好的开始，具体取决于图形数据更改的速度。
- **MaxPollingInterval** – 最长连续轮询周期（以秒为单位）。使用此参数设置 Lambda 轮询函数的超时。该值应介于 5 秒到 900 秒之间。默认值（600 秒）是一个很好的开始。
- **StepFunctionFallbackPeriod**— 等待轮询器的 step-function-fallback-period 单位数，之后通过 Amazon CloudWatch Events 调用步骤函数以从故障中恢复。默认值（5 分钟）是一个很好的开始。
- **StepFunctionFallbackPeriodUnit** – 用于衡量前面的 `StepFunctionFallbackPeriodUnit` 的时间单位（`minutes`、`hours` 或 `days`）。默认值 (`minutes`) 通常就足够了。

Neptune 流

- **NeptuneStreamEndpoint** –（必需）Neptune 源流的端点。它采用两种形式之一：

  - **https://*your DB cluster*:*port*/propertygraph/stream**（或其别名 https://*your DB cluster*:*port*/pg/stream）。
  - **https://*your DB cluster*:*port*/sparql/stream**.
- **Neptune Query Engine** – 选择 Gremlin、openCypher 或 SPARQL。
- **IAMAuthEnabledOnSourceStream** – 如果您的 Neptune 数据库集群使用 IAM 身份验证，请将此参数设置为 true。
- **StreamDBClusterResourceId** – 如果您的 Neptune 数据库集群使用 IAM 身份验证，请将此参数设置为集群资源 ID。资源 ID 与集群 ID 不同。相反，它采取的形式是：cluster- 后跟 28 个字母数字字符。可以在 Neptune 控制台的集群详细信息下找到它。

目标 Neptune 数据库集群

- **TargetNeptuneClusterEndpoint** – 目标备份集群的集群端点（仅限主机名）。

  请注意，如果您指定 TargetNeptuneClusterEndpoint，则不能同时指定 TargetSPARQLUpdateEndpoint。
- **TargetNeptuneClusterPort** – 目标集群的端口号。

  请注意，如果您指定 TargetSPARQLUpdateEndpoint，则会忽略 TargetNeptuneClusterPort 的设置。
- **IAMAuthEnabledOnTargetCluster** – 如果要在目标集群上启用 IAM 身份验证，则设置为 true。
- **TargetAWSRegion**— 目标备份集群的 AWS 区域，例如us-east-1）。只有当目标备份集群的 AWS 区域与 Neptune 源集群的区域不同时（例如跨区域复制），您才必须提供此参数。如果源区域和目标区域相同，则此参数是可选的。

  请注意，如果该TargetAWSRegion值不是 Nep tune 支持的有效 AWS 区域，则该过程将失败。
- **TargetNeptuneDBClusterResourceId** – 可选：只有在目标数据库集群上启用 IAM 身份验证时才需要此参数。设置为目标集群的资源 ID。
- **SPARQLTripleOnlyMode** - 确定是否启用仅限三重模式的布尔标志。在仅限三重模式下，不存在命名图形复制。默认值为 false。
- **TargetSPARQLUpdateEndpoint** – SPARQL 更新的目标端点的 URL，例如 https://abc.com/xyz。此端点可以是任何支持四元组或三元组的 SPARQL 存储。

  请注意，如果您指定 TargetSPARQLUpdateEndpoint，则无法同时指定 TargetNeptuneClusterEndpoint，TargetNeptuneClusterPort 的设置将被忽略。

- **BlockSparqlReplicationOnBlankNode** — 布尔标志，如果设置为`true`，则停止复制 SPARQL (RDF) 数据。 BlankNode 默认值为 `false`。

警报

- **Required to create Cloud watch Alarm**— `true` 如果要为新堆栈创建 CloudWatch 警报，请将其设置为。
- **SNS Topic ARN for Cloudwatch Alarm Notifications**— 应在 CloudWatch 其中发送警报通知的 SNS 主题 ARN（仅在启用警报时才需要）。
- **Email for Alarm Notifications** – 应向其发送警报通知的电子邮件地址（仅在启用警报时才需要）。

对于警报通知的目标，您可以添加仅限 SNS、仅限电子邮件或同时使用 SNS 和电子邮件。

运行 AWS CloudFormation 模板

现在，您可以完成预调配 Neptune 流使用者应用程序实例的过程，如下所示：

1. 在 AWS CloudFormation指定堆栈详细信息页面上，选择下一步。
2. 在选项页面上，选择下一步。
3. 在审核页面上，选中第一个复选框以确认 AWS CloudFormation 将创建 IAM 资源。选中第二个复选框以确认新堆栈的 `CAPABILITY_AUTO_EXPAND`。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` 明确确认在创建堆栈时将扩展宏，而无需事先审核。用户通常通过处理的模板创建更改集，以便在实际创建堆栈之前对宏所做的更改进行审核。有关更多信息，请参阅 AWS CloudFormation CreateStackAPI 参考中的AWS CloudFormation API。

   然后选择创建。

使用最新的 Lambda 构件更新流轮询器

您可以使用最新的 Lambda 代码构件更新流轮询器，如下所示：

1. 在中 AWS Management Console，导航到主父 AWS CloudFormation 堆栈 AWS CloudFormation 并选择该堆栈。
2. 为堆栈选择更新选项。
3. 选择替换当前模板。
4. 对于模板源，选择 Amazon S3 URL 并输入以下 S3 URL：

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_neptune.json
```

5. 在不更改任何 AWS CloudFormation 参数的情况下选择 “下一步”。
6. 选择 Update Stack。

现在，堆栈将使用最新的构件更新 Lambda 构件。

使用 Neptune 流跨区域复制进行灾难恢复

Neptune 提供了两种实现跨区域失效转移功能的方法：

- 跨区域快照复制和还原
- 使用 Neptune 流在两个不同区域中的两个集群之间复制数据。

跨区域快照复制和还原在不同区域中恢复 Neptune 集群的操作开销最低。但是，在区域之间复制快照可能需要大量的数据传输时间，因为快照是 Neptune 集群的完整备份。因此，跨区域快照复制和还原可用于只需要几个小时的恢复点目标 (RPO) 和几个小时的恢复时间目标 (RTO) 的场景。

恢复点目标 (RPO) 由两次备份之间的时间来衡量。它定义了从上次备份的时间到恢复数据库的时间之间可能丢失的数据量。

恢复时间目标 (RTO) 是通过执行恢复操作所花费的时间来衡量的。这是数据库集群在故障发生后失效转移到已恢复的数据库所花费的时间。

Neptune 流提供了一种使备份 Neptune 集群始终与主生产集群保持同步的方法。如果发生故障，您的数据库将失效转移到备份集群。这会将 RPO 和 RTO 缩短到几分钟，因为数据不断复制到备份集群，备份集群在任何时候都可以立即用作失效转移目标。

以这种方式使用 Neptune 流的缺点是，维护复制组件所需的操作开销以及让第二个 Neptune 数据库集群一直处于在线状态的成本都可能很高。

设置 Neptune 到 Neptune 的复制

主生产数据库集群位于给定源区域的 VPC 中。为了进行灾难恢复，您需要在不同的恢复区域中复制或模拟三项主要内容：

- 存储在集群中的数据。
- 主集群的配置。这将包括它是否使用 IAM 身份验证、是否已加密、其数据库集群参数、其实例参数、实例大小等。
- 它使用的网络拓扑，包括目标 VPC、其安全组等。

您可以使用如下所示的 Neptune 管理 API 来收集这些信息：

- DescribeDBClusters
- DescribeDBInstances
- DescribeDBClusterParameters
- DescribeDBParameters
- DescribeVpcs

根据您收集的信息，您可以使用以下过程在不同的区域中设置备份集群，如果出现故障，您的生产集群可以失效转移到该备份集群。

1：启用 Neptune 流

您可以使用 ModifyDBClusterParameterGroup 将 `neptune_streams` 参数设置为 1。然后，重启数据库集群中的所有实例，以使更改生效。

启用 Neptune 流后，最好在源数据库集群上至少执行一次添加或更新操作。这会在更改流中填充数据点，以后在将生产集群与备份集群重新同步时可以参考这些数据点。

2：在要设置备份集群的区域中创建新的 VPC

在与主集群不同的区域中创建新的 Neptune 数据库集群之前，您需要在目标区域建立一个新的 VPC 来托管该集群。主集群和备份集群之间的连接是通过 VPC 对等连接建立的，而 VPC 对等连接使用不同 VPC 中的私有子网间的流量。但是，要在两个 VPC 之间建立 VPC 对等连接，它们的 CIDR 块或 IP 地址空间不得重叠。这意味着，您不能在这两个区域中都只使用默认 VPC，因为默认 VPC 的 CIDR 块始终相同 (`172.31.0.0/16`)。

您可以使用目标区域中的现有 VPC，只要它满足以下条件即可：

- 它的 CIDR 块不与您的主集群所在 VPC 的 CIDR 块重叠。
- 它与另一个 VPC（与您的主集群所在的 VPC 具有相同 CIDR 块）尚未建立对等连接。

如果目标区域中没有合适的 VPC 可用，请使用 Amazon EC2 CreateVpc API 创建一个。

3：创建主集群的快照并将其还原到目标备份区域

现在，您可以在目标备份区域的相应 VPC 中创建一个新的 Neptune 集群，该集群是您的生产集群的副本：

在备份区域中制作生产集群的副本

1. 在目标备份区域中，重新创建生产数据库集群使用的参数和参数组。为此，您可以使用 CreateDBClusterParameterGroup、CreateDBParameterGroup、ModifyDBClusterParamete 和 ModifyDBParameterGroup。

   请注意，CopyDBClusterParameterGroup 和 CopyDBParameterGroup API 目前不支持跨区域复制。
2. 使用 CreateDBClusterSnapshot 在生产区域的 VPC 中创建生产集群的快照。
3. 使用 CopyDBClusterSnapshot 将快照复制到目标备份区域中的 VPC。
4. 使用 RestoreDBClusterFromSnapshot 通过复制的快照在目标备份区域的 VPC 中创建新的数据库集群。使用您从主生产集群中复制的配置设置和参数。
5. 新的 Neptune 集群现已存在，但不包含任何实例。使用 CreateDBInstance 创建新的主/写入器实例，其实例类型和大小与生产集群的写入器实例相同。此时无需创建其它只读副本，除非您的备份实例将在失效转移之前用于为目标区域的读取 I/O 提供服务。

4：在主集群的 VPC 和新的备份集群的 VPC 之间建立 VPC 对等连接

通过设置 VPC 对等连接，您可以使主集群的 VPC 与备份集群的 VPC 通信，就像它们是单个私有网络一样。为此，请执行以下步骤：

1. 从生产集群的 VPC 中，调用 CreateVpcPeeringConnection API 以建立对等连接。
2. 从目标备份集群的 VPC 中，调用 AcceptVpcPeeringConnection API 以接受对等连接。
3. 从生产集群的 VPC 中，使用 CreateRoute API 向 VPC 的路由表添加一条路由，该路由将所有流量重定向到目标 VPC 的 CIDR 块，以便它使用 VPC 对等连接前缀列表。
4. 同样，从目标备份集群的 VPC，使用 CreateRoute API 向 VPC 的路由表添加一条路由，该路由将流量路由到主集群的 VPC。

5：设置 Neptune 流复制基础设施

现在，两个集群都已部署完毕，两个区域之间的网络通信也已建立，请使用 Nep tune-to-Neptune 模板 AWS CloudFormation 来部署 N eptune 流使用者 Lambda 函数以及支持数据复制的额外基础架构。在主生产集群的 VPC 中执行此操作。

您需要为此 AWS CloudFormation 堆栈提供的参数是：

- **`NeptuneStreamEndpoint`** – 主集群的流端点，采用 URL 格式。例如：`https://(cluster name):8182/pg/stream`。
- **`QueryEngine`** – 这必须是 `gremlin`、`sparql` 或 `openCypher`。
- **`RouteTableIds`** – 允许您为 DynamoDB VPC 端点和监控 VPC 端点添加路由。

  如果主集群的 VPC 上尚不存在另外两个参数，即 `CreateMonitoringEndpoint` 和 `CreateDynamoDBEndpoint` ，则也必须将它们设置为 true。如果它们已经存在，请确保将其设置为 false，否则 AWS CloudFormation 创建将失败。
- **`SecurityGroupIds`** – 指定 Lambda 使用者用于与主集群的 Neptune 流端点通信的安全组。

  在目标备份集群中，附加一个允许来自该安全组的流量的安全组。
- **`SubnetIds`** – 主集群 VPC 中的子网 ID 列表，Lambda 使用者可以使用该子网与主集群进行通信。
- **`TargetNeptuneClusterEndpoint`** – 目标备份集群的集群端点（仅限主机名）。
- **`TargetAWSRegion`**— 目标备份集群的 AWS 区域，例如`us-east-1`）。只有当目标备份集群的 AWS 区域与 Neptune 源集群的区域不同时（例如跨区域复制），您才必须提供此参数。如果源区域和目标区域相同，则此参数是可选的。

  请注意，如果该`TargetAWSRegion`值不是 Nep tune 支持的有效 AWS 区域，则该过程将失败。
- **`VPC`** – 主集群的 VPC 的 ID。

所有其它参数均可保留默认值。

AWS CloudFormation 模板部署完成后，Neptune 将开始将所有更改从主集群复制到备份集群。您可以在 Lambda 使用者函数生成的 CloudWatch 日志中监控此复制。

其他考虑因素

- 如果您需要在主集群和备份集群之间使用 IAM 身份验证，也可以在调用模板时进行 AWS CloudFormation 设置。

- 如果对主集群启用了静态加密，则在将快照复制到目标区域并在目标区域中关联新的 KMS 密钥时，请考虑如何管理关联的 KMS 密钥。
- 最佳实践是在应用程序中使用的 Neptune 端点前方使用 DNS CNAME。然后，如果您需要手动失效转移到目标备份集群，则可以将这些 CNAME 更改为指向目标集群和/或实例端点。

使用亚马逊服务在 Amazon Neptune 中进行全文搜索 OpenSearch

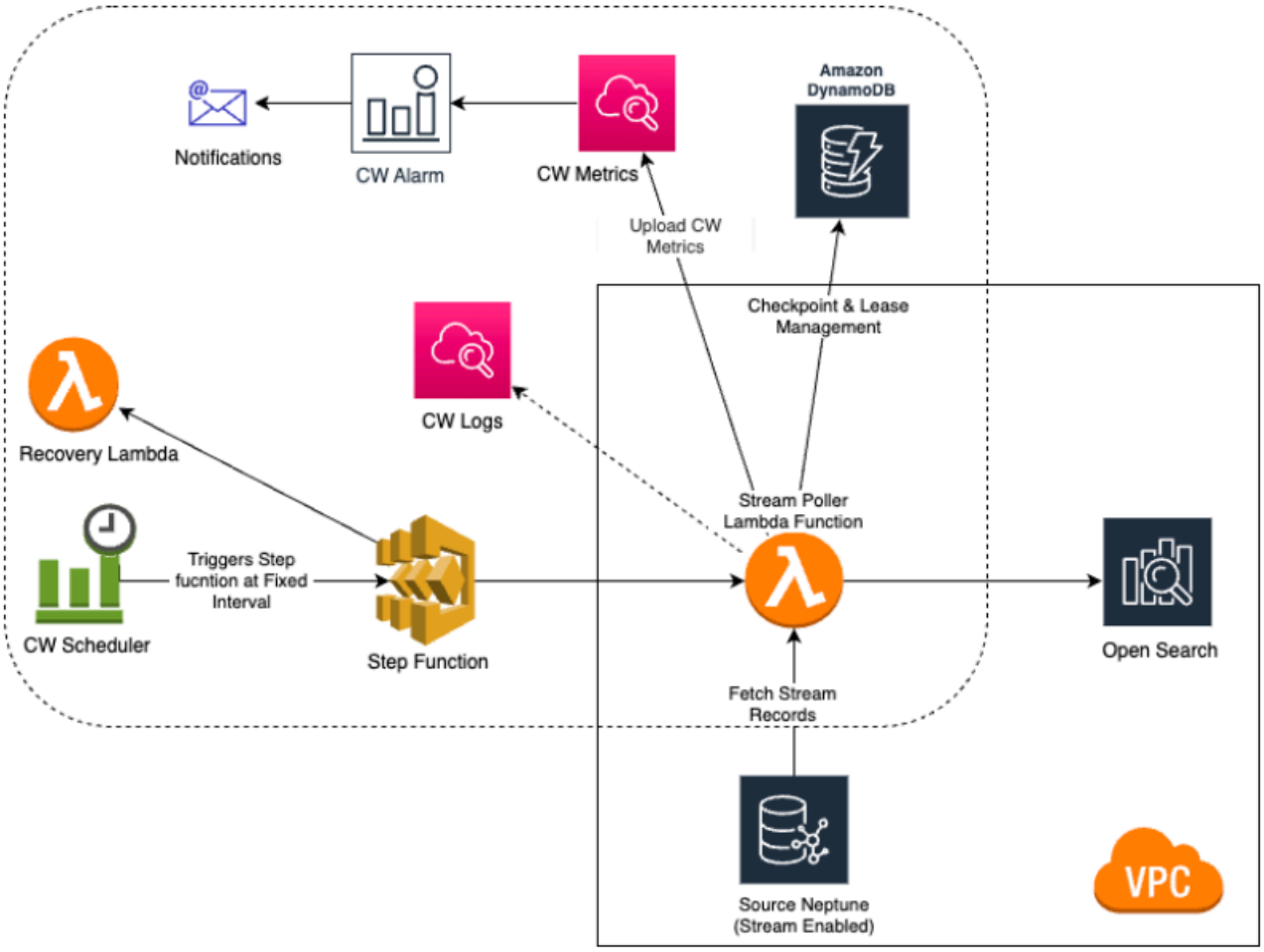
Neptune 与亚马逊 OpenSearch 服务（OpenSearch 服务）集成，支持在 Gremlin 和 SPARQL 查询中进行全文搜索。此特征从 Neptune 引擎版本 1.0.2.1 开始提供，但我们建议将其与引擎版本 1.0.4.2 或更高版本结合使用，以利用最新的修复程序。

从引擎版本 1.3.0.0 开始，亚马逊 Neptune 支持使用 OpenSearch 亚马逊无服务器服务在 Gremlin 和 SPARQL 查询中进行全文搜索。

> **Note**
>
> 与亚马逊 OpenSearch 服务集成时，Neptune 需要 Elasticsearch 版本 7.1 或更高版本，并且可以使用 OpenSearch 2.3、2.5 及更高版本。Neptune 也适用于无服务器。OpenSearch

您可以将 Neptune 与已根据填充的现有 OpenSearch 服务集群一起使用。OpenSearch 数据的 Neptune 数据模型或者，您可以使用堆栈创建与 Neptune 关联的 OpenSearch AWS CloudFormation 服务域。

> **Important**
>
> 此处描述的 Neptune 到 OpenSearch 复制过程不会复制空白节点。这是一个需要注意的重要限制。
>
> 此外，如果您在 OpenSearch 集群上启用精细访问控制，则还需要在 Neptune 数据库中启用 IAM 身份验证。



主题

- 亚马逊 Neptune-to-复制 OpenSearch
- 复制到 OpenSearch 无服务器
- 从启用了精细访问控制 (FGAC) 的 OpenSearch 集群进行查询
- 在 Neptune 全文搜索查询中使用 Apache Lucene 查询语法
- OpenSearch 数据的 Neptune 数据模型
- Neptune 全文搜索参数
- Amazon Neptune 中的非字符串 OpenSearch 索引
- Amazon Neptune 中的全文搜索查询执行
- Neptune 中使用全文搜索的示例 SPARQL 查询
- 在 Gremlin 查询中使用 Neptune 全文搜索
- Neptune 全文搜索故障排除

亚马逊 Neptune-to-复制 OpenSearch

Amazon Neptune 支持使用亚马逊服务（服务）在 Gremlin 和 SPARQL 查询中进行全文搜索。OpenSearch OpenSearch 您可以使用 AWS CloudFormation 堆栈将 OpenSearch 服务域链接到 Neptune。该 AWS CloudFormation 模板创建了一个流使用者应用程序实例，该实例提供 Neptune 到-的复制。OpenSearch

在开始之前，您需要一个启用了流的现有 Neptune 数据库集群作为源，并需要一个 OpenSearch 服务域作为复制目标。

如果您在 Neptune 数据库集群所在的 VPC 中已经有一个可以通过 Lambda 访问的现有目标 OpenSearch 服务域，则模板可以使用该服务域。否则，您需要创建一个新目标域。

> **Note**
>
> 您创建的 OpenSearch 集群和 Lambda 函数必须与您的 Neptune 数据库集群位于同一 VPC 中，并且必须在 VPC 模式（不是互联网模式）下配置 OpenSearch 集群。

我们建议您使用新创建的 Neptune 实例与服务配合 OpenSearch 使用。如果您使用已有数据的现有实例，则应在进行查询之前执行 OpenSearch 服务数据同步，否则可能会出现数据不一致的情况。该 GitHub 项目提供了如何执行同步的示例：将 Nep tune 导出到 OpenSearch (https://github.com/awslabs/ amazon-neptune-tools /tree/ export-neptune-to-elasticsearch master/)。

> **Important**
>
> 与亚马逊 OpenSearch 服务集成时，Neptune 需要 Elasticsearch 版本 7.1 或更高版本，并且可以使用 2.3、 OpenSearch 2.5 和未来兼容的 Opensearch 版本。

> **Note**
>
> 从引擎版本 1.3.0.0 开始，亚马逊 Neptune 支持使用 OpenSearch 亚马逊无服务器服务在 Gremlin 和 SPARQL 查询中进行全文搜索。

主题

- 使用 AWS CloudFormation 模板启动 Neptune 到-的复制 OpenSearch

- 在现有 Neptune 数据库上启用全文搜索
- 更新流轮询器
- 禁用和重新启用流轮询器进程

使用 AWS CloudFormation 模板启动 Neptune 到-的复制 OpenSearch

启动特定于您所在地区的 AWS CloudFormation 堆栈

以下每个 AWS CloudFormation 模板都会在特定 AWS 区域创建一个 Streams-consumer 应用程序实例。要使用 AWS CloudFormation 控制台启动相应的堆栈，请根据要使用的 AWS 区域，选择下表中的启动堆栈按钮之一。

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	
美国东部（俄亥俄州）	视图	在 Designer 中查看	
美国西部（北加利福尼亚）	视图	在 Designer 中查看	
US West（Oregon）	视图	在 Designer 中查看	
加拿大（中部）	视图	在 Designer 中查看	
南美洲（圣保罗）	视图	在 Designer 中查看	
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	
欧洲地区（爱尔兰）	视图	在 Designer 中查看	
欧洲地区（伦敦）	视图	在 Designer 中查看	

区域	查看	在 Designer 中查看	发布
欧洲地区（巴黎）	视图	在 Designer 中查看	Launch Stack
欧洲地区（法兰克福）	视图	在 Designer 中查看	Launch Stack
中东（巴林）	视图	在 Designer 中查看	Launch Stack
中东（阿联酋）	视图	在 Designer 中查看	Launch Stack
以色列（特拉维夫）	视图	在 Designer 中查看	Launch Stack
非洲（开普敦）	视图	在 Designer 中查看	Launch Stack
亚太地区（香港）	视图	在 Designer 中查看	Launch Stack
亚太地区（东京)	视图	在 Designer 中查看	Launch Stack
亚太地区（首尔）	视图	在 Designer 中查看	Launch Stack
亚太地区（新加坡）	视图	在 Designer 中查看	Launch Stack
亚太地区（孟买）	视图	在 Designer 中查看	Launch Stack
中国（北京）	视图	在 Designer 中查看	Launch Stack
中国（宁夏）	视图	在 Designer 中查看	Launch Stack
AWS GovCloud（美国西部）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
AWS GovCloud （美国东部）	视图	在 Designer 中查看	

在 Create stack (创建堆栈) 主页上，选择 Next (下一步)。

添加有关您正在创建的新 OpenSearch堆栈的详细信息

指定堆栈详细信息页面提供了属性和参数，可用于控制全文搜索的设置：

堆栈名称-您正在创建的新 AWS CloudFormation 堆栈的名称。通常可以使用默认值 `NeptuneStreamPoller`。

在 Parameters (参数) 下，提供以下内容：

流使用者运行的 VPC 的网络配置

- **`VPC`** – 提供将运行轮询 Lambda 函数的 VPC 的名称。
- **`List of Subnet IDs`** – 建立网络接口的子网。添加与您的 Neptune 集群对应的子网。
- **`List of Security Group Ids`** – 提供向源 Neptune 数据库集群授予写入入站访问权限的安全组的 ID。
- **`List of Route Table Ids`** – 如果您还没有 Amazon DynamoDB 端点，则需要在 Neptune VPC 中创建一个。您必须提供与子网关联的路由表 ID 的逗号分隔列表。
- **`Require to create Dynamo DB VPC Endpoint`** – 默认为 `true` 的布尔值。如果您已在 VPC 中创建 DynamoDB 端点，则只需将其更改为 `false`。
- **`Require to create Monitoring VPC Endpoint`** – 默认为 `true` 的布尔值。如果您已在 VPC 中创建镜像终端节点，则只需将其更改为 `false`。

流轮询器

- **`Application Name`** – 您通常可以将此设置保留为默认值 (`NeptuneStream`)。如果您使用不同的名称，则该名称必须是唯一的。
- **`Memory size for Lambda Poller`** – 用于设置 lambda 轮询器函数可用的内存大小。默认值为 2,048 MB。
- **`Lambda Runtime`** – 从 Neptune 流中检索项目的 Lambda 函数使用的语言。您可以将其设置为 `python3.9` 或 `java8`。

- **S3 Bucket having Lambda code artifacts** – 除非您使用从其它 S3 桶加载的自定义 Lambda 轮询函数，否则将此设置保留为空。
- **S3 Key corresponding to Lambda Code artifacts** – 除非您使用自定义 Lambda 轮询函数，否则将此设置保留为空。
- **StartingCheckpoint** – 流轮询器的起始检查点。默认值为 `0:0`，表示从 Neptune 流的开头开始。
- **StreamPollerInitialState** – 轮询器的初始状态。默认值为 `ENABLED`，这意味着流复制将在整个堆栈创建完成后立即开始。
- **Logging level for Lambda** – 通常，将此设置保留为默认值 `INFO`。
- **Managed Policies for Lambda Execution** – 通常，除非您使用自定义 Lambda 轮询函数，否则将此设置保留为空。
- **Stream Records Handler** – 通常，除非您为 Neptune 流中的记录使用自定义处理程序，否则将此设置保留为空。
- **Maximum records Fetched from Stream** – 您可以使用此参数来优化性能。默认值 (`100`) 是一个很好的开始。允许的最大值为 10,000。数字越大，从流读取记录所需的网络调用就越少，但处理记录所需的内存越多。
- **Max wait time between two Polls (in Seconds)** – 确定调用 Lambda 轮询器来轮询 Neptune 流的频率。将此值设置为 0 以进行连续轮询。最大值为 3600 秒（1 小时）。默认值（60 秒）是一个很好的开始，具体取决于图形数据更改的速度。
- **Maximum Continuous polling period (in Seconds)** – 用于设置 Lambda 轮询函数的超时。它应该是 5 秒到 900 秒之间。默认值（600 秒）是一个很好的开始。
- **Step Function Fallback Period**— 等待轮询器的 step-function-fallback-period 单位数，之后通过 Amazon CloudWatch Events 调用步骤函数以从故障中恢复。默认值（5 分钟）是一个很好的开始。
- **Step Function Fallback Period Unit** –用于测量上述 `Step Function Fallback Period` 的时间单位（分钟、小时、天）。默认值（分钟）通常就足够了。
- **Data replication scope**— 确定是同时复制节点和边缘，还是只复制节点 OpenSearch （这仅适用于 Gremlin 引擎数据）。默认值（All (全部)）通常是一个很好的开始。
- **Ignore OpenSearch missing document error**— 用于确定是否 OpenSearch 可以忽略中的缺失文档错误的标志。缺少文档错误很少发生，但如果不忽略，则需要手动干预。默认值（`True`）通常是一个很好的开始。
- **Enable Non-String Indexing** – 用于启用或禁用对没有字符串内容的字段编制索引的标志。如果将此标志设置为`true`，则对非字符串字段进行索引 OpenSearch，或者如果`false`仅对字符串字段进行索引。默认值为 `true`。

- **Properties to exclude from being inserted into OpenSearch**— 要从索引中排除的属性或谓词键的逗号分隔列表。 OpenSearch 如果将此 CFN 参数值留空，则会对所有属性键编制索引。
- **Datatypes to exclude from being inserted into OpenSearch**— 要从索引中排除的属性或谓词数据类型的逗号分隔列表。 OpenSearch 如果将此 CFN 参数值留空，则会对所有可以安全转换为 OpenSearch 数据类型的属性值进行索引。

Neptune 流

- **Endpoint of source Neptune Stream** – (必需) 采用以下两种形式之一：
  - **https://*your DB cluster*:*port*/propertygraph/stream** (或其别名 https://*your DB cluster*:*port*/pg/stream) 。
  - https://*your DB cluster*:*port*/sparql/stream
- **Neptune Query Engine** – 选择 Gremlin 或 SPARQL。
- **Is IAM Auth Enabled?** – 如果您的 Neptune 数据库集群使用 IAM 身份验证，请将此参数设置为 true。
- **Neptune Cluster Resource Id** – 如果您的 Neptune 数据库集群使用 IAM 身份验证，请将此参数设置为集群资源 ID。资源 ID 与集群 ID 不同。相反，它采取的形式是：cluster- 后跟 28 个字母数字字符。可以在 Neptune 控制台的集群详细信息下找到它。

目标 OpenSearch 集群

- **Endpoint for OpenSearch service**— (必需) 在您的 VPC 中提供 OpenSearch 服务的终端节点。
- **Number of Shards for OpenSearch Index** – 默认值 (5) 通常是一个很好的开始。
- **Number of Replicas for OpenSearch Index** – 默认值 (1) 通常是一个很好的开始。
- **Geo Location Fields for Mapping** – 如果您使用的是地理位置字段，请在此处列出属性键。

警报

- **Require to create Cloud watch Alarm**— true 如果要为新堆栈创建 CloudWatch 警报，请将其设置为。
- **SNS Topic ARN for Cloudwatch Alarm Notifications**— 应在 CloudWatch 其中发送警报通知的 SNS 主题 ARN (仅在启用警报时才需要) 。

- **Email for Alarm Notifications** – 应向其发送警报通知的电子邮件地址（仅在启用警报时才需要）。

对于警报通知的目标，您可以添加仅限 SNS、仅限电子邮件或同时使用 SNS 和电子邮件。

运行 AWS CloudFormation 模板

现在，您可以完成预调配 Neptune 流使用者应用程序实例的过程，如下所示：

1. 在 AWS CloudFormation指定堆栈详细信息页面上，选择下一步。
2. 在选项页面上，选择下一步。
3. 在审核页面上，选中第一个复选框以确认 AWS CloudFormation 将创建 IAM 资源。选中第二个复选框以确认新堆栈的 `CAPABILITY_AUTO_EXPAND`。

   > **Note**
   >
   > `CAPABILITY_AUTO_EXPAND` 明确确认在创建堆栈时将扩展宏，而无需事先审核。用户通常通过处理的模板创建更改集，以便在实际创建堆栈之前对宏所做的更改进行审核。有关更多信息，请参阅《 AWS CloudFormation CreateStackAPI 参考》中的AWS CloudFormation API 操作。

   然后选择创建。

在现有 Neptune 数据库上启用全文搜索

如果您可以暂停写入工作负载

在现有 Neptune 数据库上启用全文搜索的最佳方法通常如下所示，前提是您可以暂停写入工作负载。它需要创建克隆，使用集群参数启用流，然后重新启动所有实例。创建克隆的操作相对较快，因此所需的停机时间是有限的。

下面列出了所需的步骤：

1. 停止数据库上的所有写入工作负载。
2. 在数据库上启用流（请参阅启用 Neptune 流）。
3. 创建数据库的克隆（请参阅 Neptune 中的数据库克隆）。

4. 恢复写入工作负载。
5. 使用 github 上的export-neptune-to-elasticsearch工具执行从克隆的数据库到 OpenSearch域的一次性同步。
6. 使用您所在区域的AWS CloudFormation 模板从原始数据库开始同步，并进行持续更新（无需更改模板中的配置）。
7. 删除克隆的数据库和为该`export-neptune-to-elasticsearch`工具创建的 AWS CloudFormation 堆栈。

如果您无法暂停写入工作负载

如果您承担不起在数据库上暂停写入工作负载的代价，那么以下方法所需的停机时间甚至比上述推荐的方法还要少，但需要谨慎行事：

1. 在数据库上启用流（请参阅启用 Neptune 流）。
2. 创建数据库的克隆（请参阅 Neptune 中的数据库克隆）。
3. 通过对 Streams API 端点执行此类命令，获取克隆数据库上流的最新 `eventID`（有关更多信息，请参阅调用 Neptune Streams REST API）：

   ```
   curl "https://(your neptune endpoint):(port)/(propertygraph or sparql)/stream?
   iteratorType=LATEST"
   ```

   记下响应的 `lastEventId` 对象中 `commitNum` 和 `opNum` 字段的值。
4. 使用 github 上的export-neptune-to-elasticsearch工具执行从克隆的数据库到 OpenSearch域的一次性同步。
5. 使用您所在区域的AWS CloudFormation 模板从原始数据库开始同步，并进行持续更新。

   创建堆栈时进行以下更改：在堆栈详细信息页面的参数部分中，使用上面记录的 `commitNum` 和 `opNum` 值将 `StartingCheckpoint` 字段的值设置为 *`commitNum`*:*`opnum`*。
6. 删除克隆的数据库和为该`export-neptune-to-elasticsearch`工具创建的 AWS CloudFormation 堆栈。

更新流轮询器

使用最新的 Lambda 构件更新流轮询器

您可以使用最新的 Lambda 代码构件更新流轮询器，如下所示：

1. 在中 AWS Management Console，导航到主父 AWS CloudFormation 堆栈 AWS CloudFormation 并选择该堆栈。
2. 为堆栈选择更新选项。
3. 选择替换当前模板。
4. 对于模板源，选择 Amazon S3 URL 并输入以下 S3 URL：

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_elastic_search.json
```

5. 在不更改任何 AWS CloudFormation 参数的情况下选择 “下一步”。
6. 选择 Update Stack。

现在，堆栈将使用最新的构件更新 Lambda 构件。

扩展流轮询器以支持自定义字段

可以轻松扩展当前的流轮询器以编写用于处理自定义字段的自定义代码，正如这篇博客文章中详细介绍的那样：使用 Neptune Streams 捕获图形更改。

> **Note**
>
> 在中添加自定义字段时 OpenSearch，请确保将新字段添加为谓词的内部对象（请参阅Neptune 全文搜索数据模型）。

禁用和重新启用流轮询器进程

> **Warning**
>
> 禁用流轮询器进程时务必小心！如果进程的暂停时间超过流到期时段，则可能会发生数据丢失。默认时段为 7 天，但从引擎版本 1.2.0.0 开始，您可以将自定义流过期时段设置为最长 90 天。

禁用（暂停）流轮询器进程

1. 登录 AWS Management Console 并打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。

2. 在导航窗格中选择规则。
3. 选择一个规则，其名称包含您在用于设置直播轮询器的 AWS CloudFormation 模板中作为应用程序名称提供的名称。
4. 选择 禁用。
5. 打开 Step Functions 控制台，网址为 https://console.aws.amazon.com/states/。
6. 选择与流轮询器进程对应的正在运行的步进函数。同样，该步骤函数的名称包含您在用于设置流轮询器的 AWS CloudFormation 模板中作为应用程序名称提供的名称。您可以按函数执行状态进行筛选，以仅查看正在运行的函数。
7. 选择停止。

重新启用流轮询器进程

1. 登录 AWS Management Console 并打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。
2. 在导航窗格中选择规则。
3. 选择一个规则，其名称包含您在用于设置直播轮询器的 AWS CloudFormation 模板中作为应用程序名称提供的名称。
4. 选择 禁用。基于指定计划间隔的事件规则现在将触发步进函数的新执行。

复制到 OpenSearch 无服务器

从引擎版本 1.3.0.0 开始，Amazon Neptune 支持使用 Amazon OpenSearch Service 无服务器在 Gremlin 和 SPARQL 查询中进行全文搜索。

如果您要复制到 OpenSearch 无服务器，请将 Lambda 流轮询器执行角色添加到 OpenSearch 无服务器集合的数据访问策略中。Lambda 流轮询器执行角色的 ARN 格式如下：

```
arn:aws:iam::(account ID):role/stack-name-NeptuneOSReplication-
NeptuneStreamPollerExecu-(uuid)
```

有关更多信息，请参阅 Data access control for Amazon OpenSearch Serverless（Amazon OpenSearch 无服务器的数据访问控制）。

如果您已在 OpenSearch 集群上启用了精细访问控制，则还需要在 Neptune 数据库中启用 IAM 身份验证。

用于连接 Neptune 数据库的 IAM 实体（用户或角色）应同时具有 Neptune 和 OpenSearch 无服务器集合的权限。这意味着您的用户或角色必须附有如下所示的 OpenSearch 无服务器策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::(account ID):root"
      },
      "Action": "aoss:APIAccessAll",
      "Resource": "arn:aws:aoss:(region):(account ID):collection/(collection ID)"
    }
  ]
}
```

参阅 Amazon Neptune 的自定义 IAM 数据访问策略语句 了解更多信息。

从启用了精细访问控制 (FGAC) 的 OpenSearch 集群进行查询

如果您已在 OpenSearch 集群上启用了精细访问控制，则还需要在 Neptune 数据库中启用 IAM 身份验证。

用于连接 Neptune 数据库的 IAM 实体（用户或角色）应同时具有 Neptune 和 OpenSearch 集群的权限。这意味着您的用户或角色必须附有如下所示的 OpenSearch Service 策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::account-id:root"
      },
      "Action": "es:*",
      "Resource": "arn:aws:es:region:account-id:es-resource-id/*"
    }
  ]
}
```

参阅 Amazon Neptune 的自定义 IAM 数据访问策略语句 了解更多信息。

在 Neptune 全文搜索查询中使用 Apache Lucene 查询语法

OpenSearch 支持使用 Apache Lucene 语法进行 query_string 查询。这对于在查询中传递多个筛选条件特别有用。

Neptune 使用嵌套结构将属性存储在 OpenSearch 文档中（请参阅Neptune 全文搜索数据模型）。使用 Lucene 语法时，需要使用此嵌套模型中属性的完整路径。

以下是一个 Gremlin 示例：

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V()
 .has("*", "Neptune#fts predicates.name.value:\"Jane Austin\" AND entity_type:Book")
```

以下是一个 SPARQL 示例：

```
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
  neptune-fts:config neptune-fts:endpoint 'http://localhost:9200 (http://
localhost:9200/)' .
  neptune-fts:config neptune-fts:queryType 'query_string' .
  neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*name.value:Ronak AND
 predicates.\\*foaf\\*surname.value:Sh*" .
  neptune-fts:config neptune-fts:field '*' .
  neptune-fts:config neptune-fts:return ?res .
}
```

OpenSearch 数据的 Neptune 数据模型

Amazon Neptune 使用统一的 JSON 文档结构在 OpenSearch Service 中存储 SPARQL 和 Gremlin 数据。OpenSearch 中的每个文档对应于一个实体，并存储该实体的所有相关信息。对于 Gremlin，顶点和边缘视为实体，因此对应的 OpenSearch 文档包含有关顶点、标签和属性的信息。对于 SPARQL，主语可以视为实体，因此对应的 OpenSearch 文档在一个文档中包含有关所有谓词-对象对的信息。

> **Note**
>
> Neptune 至 OpenSearch 复制实施仅存储字符串数据。但是，您可以对其进行修改以存储其他数据类型。

统一 JSON 文档结构如下所示。

```
{
  "entity_id": "Vertex Id/Edge Id/Subject URI",
  "entity_type": [List of Labels/rdf:type object value],
  "document_type": "vertex/edge/rdf-resource"
  "predicates": {
    "Property name or predicate URI": [
      {
        "value": "Property Value or Object Value",
        "graph": "(Only for Sparql) Named Graph Quad is present"
        "language": "(Only for Sparql) rdf:langString"
      },
      {
        "value": "Property Value 2/ Object Value 2",
      }
    ]
  }
}
```

- `entity_id` – 表示文档的实体唯一 ID。
  - 对于 SPARQL，这是主语 URI。
  - 对于 Gremlin，这是 `Vertex_ID` 或 `Edge_ID`。
- `entity_type` – 表示顶点或边缘的一个或多个标签，或者表示主题的零个或多个 `rdf:type` 谓词值。
- `document_type` – 用于指定当前文档表示顶点、边缘还是 rdf 资源。
- `predicates` – 对于 Gremlin，存储顶点或边缘的属性和值。对于 SPARQL，它存储谓语-宾语对。

  属性名称在 OpenSearch 中采用 `properties.name.value` 格式。要查询它，您必须以该形式命名它。
- `value` – Gremlin 的属性值或者 SPARQL 的对象值。
- `graph` – SPARQL 的命名图形。
- `language` – SPARQL 中 `rdf:langString` 文本的语言标签。

示例 SPARQL OpenSearch 文档

数据

```
@prefix dt:   <http://example.org/datatype#> .
@prefix ex:   <http://example.org/> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:simone   rdf:type     ex:Person                    ex:g1
ex:michael  rdf:type     ex:Person                    ex:g1
ex:simone   ex:likes     "spaghetti"                  ex:g1

ex:simone   ex:knows     ex:michael                   ex:g2   # Not stored in ES
ex:simone   ex:likes     "spaghetti"                  ex:g2
ex:simone   ex:status    "La vita è un sogno"@it      ex:g2

ex:simone   ex:age       "40"^^xsd:int                DG      # Not stored in ES
ex:simone   ex:dummy     "testData"^^dt:newDataType   DG
ex:simone   ex:hates     _:bnode                              # Not stored in ES
_:bnode     ex:means     "coding"                     DG      # Not stored in ES
```

文档

```
{
  "entity_id": "http://example.org/simone",
  "entity_type": ["http://example.org/Person"],
  "document_type": "rdf-resource"
  "predicates": {
    "http://example.org/likes": [
      {
        "value": "spaghetti",
        "graph": "http://example.org/g1"
      },
      {
        "value": "spaghetti",
        "graph": "http://example.org/g2"
      }
    ]
    "http://example.org/status": [
      {
        "value": "La vita è un sogno",
        "language": "it"       // Only present for rdf:langString
      }
    ]
  }
```

```
}
```

```
{
  "entity_id" : "http://example.org/michael",
  "entity_type" : ["http://example.org/Person"],
  "document_type": "rdf-resource"
}
```

示例 Gremlin OpenSearch 文档

数据

```
# Vertex 1
simone   label    Person       <== Label
simone   likes    "spaghetti"  <== Property
simone   likes    "rice"       <== Property
simone   age      40           <== Property

# Vertex 2
michael  label    Person       <== Label

# Edge 1
simone  knows     michael      <== Edge
e1      updated  "2019-07-03"  <== Edge Property
e1      through  "company"     <== Edge Property
e1      since     10          <== Edge Property
```

文档

```
{
  "entity_id": "simone",
  "entity_type": ["Person"],
  "document_type": "vertex",
  "predicates": {
    "likes": [
      {
        "value": "spaghetti"
      },
      {
        "value": "rice"
      }
```

```
    ]
  }
}
```

```
{
  "entity_id" : "michael",
  "entity_type" : ["Person"],
  "document_type": "vertex"
}
```

```
{
  "entity_id": "e1",
  "entity_type": ["knows"],
  "document_type": "edge"
  "predicates": {
    "through": [
      {
        "value": "company"
      }
    ]
  }
}
```

Neptune 全文搜索参数

Amazon Neptune 使用以下参数在 Gremlin 和 SPARQL 中指定全文 OpenSearch 查询：

- **`queryType`** –（必需）OpenSearch 查询的类型。（有关查询类型的列表，请参阅 OpenSearch 文档）。Neptune 支持以下 OpenSearch 查询类型：
  - simple_query_string – 基于提供的查询字符串返回文档，使用具有有限但容错的 Lucene 语法的分析器。这是默认查询类型。

    此查询使用简单的语法来解析提供的查询字符串并将其拆分为基于特殊运算符的词条。然后，查询会在返回匹配文档之前独立分析每个词条。

    虽然其语法比 `query_string` 查询更有限，但 `simple_query_string` 查询不会返回无效语法的错误。相反，它忽略查询字符串的任何无效部分。
  - match – `match` 查询是执行全文搜索的标准查询，包括模糊匹配的选项。
  - prefix – 返回在指定字段中包含特定前缀的文档。

- fuzzy – 如果文档包含与搜索词相似的词条（相似度用 Levenshtein 编辑距离衡量），则返回该文档。

  编辑距离是将一个词条转换为另一个词条所需的单字符更改数。这些更改可包括：

  - 更改一个字符（box 更改为 fox）。
  - 删除一个字符（black 更改为 lack）。
  - 插入一个字符（sic 更改为 sick）。
  - 转置两个相邻的字符（act 更改为 cat）。

  查找相似词条时，fuzzy 查询会在指定的编辑距离内为搜索词创建所有可能变体和扩展，然后返回每个变体的精确匹配。

- term – 如果文档的指定字段之一包含指定词条的精确匹配，则返回该文档。

  您可以使用 `term` 查询基于精确值（如价格、产品 ID 或用户名）查找文档。

  > **Warning**
  >
  > 避免对文本字段使用 term 查询。默认情况下，作为分析的一部分，OpenSearch 会更改文本字段的值，这会使得很难找到文本字段值的精确匹配。
  > 要搜索文本字段值，请改用 match 查询。

- query_string – 基于提供的查询字符串返回文档，使用具有严格语法（Lucene 语法）的分析器。

  此查询使用语法来解析和拆分基于运算符（如 AND 或 NOT）提供的查询字符串。然后，查询会在返回匹配文档之前独立分析每个拆分文本。

  您可以使用 `query_string` 查询创建复杂的搜索，其中包括通配符、跨多个字段的搜索等。虽然查询灵活多变，但它是严格的，如果查询字符串包含任何无效语法，则会返回错误。

  > **Warning**
  >
  > 由于它对任何无效语法返回错误，因此我们不建议对搜索框使用 `query_string` 查询。如果您不需要支持查询语法，请考虑使用 `match` 查询。如果您需要查询语法的功能，请使用不太严格的 `simple_query_string` 查询。

- **`field`** – OpenSearch 中用来运行搜索的字段。只有在 `queryType` 允许它（如 `simple_query_string` 和 `query_string` 那样）时才能省略此项，在这种情况下，搜索针对所有字段。在 Gremlin 中，这是隐式的。

如果查询允许，则可以指定多个字段，如 `simple_query_string` 和 `query_string` 查询。

- **`query`** – （必需）要针对 OpenSearch 运行的查询。此字段的内容可能因 queryType 而异。不同的 queryType 接受不同的语法，如 `Regexp`。在 Gremlin 中，`query` 是隐式的。
- **`maxResults`** – 要返回的最大结果数量。默认为 `index.max_result_window` OpenSearch 设置，该设置本身默认为 10000。`maxResults` 参数可以指定小于该值的任意数字。

  > **Important**
  >
  > 如果您将 `maxResults` 的值设置为高于 OpenSearch `index.max_result_window` 值，并且尝试检索的结果数超过 `index.max_result_window` 的值，则 OpenSearch 将失败，并显示 `Result window is too large` 错误。但是，Neptune 正常处理这个问题，而不会传播错误。如果您试图提取的结果数超过 `index.max_result_window` 的值，请记住这一点。

- **`minScore`** – 返回的搜索结果必须具有的最低分数。有关结果评分的说明，请参阅 OpenSearch 相关性文档。
- **`batchSize`** – Neptune 始终以批量方式提取数据（默认批量大小为 100）。您可以使用此参数来优化性能。批量大小不能超过 `index.max_result_window` OpenSearch 设置，该设置默认为 10000。
- **`sortBy`** – 一个可选参数，允许您按以下选项之一对 OpenSearch 返回的结果进行排序：
  - 文档中的特定字符串字段 –

    例如，在 SPARQL 查询中，您可以指定：

    ```
    neptune-fts:config neptune-fts:sortBy foaf:name .
    ```

    在类似的 Gremlin 查询中，您可以指定：

    ```
    .withSideEffect('Neptune#fts.sortBy', 'name')
    ```

  - 文档中的特定非字符串字段（*`long`*、*`double`* 等）–

    请注意，对非字符串字段进行排序时，需要在字段名称后面追加 `.value`，以将其与字符串字段区分开来。

    例如，在 SPARQL 查询中，您可以指定：

```
    neptune-fts:config neptune-fts:sortBy foaf:name.value .
```

在类似的 Gremlin 查询中，您可以指定：

```
    .withSideEffect('Neptune#fts.sortBy', 'name.value')
```

- `score` – 按匹配得分排序（默认）。

  如果 `sortOrder` 参数存在但 `sortBy` 不存在，则 `score` 按照由 `sortOrder` 指定的顺序对结果进行排序。

- `id` – 按 ID 排序，这意味着 SPARQL 主题 URI 或 Gremlin 顶点或边缘 ID。

  例如，在 SPARQL 查询中，您可以指定：

```
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
```

  在类似的 Gremlin 查询中，您可以指定：

```
    .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
```

- `label` – 按标签排序。

  例如，在 SPARQL 查询中，您可以指定：

```
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
```

  在类似的 Gremlin 查询中，您可以指定：

```
    .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
```

- `doc_type` – 按文档类型（即 SPARQL 或 Gremlin）排序。

  例如，在 SPARQL 查询中，您可以指定：

```
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
```

  在类似的 Gremlin 查询中，您可以指定：

```
    .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
```

默认情况下，OpenSearch 结果不进行排序，它们的顺序是不确定的，这意味着同一个查询在每次运行时都可能以不同的顺序返回项目。因此，如果结果集大于 `max_result_window`，则每次运行查询时都会返回总结果的一个非常不同的子集。但是，通过排序，您可以使不同运行的结果变得可更直接地进行比较。

如果没有 `sortOrder` 参数随附 `sortBy`，则使用从最大到最小的降序 (DESC)。

- **`sortOrder`** – 一个可选参数，用于指定 OpenSearch 结果是从最小到最大，还是从最大到最小排序（默认）：

  - `ASC` – 升序，从最小到最大。
  - `DESC` – 降序，从最大到最小。

    这是默认值，当 `sortBy` 参数存在但未指定 `sortOrder` 时使用。

  如果既不存在 `sortBy` 也不存在 `sortOrder`，则默认情况下不会对 OpenSearch 结果进行排序。

Amazon Neptune 中的非字符串 OpenSearch 索引

Amazon Neptune 中的非字符串 OpenSearch 索引编制允许使用流轮询器将谓词的非字符串值复制到 OpenSearch。然后，所有可以安全地转换为相应的 OpenSearch 映射或数据类型的谓词值都将复制到 OpenSearch。

要在新堆栈上启用非字符串索引编制，必须将 AWS CloudFormation 模板中的 `Enable Non-String Indexing` 标志设置为 `true`。这是默认设置。要更新现有堆栈以支持非字符串索引编制，请参阅下面的更新现有的堆栈。

> **Note**
>
> - 最好不要在 **`1.0.4.2`** 之前的引擎版本上启用非字符串索引编制。
> - OpenSearch 查询使用正则表达式查找与多个字段匹配的字段名，其中一些字段包含字符串值，另一些字段包含非字符串值，但查询会失败并返回错误。如果 Neptune 中的全文搜索查询属于这种类型，也会发生同样的情况。
> - 按非字符串字段排序时，请在字段名称后面附加“.value”，以将其与字符串字段区分开来。

目录

更新现有的 Neptune 全文搜索堆栈以支持非字符串索引编制

如果您已经在使用 Neptune 全文搜索，则需要采取以下步骤来支持非字符串索引编制：

1. 停止流轮询器 Lambda 函数。这样可以确保在导出过程中不会复制任何新的更新。为此，请禁用用于调用 Lambda 函数的云事件规则：
   - 在 AWS Management Console 中，导航到 CloudWatch。
   - 选择规则。
   - 选择带有 Lambda 流轮询器名称的规则。
   - 选择禁用以暂时禁用该规则。
2. 在 OpenSearch 中删除当前 Neptune 索引。使用以下 `curl` 查询从 OpenSearch 集群中删除 `amazon_neptune` 索引：

   ```
   curl -X DELETE "your OpenSearch endpoint/amazon_neptune"
   ```

3. 开始从 Neptune 一次性导出到 OpenSearch。此时最好设置一个新的 OpenSearch 堆栈，以便为执行导出的轮询器获取新的构件。

   按照 GitHub 中此处列出的步骤，开始将您的 Neptune 数据一次性导出到 OpenSearch 中。
4. 更新现有流轮询器的 Lambda 构件。成功完成将 Neptune 数据导出到 OpenSearch 后，请执行以下步骤：

- 在 AWS Management Console 中，导航到 AWS CloudFormation。
- 选择主要的父 AWS CloudFormation 堆栈。
- 为该堆栈选择更新选项。
- 选择从选项替换当前模板。
- 对于模板源，请选择 Amazon S3 URL。
- 对于 Amazon S3 URL，输入：

```
https://aws-neptune-customer-samples.s3.amazonaws.com/neptune-stream/
neptune_to_elastic_search.json
```

- 在不更改任何 AWS CloudFormation 参数的情况下选择下一步。
- 选择更新堆栈。AWS CloudFormation 将用最新的构件替换流轮询器的 Lambda 代码构件。

5. 再次启动流轮询器。为此，请启用相应的 CloudWatch 规则：

- 在 AWS Management Console 中，导航到 CloudWatch。
- 选择规则。
- 选择带有 Lambda 流轮询器名称的规则。
- 选择启用。

在 Neptune 全文搜索中筛选哪些字段已编制索引

AWS CloudFormation 模板详细信息中有两个字段可让您指定要从 OpenSearch 索引编制中排除的属性、谓词键或数据类型：

按属性或谓词名称筛选

您可以使用名为 `Properties to exclude from being inserted into Elastic Search Index` 的可选 AWS CloudFormation 模板参数，来提供要从 OpenSearch 索引编制中排除的以逗号分隔的属性或谓词键列表。

例如，假设您将此参数设置为 `bob`。

```
"Properties to exclude from being inserted into Elastic Search Index" : bob
```

在这种情况下，以下 Gremlin 更新查询的流记录将被删除，而不是进入索引：

```
g.V("1").property("bob", "test")
```

同样，您可以将此参数设置为 `http://my/example#bob`：

```
"Properties to exclude from being inserted into Elastic Search Index" : http://my/
example#bob
```

在这种情况下，以下 SPARQL 更新查询的流记录将被删除，而不是进入索引：

```
PREFIX ex: <http://my/example#>
INSERT DATA { ex:s1 ex:bob "test"}.
```

如果您未在此 AWS CloudFormation 模板参数中输入任何内容，则所有未以其它方式排除的属性键都将编制索引。

按属性或谓词值类型筛选

您可以使用名为 `Datatypes to exclude from being inserted into Elastic Search Index` 的可选 AWS CloudFormation 模板参数，来提供要从 OpenSearch 索引编制中排除的以逗号分隔的属性或谓词值数据类型列表。

对于 SPARQL，您无需列出完整的 XSD 类型 URI，只需列出数据类型令牌即可。您可以列出的有效数据类型令牌有：

- `string`
- `boolean`
- `float`
- `double`
- `dateTime`
- `date`
- `time`
- `byte`
- `short`
- `int`
- `long`
- `decimal`

- integer
- nonNegativeInteger
- nonPositiveInteger
- negativeInteger
- unsignedByte
- unsignedShort
- unsignedInt
- unsignedLong

对于 Gremlin，要列出的有效数据类型有：

- string
- date
- bool
- byte
- short
- int
- long
- float
- double

例如，假设您将此参数设置为 string。

```
"Datatypes to exclude from being inserted into Elastic Search Index" : string
```

在这种情况下，以下 Gremlin 更新查询的流记录将被删除，而不是进入索引：

```
g.V("1").property("myStringval", "testvalue")
```

同样，您可以将此参数设置为 int：

```
"Datatypes to exclude from being inserted into Elastic Search Index" : int
```

在这种情况下，以下 SPARQL 更新查询的流记录将被删除，而不是进入索引：

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
INSERT DATA { ex:s1 ex:bob "11"^^xsd:int }.
```

如果您未在此 AWS CloudFormation 模板参数中输入任何内容，则其值可以安全地转换为 OpenSearch 等效值的所有属性都将编制索引。查询语言不支持的列出类型将被忽略。

将 SPARQL 和 Gremlin 数据类型映射到 OpenSearch

OpenSearch 中的新数据类型映射是根据属性或对象中使用的数据类型创建的。由于某些字段包含不同类型的值，因此初始映射可能会排除该字段的某些值。

Neptune 数据类型映射到 OpenSearch 数据类型，如下所示：

SPARQL 类型	Gremlin 类型	OpenSearch 类型
XSD:int	byte	long
XSD:unsignedInt	short	
XSD:integer	int	
XSD:byte	long	
XSD:unsignedByte		
XSD:short		
XSD:unsignedShort		
XSD:long		
XSD:unsignedLong		
XSD:float	float	double
XSD:double	double	
XSD:decimal		
XSD:boolean	bool	boolean

SPARQL 类型	Gremlin 类型	OpenSearch 类型
XSD:datetime XSD:date	date	date
XSD:string XSD:time	string	text
自定义数据类型	不适用	text
任何其它数据类型	不适用	text

例如，以下 Gremlin 更新查询会导致向 OpenSearch 中添加“newField”的新映射，即 `{ "type" : "double" }`：

```
g.V("1").property("newField" 10.5)
```

同样，以下 SPARQL 更新查询会导致向 OpenSearch 中添加“ex:byte”的新映射，即 `{ "type" : "long" }`：

```
PREFIX ex: <http://my/example#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>

INSERT DATA { ex:test ex:byte "123"^^xsd:byte }.
```

> **Note**
>
> 如您所见，从 Neptune 映射到 OpenSearch 的项目在 OpenSearch 中的数据类型可能与它在 Neptune 中具有的数据类型不同。但是，OpenSearch 中有一个显式文本字段，即“datatype”，用于记录该项目在 Neptune 中的数据类型。

验证数据映射

使用以下过程将数据从 Neptune 复制到 OpenSearch：

- 如果 OpenSearch 中已经存在相关字段的映射：
  - 如果可以使用数据验证规则将数据安全地转换为现有映射，则将该字段存储在 OpenSearch 中。
  - 否则，请删除相应的流更新记录。
- 如果相关字段没有现有的映射，请在 Neptune 中查找与该字段的数据类型对应的 OpenSearch 数据类型。
  - 如果可以使用数据验证规则将字段数据安全地转换为 OpenSearch 数据类型，则将新的映射和字段数据存储在 OpenSearch 中。
  - 否则，请删除相应的流更新记录。

值是根据等效的 OpenSearch 类型或现有 OpenSearch 映射而不是根据 Neptune 类型进行验证的。例如，对于 `"123"^^xsd:int` 中的值 `"123"` 的验证是针对 `long` 类型而不是 `int` 类型进行的。

尽管 Neptune 尝试将所有数据复制到 OpenSearch，但在某些情况下，OpenSearch 中的数据类型与 Neptune 中的数据类型完全不同，在这种情况下，会跳过记录，而不是在 OpenSearch 中编制索引。

例如，在 Neptune 中，一个属性可以有多个不同类型的值，而在 OpenSearch 中，一个字段在索引中必须具有相同的类型。

通过启用调试日志，您可以查看在从 Neptune 导出到 OpenSearch 期间删除了哪些记录。调试日志条目的示例为：

```
Dropping Record : Data type not a valid Gremlin type
<Record>
```

数据类型按如下所示进行验证：

- **`text`** – Neptune 中的所有值都可以安全地映射到 OpenSearch 中的文本。
- **`long`** – 当 OpenSearch 映射类型为 long 时，Neptune 数据类型的以下规则适用（在以下示例中，假设 `"testLong"` 具有 `long` 映射类型）：
  - `boolean` – 无效，无法转换，并且会删除相应的流更新记录。

    无效的 Gremlin 示例为：

    ```
    "testLong" : true.
    "testLong" : false.
    ```

    无效的 SPARQL 示例为：

```
":testLong" : "true"^^xsd:boolean
":testLong" : "false"^^xsd:boolean
```

- `datetime` – 无效，无法转换，并且会删除相应的流更新记录。

  无效的 Gremlin 示例为：

  ```
  ":testLong" :  datetime('2018-11-04T00:00:00').
  ```

  无效的 SPARQL 示例为：

  ```
  ":testLong" : "2016-01-01"^^xsd:date
  ```

- `float`、`double` 或 `decimal` – 如果 Neptune 中的值是一个可以容纳 64 位的整数，则它是有效的，并且以 long 形式存储在 OpenSearch 中；但如果它有小数部分，或者是 `NaN` 或 `INF`，或者大于 9,223,372,036,854,775,807 或小于 -9,223,372,036,854,775,808，则它是无效的且相应的流更新记录被删除。

  有效的 Gremlin 示例为：

  ```
  "testLong" :  145.0.
  ":testLong" :  123
  ":testLong" :  -9223372036854775807
  ```

  有效的 SPARQL 示例为：

  ```
  ":testLong" : "145.0"^^xsd:float
  ":testLong" :  145.0
  ":testLong" : "145.0"^^xsd:double
  ":testLong" : "145.0"^^xsd:decimal
  ":testLong" : "-9223372036854775807"
  ```

  无效的 Gremlin 示例为：

  ```
  "testLong" :  123.45
  ":testLong" :  9223372036854775900
  ```

  无效的 SPARQL 示例为：

```
":testLong" :  123.45
":testLong" :  9223372036854775900
":testLong" : "123.45"^^xsd:float
":testLong" : "123.45"^^xsd:double
":testLong" : "123.45"^^xsd:decimal
```

- `string` – 如果 Neptune 中的值是可以包含在 64 位整数中的整数的字符串表示形式，则该值是有效的，并在 OpenSearch 中转换为 `long`。任何其它字符串值对于 Elasticseearch `long` 映射都无效，相应的数据流更新记录将被删除。

  有效的 Gremlin 示例为：

  ```
  "testLong" :  "123".
  ":testLong" :  "145.0"
  ":testLong" :  "-9223372036854775807"
  ```

  有效的 SPARQL 示例为：

  ```
  ":testLong" : "145.0"^^xsd:string
  ":testLong" : "-9223372036854775807"^^xsd:string
  ```

  无效的 Gremlin 示例为：

  ```
  "testLong" :  "123.45"
  ":testLong" :  "9223372036854775900"
  ":testLong" :  "abc"
  ```

  无效的 SPARQL 示例为：

  ```
  ":testLong" : "123.45"^^xsd:string
  ":testLong" : "abc"
  ":testLong" : "9223372036854775900"^^xsd:string
  ```

- **`double`** – 如果 OpenSearch 映射类型为 `double`，则适用以下规则（此处，假定“testDouble”字段在 OpenSearch 中具有 `double` 映射）：
  - `boolean` – 无效，无法转换，并且会删除相应的流更新记录。

    无效的 Gremlin 示例为：

```
  "testDouble" : true.
  "testDouble" : false.
```

无效的 SPARQL 示例为：

```
  ":testDouble" : "true"^^xsd:boolean
  ":testDouble" : "false"^^xsd:boolean
```

- `datetime` – 无效，无法转换，并且会删除相应的流更新记录。

  无效的 Gremlin 示例为：

  ```
    ":testDouble" :  datetime('2018-11-04T00:00:00').
  ```

  无效的 SPARQL 示例为：

  ```
    ":testDouble" : "2016-01-01"^^xsd:date
  ```

- 浮点 `NaN` 或 `INF` – 如果 SPARQL 中的值是浮点 `NaN` 或 `INF`，则该值无效，并且会删除相应的流更新记录。

  无效的 SPARQL 示例为：

  ```
  "  :testDouble" : "NaN"^^xsd:float
    ":testDouble" : "NaN"^^double
    ":testDouble" : "INF"^^double
    ":testDouble" : "-INF"^^double
  ```

- 数字或数字字符串 – 如果 Neptune 中的值是任何其它数字或数字的数字字符串表示（可以安全地表示为 `double`），那么它是有效的，并在 OpenSearch 中转换为 `double`。任何其它字符串值对于 OpenSearch `double` 映射都无效，并将删除相应的数据流更新记录。

  有效的 Gremlin 示例为：

  ```
    "testDouble" :  123
    ":testDouble" :  "123"
    ":testDouble" :  145.67
    ":testDouble" :  "145.67"
  ```

有效的 SPARQL 示例为：

```
  ":testDouble" :  123.45
  ":testDouble" :  145.0
  ":testDouble" : "123.45"^^xsd:float
  ":testDouble" : "123.45"^^xsd:double
  ":testDouble" : "123.45"^^xsd:decimal
  ":testDouble" : "123.45"^^xsd:string
```

无效的 Gremlin 示例为：

```
  ":testDouble" :  "abc"
```

无效的 SPARQL 示例为：

```
  ":testDouble" : "abc"
```

- **`date`** – 如果 OpenSearch 映射类型为 `date`，则 Neptune `date` 和 `dateTime` 值有效，任何可以成功解析为 `dateTime` 格式的字符串值也是有效的。

  Gremlin 或 SPARQL 中的有效示例为：

```
Date(2016-01-01)
"2016-01-01" "
2003-09-25T10:49:41"
"2003-09-25T10:49"
"2003-09-25T10"
"20030925T104941-0300"
"20030925T104941"
"2003-Sep-25" "
Sep-25-2003"
"2003.Sep.25"
"2003/09/25"
"2003 Sep 25" "
Wed, July 10, '96"
"Tuesday, April 12, 1952 AD 3:30:42pm PST"
"123"
"-123"
"0"
"-0"
"123.00"
```

```
  "-123.00"
```

无效的示例为：

```
  123.45
  True
  "abc"
```

Neptune 中的非字符串 OpenSearch 查询示例

Neptune 目前不直接支持 OpenSearch 范围查询。但是，您可以使用 Lucene 语法和 query-type="query_string" 实现相同的效果，如以下示例查询所示。

1. 获取所有使用期限大于 30 且名称以“Si”开头的顶点

在 Gremlin 中：

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.age.value:>30 && predicates.name.value:Si*');
```

在 SPARQL 中：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:>30 AND
 predicates.\\*foaf\\*name.value:Si*" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

为了简洁起见，这里使用 "*foaf*age 来代替完整的 URI。此正则表达式将检索 URI 中同时包含 foaf 和 age 的所有字段。

2. 获取所有使用期限介于 10 到 50 之间且名称与“Ronka”模糊匹配的节点

在 Gremlin 中：

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.age.value:[10 TO 50] AND
 predicates.name.value:Ronka~');
```

在 SPARQL 中：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\\*age.value:[10 TO 50] AND
 predicates.\\*foaf\\*name.value:Ronka~" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

3. 获取时间戳在过去 25 天内的所有节点

在 Gremlin 中：

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:>now-25d');
```

在 SPARQL 中：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://localhost:9200' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
```

```
    neptune-fts:config neptune-fts:query "predicates.\\*foaf\
\*timestamp.value:>now-25d~" .
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

4. 获取时间戳在给定年份和月份内的所有节点

在 Gremlin 中，使用 Lucene 语法中的日期数学表达式，对于 2020 年 12 月：

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:>2020-12');
```

Gremlin 的替代形式：

```
g.withSideEffect('Neptune#fts.endpoint', 'http://your-es-endpoint')
 .withSideEffect("Neptune#fts.queryType", "query_string")
 .V().has('*', 'Neptune#fts predicates.timestamp.value:[2020-12 TO 2021-01]');
```

Amazon Neptune 中的全文搜索查询执行

在包含全文搜索的查询中，Neptune 尝试首先将全文搜索调用放在查询的其它部分之前。这减少了对 OpenSearch 的调用次数，并且在大多数情况下显著提高了性能。然而，这绝非硬性规定。例如，在某些情况下，`PatternNode` 或 `UnionNode` 可能在全文搜索调用之前。

假设对包含 10 万个 `Person` 实例的数据库进行以下 Gremlin 查询：

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')
 .hasLabel('Person')
 .has('name', 'Neptune#fts marcello~');
```

如果此查询按步骤出现的顺序执行，则 10 万个解将流入 OpenSearch，从而导致数百个 OpenSearch 调用。实际上，Neptune 先调用 OpenSearch，然后将结果与 Neptune 的结果联接起来。在大多数情况下，这比按原始顺序执行查询要快得多。

您可以使用 noReordering 查询提示防止查询步骤执行采用这种重新排序：

```
g.withSideEffect('Neptune#fts.endpoint', 'your-es-endpoint-URL')
 .withSideEffect('Neptune#noReordering', true)
```

```
  .hasLabel('Person')
  .has('name', 'Neptune#fts marcello~');
```

在第二种情况下，首先执行 `.hasLabel` 步骤，其次执行 `.has('name', 'Neptune#fts marcello~')` 步骤。

对于另一个示例，假设针对相同类型的数据进行 SPARQL 查询：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT ?person WHERE {
  ?person rdf:type foaf:Person .
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mike' .
    neptune-fts:config neptune-fts:return ?person .
  }
}
```

再次，Neptune 首先执行查询的 `SERVICE` 部分，然后将结果与 `Person` 数据联接。您可以使用 joinOrder 查询提示禁止此行为：

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
PREFIX hint: <http://aws.amazon.com/neptune/vocab/v01/QueryHints#>
SELECT ?person WHERE {
  hint:Query hint:joinOrder "Ordered" .
  ?person rdf:type foaf:Person .
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mike' .
    neptune-fts:config neptune-fts:return ?person .
  }
}
```

同样，在第二个查询中，查询的各个部分按照它们在查询中显示的顺序执行。

Neptune 中使用全文搜索的示例 SPARQL 查询

下面是一些在 Amazon Neptune 中使用全文搜索的示例 SPARQL 查询。

SPARQL 匹配查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'match' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'michael' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 前缀查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'prefix' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mich' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 模糊查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'fuzzy' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'mikael' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 术语查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'term' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:query 'Dr. Kunal' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL query_string 查询示例

此查询指定多个字段。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ OR rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:field foaf:surname .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL simple_query_string 查询示例

以下查询使用通配符 ('*') 指定字段。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint.com' .
    neptune-fts:config neptune-fts:queryType 'simple_query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
```

```
    neptune-fts:config neptune-fts:field '*' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 按字符串排序字段查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy foaf:name .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 按非字符串字段排序查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name.value .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy dc:date.value .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 按 ID 排序查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
```

```
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_id' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 按标签排序查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.entity_type' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

SPARQL 按 doc_type 排序查询示例

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'mikael~ | rondelli' .
    neptune-fts:config neptune-fts:field foaf:name .
    neptune-fts:config neptune-fts:sortOrder 'asc' .
    neptune-fts:config neptune-fts:sortBy 'Neptune#fts.document_type' .
    neptune-fts:config neptune-fts:return ?res .
  }
```

```
}
```

在 SPARQL 中使用 Lucene 语法的示例

只有 OpenSearch 中的 `query_string` 查询支持 Lucene 语法。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX neptune-fts: <http://aws.amazon.com/neptune/vocab/v01/services/fts#>
SELECT * WHERE {
  SERVICE neptune-fts:search {
    neptune-fts:config neptune-fts:endpoint 'http://your-es-endpoint' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:queryType 'query_string' .
    neptune-fts:config neptune-fts:query 'predicates.\\foaf\\name.value:micheal AND
 predicates.\\foaf\\surname.value:sh' .
    neptune-fts:config neptune-fts:field '' .
    neptune-fts:config neptune-fts:return ?res .
  }
}
```

在 Gremlin 查询中使用 Neptune 全文搜索

对未转换为 Neptune 步骤的 Gremlin 遍历部分，NeptuneSearchStep 启用全文搜索查询。例如，请考虑以下查询：

```
g.withSideEffect("Neptune#fts.endpoint", "your-es-endpoint-URL")
  .V()
      .tail(100)
      .has("name", "Neptune#fts mark*")            <== # Limit the search on name
```

此查询在 Neptune 中将转换为以下优化遍历。

```
Neptune steps:
[
    NeptuneGraphQueryStep(Vertex) {
        JoinGroupNode {
            PatternNode[(?1, <~label>, ?2, <~>) . project distinct ?1 .],
 {estimatedCardinality=INFINITY}
        }, annotations={path=[Vertex(?1):GraphStep], maxVarId=4}
    },
    NeptuneTraverserConverterStep
```

```
]
+ not converted into Neptune steps: [NeptuneTailGlobalStep(100),
 NeptuneTinkerpopTraverserConverterStep, NeptuneSearchStep {
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=your-OpenSearch-endpoint-URL}
    }
    JoinGroupNode {
        SearchNode[(idVar=?3, query=mark*, field=name) . project ask .],
 {endpoint=your-OpenSearch-endpoint-URL}
    }
}]
```

以下是 Gremlin 对航空航线数据的查询示例：

不区分大小写的 Gremlin 基本 **match** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'match')
  .V().has("city","Neptune#fts dallas")

==>v[186]
==>v[8]
```

Gremlin **match** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'match')
  .V().has("city","Neptune#fts southampton")
     .local(values('code','city').fold())
     .limit(5)

==>[SOU, Southampton]
```

Gremlin **fuzzy** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has("city","Neptune#fts allas~").values('city').limit(5)
```

```
==>Dallas
==>Dallas
==>Walla Walla
==>Velas
==>Altai
```

Gremlin **query_string** 模糊查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has("city","Neptune#fts allas~").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin **query_string** 正则表达式查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has("city","Neptune#fts /[dp]allas/").values('city').limit(5)

==>Dallas
==>Dallas
```

Gremlin 混合查询

此查询在同一查询中使用 Neptune 内部索引和 OpenSearch 索引。

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has("region","GB-ENG")
      .has('city','Neptune#fts L*')
      .values('city')
      .dedup()
      .limit(10)

==>London
==>Leeds
```

```
==>Liverpool
==>Land's End
```

简单 Gremlin 全文搜索示例

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .V().has('desc','Neptune#fts regional municipal')
      .local(values('code','desc').fold())
      .limit(100)

==>[HYA, Barnstable Municipal Boardman Polando Field]
==>[SPS, Sheppard Air Force Base-Wichita Falls Municipal Airport]
==>[ABR, Aberdeen Regional Airport]
==>[SLK, Adirondack Regional Airport]
==>[BFD, Bradford Regional Airport]
==>[EAR, Kearney Regional Airport]
==>[ROT, Rotorua Regional Airport]
==>[YHD, Dryden Regional Airport]
==>[TEX, Telluride Regional Airport]
==>[WOL, Illawarra Regional Airport]
==>[TUP, Tupelo Regional Airport]
==>[COU, Columbia Regional Airport]
==>[MHK, Manhattan Regional Airport]
==>[BJI, Bemidji Regional Airport]
==>[HAS, Hail Regional Airport]
==>[ALO, Waterloo Regional Airport]
==>[SHV, Shreveport Regional Airport]
==>[ABI, Abilene Regional Airport]
==>[GIZ, Jizan Regional Airport]
==>[USA, Concord Regional Airport]
==>[JMS, Jamestown Regional Airport]
==>[COS, City of Colorado Springs Municipal Airport]
==>[PKB, Mid Ohio Valley Regional Airport]
```

使用 **query_string** 以及“+”和“-”运算符的 Gremlin 查询

尽管 query_string 查询类型比默认 simple_query_string 类型要严格得多，但它确实可以获得更精确的查询结果。下面的第一个查询使用 query_string，而第二个查询使用默认的 simple_query_string：

```
g.withSideEffect("Neptune#fts.endpoint",
```

```
                  "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
 . V().has('desc','Neptune#fts +London -(Stansted|Gatwick)')
      .local(values('code','desc').fold())
      .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
```

请注意以下示例中的 simple_query_string 如何悄悄地忽略“+”和“-”运算符：

```
g.withSideEffect("Neptune#fts.endpoint",
                  "your-OpenSearch-endpoint-URL")
  .V().has('desc','Neptune#fts +London -(Stansted|Gatwick)')
      .local(values('code','desc').fold())
      .limit(10)

==>[LHR, London Heathrow]
==>[YXU, London Airport]
==>[LGW, London Gatwick]
==>[STN, London Stansted Airport]
==>[LTN, London Luton Airport]
==>[SEN, London Southend Airport]
==>[LCY, London City Airport]
==>[SKG, Thessaloniki Macedonia International Airport]
==>[ADB, Adnan Menderes International Airport]
==>[BTV, Burlington International Airport]
```

```
g.withSideEffect("Neptune#fts.endpoint",
                  "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc','Neptune#fts +(regional|municipal) -(international|bradford)')
      .local(values('code','desc').fold())
      .limit(10)

==>[CZH, Corozal Municipal Airport]
==>[MMU, Morristown Municipal Airport]
==>[YBR, Brandon Municipal Airport]
==>[RDD, Redding Municipal Airport]
==>[VIS, Visalia Municipal Airport]
```

```
==>[AIA, Alliance Municipal Airport]
==>[CDR, Chadron Municipal Airport]
==>[CVN, Clovis Municipal Airport]
==>[SDY, Sidney Richland Municipal Airport]
==>[SGU, St George Municipal Airport]
```

使用 **AND** 和 **OR** 运算符的 Gremlin **query_string** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'query_string')
  .V().has('desc','Neptune#fts (St AND George) OR (St AND Augustin)')
      .local(values('code','desc').fold())
      .limit(10)

==>[YIF, St Augustin Airport]
==>[STG, St George Airport]
==>[SGO, St George Airport]
==>[SGU, St George Municipal Airport]
```

Gremlin **term** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'term')
  .V().has("SKU","Neptune#fts ABC123DEF9")
      .local(values('code','city').fold())
      .limit(5)

==>[AUS, Austin]
```

Gremlin **prefix** 查询

```
g.withSideEffect("Neptune#fts.endpoint",
                 "your-OpenSearch-endpoint-URL")
  .withSideEffect('Neptune#fts.queryType', 'prefix')
  .V().has("icao","Neptune#fts ka")
      .local(values('code','icao','city').fold())
      .limit(5)

==>[AZO, KAZO, Kalamazoo]
```

```
==>[APN, KAPN, Alpena]
==>[ACK, KACK, Nantucket]
==>[ALO, KALO, Waterloo]
==>[ABI, KABI, Abilene]
```

在 Neptune Gremlin 中使用 Lucene 语法

在 Neptune Gremlin 中，您还可以使用 Lucene 查询语法编写功能非常强大的查询。请注意，只有 OpenSearch 中的 `query_string` 查询支持 Lucene 语法。

假设以下数据：

```
g.addV("person")
        .property(T.id, "p1")
        .property("name", "simone")
        .property("surname", "rondelli")

g.addV("person")
        .property(T.id, "p2")
        .property("name", "simone")
        .property("surname", "sengupta")

g.addV("developer")
        .property(T.id, "p3")
        .property("name", "simone")
        .property("surname", "rondelli")
```

使用 Lucene 语法（当 `queryType` 为 `query_string` 时调用），您可以按名字和姓氏搜索此数据，如下所示：

```
g.withSideEffect("Neptune#fts.endpoint", "es_endpoint")
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts predicates.name.value:simone AND
 predicates.surname.value:rondelli")

==> v[p1], v[p3]
```

请注意，在上面的 `has()` 步骤中，此字段替换为 `"*"`。实际上，放在此处的任何值都会被您在查询中访问的字段所覆盖。您可以使用 `predicates.name.value`，访问名字字段，因为这是数据模型的结构方式。

您可以按名字、姓氏和标签进行搜索，如下所示：

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts predicates.name.value:simone AND
 predicates.surname.value:rondelli AND entity_type:person")

==> v[p1]
```

该标签使用 entity_type 进行访问，也是因为这是数据模型的结构方式。

您还可以包括嵌套条件：

```
g.withSideEffect("Neptune#fts.endpoint", getEsEndpoint())
    .withSideEffect("Neptune#fts.queryType", "query_string")
    .V()
    .has("*", "Neptune#fts (predicates.name.value:simone AND
 predicates.surname.value:rondelli AND entity_type:person) OR
 predicates.surname.value:sengupta")

==> v[p1], v[p2]
```

插入现代 TinkerPop 图形

```
g.addV('person').property(T.id, '1').property('name', 'marko').property('age', 29)
 .addV('personr').property(T.id, '2').property('name', 'vadas').property('age', 27)
 .addV('software').property(T.id, '3').property('name', 'lop').property('lang', 'java')
 .addV('person').property(T.id, '4').property('name', 'josh').property('age', 32)
 .addV('software').property(T.id, '5').property('name', 'ripple').property('lang',
 'java')
 .addV('person').property(T.id, '6').property('name', 'peter').property('age', 35)

g.V('1').as('a').V('2').as('b').addE('knows').from('a').to('b').property('weight',
 0.5f).property(T.id, '7')
 .V('1').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.4f).property(T.id, '9')
 .V('4').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.4f).property(T.id, '11')
 .V('4').as('a').V('5').as('b').addE('created').from('a').to('b').property('weight',
 1.0f).property(T.id, '10')
```

```
 .V('6').as('a').V('3').as('b').addE('created').from('a').to('b').property('weight',
 0.2f).property(T.id, '12')
 .V('1').as('a').V('4').as('b').addE('knows').from('a').to('b').property('weight',
 1.0f).property(T.id, '8')
```

按字符串字段值排序示例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
 .withSideEffect('Neptune#fts.queryType', 'query_string')
 .withSideEffect('Neptune#fts.sortOrder', 'asc')
 .withSideEffect('Neptune#fts.sortBy', 'name')
 .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

按非字符串字段值排序示例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
 .withSideEffect('Neptune#fts.queryType', 'query_string')
 .withSideEffect('Neptune#fts.sortOrder', 'asc')
 .withSideEffect('Neptune#fts.sortBy', 'age.value')
 .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

按 ID 字段值排序示例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
.withSideEffect('Neptune#fts.queryType', 'query_string')
.withSideEffect('Neptune#fts.sortOrder', 'asc')
.withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_id')
.V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

按标签字段值排序示例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
 .withSideEffect('Neptune#fts.queryType', 'query_string')
 .withSideEffect('Neptune#fts.sortOrder', 'asc')
 .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.entity_type')
 .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

按 document_type 字段值排序示例

```
g.withSideEffect("Neptune#fts.endpoint", "your-OpenSearch-endpoint-URL")
 .withSideEffect('Neptune#fts.queryType', 'query_string')
 .withSideEffect('Neptune#fts.sortOrder', 'asc')
 .withSideEffect('Neptune#fts.sortBy', 'Neptune#fts.document_type')
 .V().has('name', 'Neptune#fts marko OR vadas OR ripple')
```

Neptune 全文搜索故障排除

> **Note**
>
> 如果您已在 OpenSearch 集群上启用了精细访问控制，则还需要在 Neptune 数据库中启用 IAM 身份验证。

要诊断从 Neptune 复制到 OpenSearch 时遇到的问题，请查阅轮询器 Lambda 函数的 CloudWatch Logs。这些日志提供了有关从流读取的记录数以及成功复制到 OpenSearch 的记录数的详细信息。

您还可以通过更改 `LoggingLevel` 环境变量来更改 Lambda 函数的日志记录级别。

> **Note**
>
> 如果 `LoggingLevel` 设置为 `DEBUG`，则当 StreamPoller 将数据从 Neptune 复制到 OpenSearch 时，您可以查看其它详细信息，例如删除的流记录以及每个流记录被删除的原因。如果您发现自己缺少记录，这会很有用。

Neptune 流使用者应用程序会在 CloudWatch 上发布两个指标，这两个指标也可以帮助您诊断问题：

- `StreamRecordsProcessed` – 应用程序每单位时间处理的记录数。有助于跟踪应用程序运行速率。
- `StreamLagTime` – 当前时间与正处理的流记录的提交时间之间的时差（以毫秒为单位）。此指标显示了使用者应用程序落后的程度。

此外，与复制进程相关的所有指标都会在 CloudWatch 的控制面板中显示，其名称与您使用 CloudWatch 模板实例化应用程序时提供的 `ApplicationName` 相同。

您还可以选择创建一个 CloudWatch 警报，每当轮询连续失败超过两次时触发该警报。通过在实例化应用程序时将 `CreateCloudWatchAlarm` 字段设置为 `true` 来执行此操作。然后指定您希望在触发警报时收到通知的电子邮件地址。

对从流读取记录时失败的进程排除故障

如果进程在从流读取记录时失败，请确保您符合以下条件：

- 该流已在您的集群上启用。
- Neptune 流端点采用正确的格式：
  - 对于 Gremlin 或 openCypher：`https://`*`your cluster endpoint`*`:`*`your cluster port`*`/propertygraph/stream` 或它的别名 `https://`*`your cluster endpoint`*`:`*`your cluster port`*`/pg/stream`
  - 对于 SPARQL：`https://`*`your cluster endpoint`*`:`*`your cluster port`*`/sparql/stream`
- 已为您的 VPC 配置了 DynamoDB 端点。
- 已为您的 VPC 子网配置了监控终端节点。

对向 OpenSearch 写入数据时失败的进程进行故障排除

如果进程在向 OpenSearch 写入记录时失败，请确保您符合以下条件：

- 您的 OpenSearch 版本为 7.1 或更高版本，或者为 Opensearch 2.3 及更高版本
- 可以在 VPC 中通过轮询器 Lambda 函数访问 OpenSearch。
- 附加到 OpenSearch 的安全策略允许入站 HTTP/HTTPS 请求。

在现有复制设置中修复 Neptune 和 OpenSearch 之间不同步的问题

您可以使用以下步骤让 Neptune 数据库和 OpenSearch 域与最新数据恢复同步，以防它们之间因 `ExpiredStreamException` 或数据损坏而出现不同步问题。

请注意，这种方法会删除 OpenSearch 域中的所有数据，然后将其从 Neptune 数据库的当前状态重新同步，因此无需在 Neptune 数据库中重新加载任何数据。

1. 按照禁用（暂停）流轮询器进程所述禁用复制过程。
2. 使用以下命令删除 OpenSearch 域上的 Neptune 索引：

```
curl -X DELETE "(your OpenSearch endpoint)/amazon_neptune"
```

3. 创建数据库的克隆（请参阅 Neptune 中的数据库克隆）。
4. 通过对 Streams API 端点执行此类命令，获取克隆数据库上流的最新 eventID（有关更多信息，请参阅调用 Neptune Streams REST API）：

   ```
   curl "https://(your neptune endpoint):(port)/(propertygraph or sparql)/stream?
   iteratorType=LATEST"
   ```

   记下响应的 lastEventId 对象中 commitNum 和 opNum 字段的值。
5. 使用 github 上的 export-neptune-to-elasticsearch 工具执行从克隆的数据库到 OpenSearch 域的一次性同步。
6. 转至复制堆栈的 DynamoDB 表。表的名称将是您在 AWS CloudFormation 模板中指定的应用程序名称（默认为 NeptuneStream），并带有 -LeaseTable 后缀。换而言之，默认表名称为 NeptuneStream-LeaseTable。

   您可以通过扫描来浏览表行，因为表中应该只有一行。使用上面记录的 commitNum 和 opNum 值进行以下更改：

   - 将表中 checkpoint 字段的值更改为您为 commitNum 记下的值。
   - 将表中 checkpointSubSequenceNumber 字段的值更改为您为 opNum 记下的值。
7. 按照重新启用流轮询器进程所述重新启用复制过程。
8. 删除克隆的数据库和为 export-neptune-to-elasticsearch 工具创建的 AWS CloudFormation 堆栈。

在 Amazon Neptune 中使用 AWS Lambda 函数

AWS Lambda 函数在 Amazon Neptune 应用程序中有许多用途。在这里，我们提供了将 Lambda 函数与任何流行的 Gremlin 驱动程序和语言变体一起使用的一般指南，以及用 Java、JavaScript 和 Python 编写的 Lambda 函数的具体示例。

> **Note**
>
> 在 Neptune 中使用 Lambda 函数的最佳方式随着最近发布的引擎而发生了变化。过去，Neptune 在回收 Lambda 执行上下文很久之后就会让空闲连接保持打开状态，这可能会导致服务器上的资源泄漏。为了缓解这种情况，我们过去建议在每次 Lambda 调用时打开和关闭连接。但是，从引擎版本 1.0.3.0 开始，空闲连接超时已缩短，以便在回收非活动的 Lambda 执行上下文后，连接不再泄漏，因此我们现在建议在执行上下文期间使用单个连接。这应该包括一些错误处理以及回退并重试样板代码，以处理意外关闭的连接。

在 AWS Lambda 函数中管理 Gremlin WebSocket 连接

如果您使用 Gremlin 语言变体来查询 Neptune，则驱动程序将使用 WebSocket 连接来连接到数据库。WebSocket 旨在支持长寿命的客户端/服务器连接方案。另一方面，AWS Lambda 旨在支持相对短寿命和无状态的执行。在使用 Lambda 查询 Neptune 时，这种设计理念的不匹配可能会导致一些意想不到的问题。

AWS Lambda 函数在执行环境中运行，该环境将该函数与其它函数隔离开来。执行上下文是在第一次调用该函数时创建的，并且可以重用于后续调用同一函数。

但是，任何一个执行上下文都不会用于处理函数的多个并发调用。如果您的函数由多个客户端同时调用，Lambda 会为该函数的每个实例启动一个额外的执行上下文。反过来，所有这些新的执行上下文可重用于该函数的后续调用。

在某个时候，Lambda 会回收执行上下文，尤其是在它们已经处于非活动状态一段时间的情况下。AWS Lambda 通过 Lambda 扩展公开执行上下文生命周期，包括 `Init`、`Invoke` 和 `Shutdown` 阶段。使用这些扩展，您可以编写在回收执行上下文时清理外部资源（例如数据库连接）的代码。

常见的最佳实践是在 Lambda 处理程序函数之外打开数据库连接，以便每次调用处理程序时都可重用该连接。如果数据库连接在某个时候断开，则可以从处理程序内部重新连接。但是，这种方法存在连接泄露的危险。如果空闲连接在执行上下文被破坏后很长一段时间仍处于打开状态，则间歇性或突发性 Lambda 调用场景可能会逐渐泄漏连接并耗尽数据库资源。

随着更高的引擎版本推出，Neptune 连接限制和连接超时已发生变化。以前，每个实例最多支持 60000 个 WebSocket 连接。现在，每个 Neptune 实例的最大并发 WebSocket 连接数因实例类型而异。

此外，从引擎版本 1.0.3.0 开始，Neptune 将连接的空闲超时从一小时缩短到大约 20 分钟。如果客户端未关闭连接，则在空闲超时 20 到 25 分钟后，连接会自动关闭。AWS Lambda 不记录执行上下文生命周期，但实验表明，新的 Neptune 连接超时与非活动的 Lambda 执行上下文超时非常吻合。当不活跃的执行上下文被回收时，Neptune 很有可能已经关闭了它的连接，或者很快就会关闭。

将 AWS Lambda 与 Amazon Neptune Gremlin 结合使用的建议

现在，我们建议在 Lambda 执行上下文的整个生命周期内使用单个连接和图形遍历源，而不是为每个函数调用使用一个连接和图形遍历源（每个函数调用仅处理一个客户端请求）。由于并发客户端请求是由在不同执行上下文中运行的不同函数实例处理的，因此无需维护连接池来处理函数实例内的并发请求。如果您使用的 Gremlin 驱动程序有连接池，请将其配置为仅使用一个连接。

要处理连接失败，请围绕每个查询使用重试逻辑。尽管目标是在执行上下文的生命周期内保持单个连接，但意外的网络事件可能会导致该连接突然终止。此类连接失败表现为不同的错误，具体取决于您使用的驱动程序。您应该对 Lambda 函数进行编码，以处理这些连接问题，并在必要时尝试重新连接。

某些 Gremlin 驱动程序会自动处理重新连接。例如，Java 驱动程序会自动尝试代表您的客户端代码重新建立与 Neptune 的连接。使用此驱动程序，您的函数代码只需要退出并重试查询即可。相比之下，JavaScript 和 Python 驱动程序不实现任何自动重新连接逻辑，因此使用这些驱动程序，您的函数代码必须在回退后尝试重新连接，并且只有在重新建立连接后才会重试查询。

这里的代码示例确实包括重新连接逻辑，而不是假设客户端正在处理它。

在 Lambda 中使用 Gremlin 写入请求的建议

如果您的 Lambda 函数修改了图形数据，请考虑采用回退并重试策略来处理以下异常：

- **`ConcurrentModificationException`** – Neptune 事务语义意味着写入请求有时会失败并引发 `ConcurrentModificationException`。在这些情况下，请尝试使用基于指数回退的重试机制。
- **`ReadOnlyViolationException`** – 由于计划内或计划外的事件会导致集群拓扑随时可能发生变化，因此，写入职责可能会从集群中的一个实例迁移到另一个实例。如果您的函数代码尝试向不再是主（写入器）实例的实例发送写入请求，则请求将失败并引发 `ReadOnlyViolationException`。发生这种情况时，请关闭现有连接，重新连接到集群端点，然后重试请求。

另外，如果您使用回退并重试策略来处理写入请求问题，请考虑为创建和更新请求实现幂等性查询（例如，使用 fold().coalesce().unfold()）。

在 Lambda 中使用 Gremlin 读取请求的建议

如果您的集群中有一个或多个只读副本，则最好在这些副本之间平衡读取请求。一种选择是使用读取器端点。即使在您添加或删除副本或将副本提升为新的主实例时，集群拓扑发生了变化，读取器端点也会平衡副本间的连接。

但是，在某些情况下，使用读取器端点可能会导致集群资源的使用不均衡。读取器端点的工作方式是定期更改 DNS 条目指向的主机。如果客户端在 DNS 条目更改之前打开了大量连接，则所有连接请求都将发送到单个 Neptune 实例。高吞吐量 Lambda 场景可能就是这种情况，在这种场景中，对 Lambda 函数的大量并发请求会导致创建多个执行上下文，每个上下文都有自己的连接。如果这些连接几乎同时创建，则这些连接很可能都指向集群中的同一个副本，并且一直指向该副本，直到执行上下文被回收为止。

跨实例分配请求的一种方法是将 Lambda 函数配置为连接到从副本实例端点列表中随机选择的实例端点，而不是读取器端点。这种方法的缺点是，它要求 Lambda 代码通过监控集群并在集群成员资格发生变化时更新端点列表，来处理集群拓扑的变化。

如果您正在编写需要在集群中的实例之间平衡读取请求的 Java Lambda 函数，则可以使用适用于 Amazon Neptune 的 Gremlin 客户端，这是一款 Java Gremlin 客户端，它知道您的集群拓扑，可以公平地在 Neptune 集群中的一组实例间分配连接和请求。这篇博客文章包括一个使用适用于 Amazon Neptune 的 Gremlin 客户端的 Java Lambda 函数示例。

可能减慢 Neptune Gremlin Lambda 函数冷启动速度的因素

第一次调用 AWS Lambda 函数称为冷启动。有几个因素会增加冷启动的延迟：

- 请务必为您的 Lambda 函数分配足够的内存。 – Lambda 函数在冷启动期间的编译速度可能比在 EC2 上慢得多，因为 AWS Lambda 按您分配给该函数的内存成比例线性分配 CPU 周期。在内存为 1,792MB 时，函数收到相当于 1 个完整 vCPU（每秒一个 vCPU 秒的积分）的处理能力。对于用 Java 编写的大型 Lambda 函数，未分配足够的内存来接收足够的 CPU 周期的影响尤其明显。
- 请注意，启用 IAM 数据库身份验证可能会减慢冷启动速度 – AWS Identity and Access Management (IAM) 数据库身份验证还会减慢冷启动速度，尤其是在 Lambda 函数必须生成新的签名密钥的情况下。这种延迟仅影响冷启动，而不影响后续请求，因为一旦 IAM 数据库身份验证建立了连接凭证，Neptune 只会定期验证这些凭证是否仍然有效。

使用 AWS CloudFormation 创建要在 Neptune 中使用的 Lambda 函数

您可以使用 AWS CloudFormation 模板创建可以访问 Neptune 的 AWS Lambda 函数。

1. 要在 AWS CloudFormation 控制台中启动 Lambda 函数堆栈，请选择下表中的启动堆栈按钮之一。

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	
美国东部（俄亥俄州）	视图	在 Designer 中查看	
美国西部（北加利福尼亚）	视图	在 Designer 中查看	
美国西部（俄勒冈州）	视图	在 Designer 中查看	
加拿大（中部）	视图	在 Designer 中查看	
南美洲（圣保罗）	视图	在 Designer 中查看	
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	
欧洲地区（爱尔兰）	视图	在 Designer 中查看	
欧洲地区（伦敦）	视图	在 Designer 中查看	
欧洲地区（巴黎）	视图	在 Designer 中查看	

区域	查看	在 Designer 中查看	发布
欧洲地区（法兰克福）	视图	在 Designer 中查看	
中东（巴林）	视图	在 Designer 中查看	
中东（阿联酋）	视图	在 Designer 中查看	
以色列（特拉维夫）	视图	在 Designer 中查看	
非洲（开普敦）	视图	在 Designer 中查看	
亚太地区（香港）	视图	在 Designer 中查看	
亚太地区（东京）	视图	在 Designer 中查看	
亚太地区（首尔）	视图	在 Designer 中查看	
亚太地区（新加坡）	视图	在 Designer 中查看	
亚太地区（悉尼）	视图	在 Designer 中查看	
亚太地区（孟买）	视图	在 Designer 中查看	
中国（北京）	视图	在 Designer 中查看	
中国（宁夏）	视图	在 Designer 中查看	
AWS GovCloud（美国西部）	视图	在 Designer 中查看	

区域	查看	在 Designer 中查看	发布
AWS GovCloud（美国东部）	视图	在 Designer 中查看	Launch Stack

2. 在 Select Template 页面上，选择 Next。
3. 在指定详细信息页面上，设置以下选项：

   a. 根据您要在 lambda 函数中使用的语言，选择 lambda 运行时系统。这些 AWS CloudFormation 模板目前支持以下语言：

      - Python 3.9（映射到 Amazon S3 URL 中的 `python39`）
      - NodeJS 18（映射到 Amazon S3 URL 中的 `nodejs18x`）
      - Ruby 2.5（映射到 Amazon S3 URL 中的 `ruby25`）

   b. 提供相应的 Neptune 集群端点和端口号。

   c. 提供相应的 Neptune 安全组。

   d. 提供相应的 Neptune 子网参数。
4. 选择下一步。
5. 在选项页面上，选择下一步。
6. 在审核页面上，选中第一个复选框以确认 AWS CloudFormation 将创建 IAM 资源。

   然后选择创建。

如果您需要对 lambda 运行时系统进行自己的更改，则可以从您区域中的 Amazon S3 位置下载一个通用的运行时系统：

```
https://s3.Amazon region.amazonaws.com/aws-neptune-customer-samples-Amazon region/
lambda/runtime-language/lambda_function.zip.
```

例如：

```
https://s3.us-west-2.amazonaws.com/aws-neptune-customer-samples-us-west-2/lambda/
python36/lambda_function.zip
```

Amazon Neptune 的 AWS Lambda 函数示例

以下示例 AWS Lambda 函数是用 Java、JavaScript 和 Python 编写的，说明了使用 `fold().coalesce().unfold()` 习惯用法用随机生成的 ID 对单个顶点进行更新插入。

每个函数中的大部分代码是样板代码，负责管理连接，并在出现错误时重试连接和查询。实际的应用程序逻辑和 Gremlin 查询分别在 `doQuery()` 和 `query()` 方法中实现。如果您使用这些示例作为自己的 Lambda 函数的基础，则可以专注于修改 `doQuery()` 和 `query()`。

这些函数配置为重试失败的查询 5 次，两次重试之间等待 1 秒钟。

这些函数要求值必须存在于以下 Lambda 环境变量中：

- **`NEPTUNE_ENDPOINT`** – 您的 Neptune 数据库集群端点。对于 Python 来说，这应该是 `neptuneEndpoint`。
- **`NEPTUNE_PORT`** – Neptune 端口。对于 Python 来说，这应该是 `neptunePort`。
- **`USE_IAM`** – (`true` 或 `false`) 如果您的数据库启用了 AWS Identity and Access Management (IAM) 数据库身份验证，请将 `USE_IAM` 环境变量设置为 `true`。这会导致 Lambda 函数对指向 Neptune 的连接请求进行 Sigv4 签名。对于此类 IAM 数据库身份验证请求，请确保 Lambda 函数的执行角色附加了相应的 IAM policy，允许该函数连接到您的 Neptune 数据库集群（请参阅IAM policy 的类型）。

Amazon Neptune 的 Java Lambda 函数示例

以下是使用 Java AWS Lambda 函数时要记住的一些事项：

- Java 驱动程序会维护其自己的连接池，而您不需要该连接池，因此请使用 `minConnectionPoolSize(1)` 和 `maxConnectionPoolSize(1)` 配置您的 `Cluster` 对象。
- `Cluster` 对象的构建速度可能很慢，因为它会创建一个或多个序列化器 [默认情况下为 Gyro，如果您已将其配置为其它输出格式（例如 `binary`），则会创建另一个序列化器]。这些可能需要一段时间才能实例化。
- 连接池使用第一个请求进行初始化。此时，如果您使用的是 IAM 数据库身份验证，驱动程序会设置 `Netty` 堆栈、分配字节缓冲区并创建签名密钥。所有这些都可能增加冷启动延迟。
- Java 驱动程序的连接池监控服务器主机的可用性，并在连接失败时自动尝试重新连接。它会启动后台任务以尝试重新建立连接。使用 `reconnectInterval( )` 配置尝试重新连接的间隔。当驱动程序尝试重新连接时，您的 Lambda 函数只需重试查询即可。

如果重试之间的间隔小于重新连接尝试之间的间隔，则在连接失败时重试会再次失败，因为主机被认为不可用。这不适用于在引发 `ConcurrentModificationException` 时重试。

- 使用 Java 8 而不是 Java 11。在 Java 11 中，默认情况下不启用 `Netty` 优化。
- 此示例使用 Retry4j 进行重试。
- 要在 Java Lambda 函数中使用 `Sigv4` 签名驱动程序，请参阅使用 Java 和 Gremlin 及签名版本 4 签名连接到 Neptune中的依赖项要求。

> **Warning**
>
> 来自 Retry4j 的 `CallExecutor` 可能不是线程安全的。考虑让每个线程使用自己的 `CallExecutor` 实例。

```
package com.amazonaws.examples.social;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.evanlennick.retry4j.CallExecutor;
import com.evanlennick.retry4j.CallExecutorBuilder;
import com.evanlennick.retry4j.Status;
import com.evanlennick.retry4j.config.RetryConfig;
import com.evanlennick.retry4j.config.RetryConfigBuilder;
import org.apache.tinkerpop.gremlin.driver.Cluster;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.neptune.auth.NeptuneNettyHttpSigV4Signer;
import org.apache.tinkerpop.gremlin.driver.remote.DriverRemoteConnection;
import org.apache.tinkerpop.gremlin.driver.ser.Serializers;
import org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource;
import org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.GraphTraversalSource;
import org.apache.tinkerpop.gremlin.structure.T;

import java.io.*;
import java.time.temporal.ChronoUnit;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.function.Function;
```

```
import static java.nio.charset.StandardCharsets.UTF_8;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.addV;
import static org.apache.tinkerpop.gremlin.process.traversal.dsl.graph.__.unfold;

public class MyHandler implements RequestStreamHandler {

  private final GraphTraversalSource g;
  private final CallExecutor<Object> executor;
  private final Random idGenerator = new Random();

  public MyHandler() {

    this.g = AnonymousTraversalSource
            .traversal()
            .withRemote(DriverRemoteConnection.using(createCluster()));


    this.executor = new CallExecutorBuilder<Object>()
            .config(createRetryConfig())
            .build();

  }

  @Override
  public void handleRequest(InputStream input,
                            OutputStream output,
                            Context context) throws IOException {

    doQuery(input, output);
  }

  private void doQuery(InputStream input, OutputStream output) throws IOException {
    try {

      Map<String, Object> args = new HashMap<>();
      args.put("id", idGenerator.nextInt());

      String result = query(args);

      try (Writer writer = new BufferedWriter(new OutputStreamWriter(output, UTF_8))) {
          writer.write(result);
      }

    } finally {
```

```
        input.close();
        output.close();
    }
  }

  private String query(Map<String, Object> args) {
    int id = (int) args.get("id");

    @SuppressWarnings("unchecked")
    Callable<Object> query = () -> g.V(id)
      .fold()
      .coalesce(
        unfold(),
        addV("Person").property(T.id, id))
      .id().next();

    Status<Object> status = executor.execute(query);

    return status.getResult().toString();
  }

  private Cluster createCluster() {
    Cluster.Builder builder = Cluster.build()

.addContactPoint(System.getenv("NEPTUNE_ENDPOINT"))

.port(Integer.parseInt(System.getenv("NEPTUNE_PORT")))
                                    .enableSsl(true)
                                    .minConnectionPoolSize(1)
                                    .maxConnectionPoolSize(1)
                                    .serializer(Serializers.GRAPHBINARY_V1D0)
                                    .reconnectInterval(2000);

  if (Boolean.parseBoolean(getOptionalEnv("USE_IAM", "true"))) {
    // For versions of TinkerPop 3.4.11 or higher:
    builder.handshakeInterceptor( r ->
      {
        NeptuneNettyHttpSigV4Signer sigV4Signer = new
NeptuneNettyHttpSigV4Signer(region, new DefaultAWSCredentialsProviderChain());
        sigV4Signer.signRequest(r);
        return r;
      }
    )
```

```
    // Versions of TinkerPop prior to 3.4.11 should use the following approach.
    // Be sure to adjust the  imports to include:
    //   import org.apache.tinkerpop.gremlin.driver.SigV4WebSocketChannelizer;
    //   builder = builder.channelizer(SigV4WebSocketChannelizer.class);

    return builder.create();
  }

  private RetryConfig createRetryConfig() {
    return new RetryConfigBuilder().retryOnCustomExceptionLogic(retryLogic())
                                   .withDelayBetweenTries(1000, ChronoUnit.MILLIS)
                                   .withMaxNumberOfTries(5)
                                   .withFixedBackoff()
                                   .build();
  }

  private Function<Exception, Boolean> retryLogic() {
    return e -> {
      StringWriter stringWriter = new StringWriter();
      e.printStackTrace(new PrintWriter(stringWriter));
      String message = stringWriter.toString();

      // Check for connection issues
      if ( message.contains("Timed out while waiting for an available host") ||
           message.contains("Timed-out waiting for connection on Host") ||
           message.contains("Connection to server is no longer active") ||
           message.contains("Connection reset by peer") ||
           message.contains("SSLEngine closed already") ||
           message.contains("Pool is shutdown") ||
           message.contains("ExtendedClosedChannelException") ||
           message.contains("Broken pipe")) {
        return true;
      }

      // Concurrent writes can sometimes trigger a ConcurrentModificationException.
      // In these circumstances you may want to backoff and retry.
      if (message.contains("ConcurrentModificationException")) {
          return true;
      }

      // If the primary fails over to a new instance, existing connections to the old
primary will
      // throw a ReadOnlyViolationException. You may want to back and retry.
      if (message.contains("ReadOnlyViolationException")) {
```

```
            return true;
        }

        return false;
      };
    }

    private String getOptionalEnv(String name, String defaultValue) {
      String value = System.getenv(name);
      if (value != null && value.length() > 0) {
        return value;
      } else {
        return defaultValue;
      }
    }
}
```

如果要在函数中加入重新连接逻辑，请参阅Java 重新连接示例。

Amazon Neptune 的 JavaScript Lambda 函数示例

有关此示例的注意事项

- JavaScript 驱动程序不维护连接池。它总是打开单个连接。
- 该示例函数使用 gremlin-aws-sigv4 中的 Sigv4 签名实用程序，对指向已启用 IAM 身份验证的数据库的请求进行签名。
- 它使用开源 async 实用程序模块中的 retry( ) 函数来处理回退和重试尝试。
- Gremlin 终端步骤会返回 JavaScript `promise`（请参阅 TinkerPop 文档）。对于 `next()`，这是一个 `{value, done}` 元组。
- 连接错误是在处理程序内部引发的，并根据此处概述的建议使用一些回退和重试逻辑进行处理，但有一个异常。有一种连接问题，驱动程序不会将其视为异常，因此这种回退和重试逻辑无法解决这个问题。

  问题在于，如果在驱动程序发送请求之后但在驱动程序收到响应之前关闭连接，则查询似乎已完成，但返回 null 值。就 lambda 函数客户端而言，该函数似乎成功完成，但响应为空。

  此问题的影响取决于您的应用程序如何处理空响应。有些应用程序可能会将读取请求中的空响应视为错误，但其它应用程序可能会错误地将其视为空结果。

遇到此连接问题的写入请求也将返回空响应。响应为空的成功调用是表示成功还是失败？ 如果调用写入函数的客户端只是将成功调用该函数视为已提交对数据库的写入，而不检查响应的正文，则系统可能会丢失数据。

此问题源于驱动程序如何处理由底层套接字发出的事件。当底层网络套接字因 ECONNRESET 错误关闭时，驱动程序使用的 WebSocket 将关闭并发出 `'ws close'` 事件。然而，驱动程序中没有任何内容可通过一种用于引发异常的方式来处理该事件。结果，查询就消失了。

为了解决此问题，此处的示例 lambda 函数添加了一个 `'ws close'` 事件处理程序，用于在创建远程连接时会向驱动程序引发异常。但是，这个异常不是沿着 Gremlin 查询的请求-响应路径引发的，因此不能用来触发 lambda 函数本身中的任何回退和重试逻辑。相反，`'ws close'` 事件处理程序引发的异常会导致未处理的异常，从而导致 lambda 调用失败。这允许调用该函数的客户端处理错误，并在适当时重试 lambda 调用。

我们建议您在 lambda 函数本身中实现回退和重试逻辑，以保护您的客户端免受间歇性连接问题的影响。但是，上述问题的解决方法也要求客户端实现重试逻辑，以处理由此特定连接问题导致的失败。

Javascript 代码

```
const gremlin = require('gremlin');
const async = require('async');
const {getUrlAndHeaders} = require('gremlin-aws-sigv4/lib/utils');

const traversal = gremlin.process.AnonymousTraversalSource.traversal;
const DriverRemoteConnection = gremlin.driver.DriverRemoteConnection;
const t = gremlin.process.t;
const __ = gremlin.process.statics;

let conn = null;
let g = null;

async function query(context) {

  const id = context.id;

  return g.V(id)
    .fold()
    .coalesce(
      __.unfold(),
      __.addV('User').property(t.id, id)
```

```
    )
    .id().next();
}

async function doQuery() {
  const id = Math.floor(Math.random() * 10000).toString();

  let result = await query({id: id});
  return result['value'];
}

exports.handler = async (event, context) => {

  const getConnectionDetails = () => {
    if (process.env['USE_IAM'] == 'true'){
       return getUrlAndHeaders(
         process.env['NEPTUNE_ENDPOINT'],
         process.env['NEPTUNE_PORT'],
         {},
         '/gremlin',
         'wss');
    } else {
      const database_url = 'wss://' + process.env['NEPTUNE_ENDPOINT'] + ':' +
 process.env['NEPTUNE_PORT'] + '/gremlin';
      return { url: database_url, headers: {}};
    }
  };


  const createRemoteConnection = () => {
    const { url, headers } = getConnectionDetails();

    const c = new DriverRemoteConnection(
      url,
      {
        mimeType: 'application/vnd.gremlin-v2.0+json',
        headers: headers
      });

     c._client._connection.on('close', (code, message) => {
         console.info(`close - ${code} ${message}`);
         if (code == 1006){
           console.error('Connection closed prematurely');
           throw new Error('Connection closed prematurely');
```

```
        }
      });

    return c;
};

const createGraphTraversalSource = (conn) => {
  return traversal().withRemote(conn);
};

if (conn == null){
  console.info("Initializing connection")
  conn = createRemoteConnection();
  g = createGraphTraversalSource(conn);
}

return async.retry(
  {
    times: 5,
    interval: 1000,
    errorFilter: function (err) {

      // Add filters here to determine whether error can be retried
      console.warn('Determining whether retriable error: ' + err.message);

      // Check for connection issues
      if (err.message.startsWith('WebSocket is not open')){
        console.warn('Reopening connection');
        conn.close();
        conn = createRemoteConnection();
        g = createGraphTraversalSource(conn);
        return true;
      }

      // Check for ConcurrentModificationException
      if (err.message.includes('ConcurrentModificationException')){
        console.warn('Retrying query because of ConcurrentModificationException');
        return true;
      }

      // Check for ReadOnlyViolationException
      if (err.message.includes('ReadOnlyViolationException')){
        console.warn('Retrying query because of ReadOnlyViolationException');
        return true;
```

```
        }

        return false;
      }

    },
    doQuery);
};
```

Amazon Neptune 的 Python Lambda 函数示例

以下是关于以下 Python AWS Lambda 示例函数的一些注意事项：

- 它使用回退模块。
- 它设置 `pool_size=1` 以防止创建不必要的连接池。
- 它设置 `message_serializer=serializer.GraphSONSerializersV2d0()`。

```
import os, sys, backoff, math
from random import randint
from gremlin_python import statics
from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
from gremlin_python.driver.protocol import GremlinServerError
from gremlin_python.driver import serializer
from gremlin_python.process.anonymous_traversal import traversal
from gremlin_python.process.graph_traversal import __
from gremlin_python.process.strategies import *
from gremlin_python.process.traversal import T
from aiohttp.client_exceptions import ClientConnectorError
from botocore.auth import SigV4Auth
from botocore.awsrequest import AWSRequest
from botocore.credentials import ReadOnlyCredentials
from types import SimpleNamespace

import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)


reconnectable_err_msgs = [
    'ReadOnlyViolationException',
    'Server disconnected',
```

```
    'Connection refused',
    'Connection was already closed',
    'Connection was closed by server',
    'Failed to connect to server: HTTP Error code 403 - Forbidden'
]

retriable_err_msgs = ['ConcurrentModificationException'] + reconnectable_err_msgs

network_errors = [OSError, ClientConnectorError]

retriable_errors = [GremlinServerError, RuntimeError, Exception] + network_errors

def prepare_iamdb_request(database_url):

    service = 'neptune-db'
    method = 'GET'

    access_key = os.environ['AWS_ACCESS_KEY_ID']
    secret_key = os.environ['AWS_SECRET_ACCESS_KEY']
    region = os.environ['AWS_REGION']
    session_token = os.environ['AWS_SESSION_TOKEN']

    creds = SimpleNamespace(
        access_key=access_key, secret_key=secret_key, token=session_token,
 region=region,
    )

    request = AWSRequest(method=method, url=database_url, data=None)
    SigV4Auth(creds, service, region).add_auth(request)

    return (database_url, request.headers.items())

def is_retriable_error(e):

    is_retriable = False
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_retriable = True
    else:
        is_retriable = any(retriable_err_msg in err_msg for retriable_err_msg in
 retriable_err_msgs)

    logger.error('error: [{}] {}'.format(type(e), err_msg))
```

```
    logger.info('is_retriable: {}'.format(is_retriable))

    return is_retriable

def is_non_retriable_error(e):
    return not is_retriable_error(e)

def reset_connection_if_connection_issue(params):

    is_reconnectable = False

    e = sys.exc_info()[1]
    err_msg = str(e)

    if isinstance(e, tuple(network_errors)):
        is_reconnectable = True
    else:
        is_reconnectable = any(reconnectable_err_msg in err_msg for
 reconnectable_err_msg in reconnectable_err_msgs)

    logger.info('is_reconnectable: {}'.format(is_reconnectable))

    if is_reconnectable:
        global conn
        global g
        conn.close()
        conn = create_remote_connection()
        g = create_graph_traversal_source(conn)

@backoff.on_exception(backoff.constant,
    tuple(retriable_errors),
    max_tries=5,
    jitter=None,
    giveup=is_non_retriable_error,
    on_backoff=reset_connection_if_connection_issue,
    interval=1)
def query(**kwargs):

    id = kwargs['id']

    return (g.V(id)
        .fold()
        .coalesce(
            __.unfold(),
```

```
            __.addV('User').property(T.id, id)
        )
        .id().next())

def doQuery(event):
    return query(id=str(randint(0, 10000)))

def lambda_handler(event, context):
    result = doQuery(event)
    logger.info('result - {}'.format(result))
    return result

def create_graph_traversal_source(conn):
    return traversal().withRemote(conn)

def create_remote_connection():
    logger.info('Creating remote connection')

    (database_url, headers) = connection_info()

    return DriverRemoteConnection(
        database_url,
        'g',
        pool_size=1,
        message_serializer=serializer.GraphSONSerializersV2d0(),
        headers=headers)

def connection_info():

    database_url = 'wss://{}:{}/gremlin'.format(os.environ['neptuneEndpoint'],
 os.environ['neptunePort'])

    if 'USE_IAM' in os.environ and os.environ['USE_IAM'] == 'true':
        return prepare_iamdb_request(database_url)
    else:
        return (database_url, {})

conn = create_remote_connection()
g = create_graph_traversal_source(conn)
```

以下是示例结果，显示了重负载和轻负载的交替时期：


GremlinWebSocketOpenConnections
Count
261
131
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
neptunedbinstance-l22qsahgvgvl
replica-python-lambdas-neptunebasestack-5rzhq9jvp2ri
GremlinRequestsPreSecond
Count/Second
861
430
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
neptunedbinstance-l22qsahgvgvl
replica-python-lambdas-neptunebasestack-5rzhq9jvp2ri
Lambda-ConcurrentExecutions
Count
86
43.5
1
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship social-b6501310_add-follows-relationship
social-b6501310_create-user social-b6501310_create-user
social-b6501310_get-recent-follows social-b6501310_get-recent-follows
social-b6501310_get-user-details social-b6501310_get-user-details
Lambda-P99-Latency
Milliseconds
2.10k
1.05k
5.46
06:05 06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship
social-b6501310_create-user
social-b6501310_get-recent-follows
social-b6501310_get-user-details
social-b6501310_recommend-new-follows
Workload
Various units
Count
2.17k
1.09k
9e-3
147k
73.7k
2
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
P99Latency
AverageThroughput
Succeeded
Errors
Count
1
0.5
0
06:10 06:15 06:20 06:25 06:30 06:35 06:40 06:45 06:50 06:55 07:00 07:05
social-b6501310_add-follows-relationship
social-b6501310_create-user
social-b6501310_get-recent-follows
social-b6501310_get-user-details
social-b6501310_recommend-new-follows

用于在图形上进行机器学习的 Amazon Neptune ML

大型互联数据集中通常存在有价值的信息，仅使用基于人类直觉的查询很难提取这些信息。机器学习 (ML) 技术可以帮助在具有数十亿个关系的图形中找到隐藏的相关性。这些相关性可能有助于推荐产品、预测信用价值、识别欺诈行为以及许多其它事情。

Neptune ML 特征可以在几小时而不是几周内在大型图形上构建和训练有用的机器学习模型。为了实现这一目标，Neptune ML 使用由 Amazon SageMaker 和深度图表库 (DGL)（开源）提供支持的图形神经网络 (GNN) 技术。图形神经网络是人工智能中的一个新兴领域（例如，请参阅图形神经网络综合调查）。有关在 DGL 中使用 GNN 的实操教程，请参阅使用深度图表库学习图形神经网络。

> **Note**
>
> 在 Neptune ML 模型中，图形顶点标识为“节点”。例如，顶点分类使用节点分类机器学习模型，而顶点回归使用节点回归模型。

Neptune ML 能做什么

Neptune 既支持转导推理，也支持归纳推理，前者返回训练时根据当时的图形数据预先计算的预测，后者根据当前数据实时返回应用的数据处理和模型评估。请参阅归纳推理和转导推理的区别。

Neptune ML 可以训练机器学习模型以支持五种不同的推理类别：

Neptune ML 目前支持的推理任务类型

- 节点分类 – 预测顶点属性的分类特征。

  例如，给定电影《肖申克的救赎》，Neptune ML 可以根据候选 `[story, crime, action, fantasy, drama, family, ...]` 集，将其 `genre` 属性预测为 `story`。

  有两种类型的节点分类任务：

  - 单类分类：在这种任务中，每个节点只有一个目标特征。例如，`Alan Turing` 的属性 `Place_of_birth` 的值为 `UK`。
  - 多类分类：在这种任务中，每个节点可以有多个目标特征。例如，电影《教父》的属性 `genre` 具有值 `crime` 和 `story`。
- 节点回归 – 预测顶点的数值属性。

例如，给定电影《复仇者联盟：残局》，Neptune ML 可以预测其 `popularity` 属性的值为 `5.0`。

- 边缘分类 – 预测边缘属性的分类特征。

  有两种类型的边缘分类任务：

  - 单类分类：在这种任务中，每个边缘只有一个目标特征。例如，用户和电影之间的评分边缘可能具有属性 `liked`，其值为“是”或“否”。
  - 多类分类：在这种任务中，每个边缘可以有多个目标特征。例如，用户和电影之间的评分可能有多个属性标签值，例如“Funny”、“Heartwarming”、“Chilling”等。
- 边缘回归 – 预测边缘的数值属性。

  例如，用户和电影之间的评分边缘可能具有数值属性 `score`，Neptune ML 可以针对给定的用户和电影预测其值。
- 链接预测 – 预测特定源节点和传出边缘最有可能的目标节点，或者预测给定目标节点和传入边缘最有可能的源节点。

  例如，对于药物疾病知识图谱（以给定 `Aspirin` 作为源节点和 `treats` 作为传出边缘），Neptune ML 可以将最相关的目标节点预测为 `heart disease`、`fever` 等。

  或者，对于维基媒体知识图谱（以 `President-of` 作为边缘或关系，并以 `United-States` 作为目标节点），Neptune ML 可以将最相关的领导人预测为 `George Washington`、`Abraham Lincoln`、`Franklin D. Roosevelt` 等。

> **Note**
>
> 节点分类和边缘分类仅支持字符串值。这意味着不支持诸如 `0` 或 `1` 之类的数值属性值，但支持等同的字符串 `"0"` 和 `"1"`。同样，布尔属性值 `true` 和 `false` 不起作用，但 `"true"` 和 `"false"` 起作用。

借助 Neptune ML，您可以使用分为两大类的机器学习模型：

Neptune ML 目前支持的机器学习模型类型

- 图形神经网络 (GNN) 模型 – 这些模型包括关系图卷积网络 (R-GCN)。GNN 模型适用于上述所有三种类型的任务。

- 知识图谱嵌入 (KGE) 模型 – 这些模型包括 `TransE`、`DistMult` 和 `RotatE` 模型。它们仅适用于链接预测。

用户定义的模型 – Neptune ML 还允许您为上面列出的所有类型的任务提供自己的自定义模型实现。在将 Neptune ML 训练 API 用于模型之前，您可以使用 Neptune ML 工具包来开发和测试基于 python 的自定义模型实现。有关如何构造和组织实现以使其与 Neptune ML 的训练基础设施兼容的详细信息，请参阅Neptune ML 中的自定义模型。

设置 Neptune ML

开始使用 Neptune ML 的最简单方式是使用 AWS CloudFormation 快速入门模板。此模板安装了所有必需的组件，包括新的 Neptune 数据库集群、所有必要的 IAM 角色和新的 Neptune 图形笔记本，以便更轻松地使用 Neptune ML。

您也可以手动安装 Neptune ML，如在不使用快速入门 AWS CloudFormation 模板的情况下设置 Neptune ML中所述。

使用 Neptune ML AWS CloudFormation 模板在新的数据库集群中快速入门

开始使用 Neptune ML 的最简单方式是使用 AWS CloudFormation 快速入门模板。此模板安装了所有必需的组件，包括新的 Neptune 数据库集群、所有必要的 IAM 角色和新的 Neptune 图形笔记本，以便更轻松地使用 Neptune ML。

创建 Neptune ML 快速入门堆栈

1. 要在 AWS CloudFormation 控制台中启动 AWS CloudFormation 堆栈，请选择下表中的启动堆栈按钮之一：

区域	查看	在 Designer 中查看	发布
美国东部（弗吉尼亚州北部）	视图	在 Designer 中查看	Launch Stack
美国东部（俄亥俄州）	视图	在 Designer 中查看	Launch Stack
美国西部（北加利福尼亚）	视图	在 Designer 中查看	Launch Stack
美国西部（俄勒冈州）	视图	在 Designer 中查看	Launch Stack
加拿大（中部）	视图	在 Designer 中查看	Launch Stack
南美洲（圣保罗）	视图	在 Designer 中查看	Launch Stack
欧洲地区（斯德哥尔摩）	视图	在 Designer 中查看	Launch Stack
欧洲地区（爱尔兰）	视图	在 Designer 中查看	Launch Stack
欧洲地区（伦敦）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
欧洲地区（巴黎）	视图	在 Designer 中查看	Launch Stack
欧洲地区（法兰克福）	视图	在 Designer 中查看	Launch Stack
中东（巴林）	视图	在 Designer 中查看	Launch Stack
中东（阿联酋）	视图	在 Designer 中查看	Launch Stack
以色列（特拉维夫）	视图	在 Designer 中查看	Launch Stack
非洲（开普敦）	视图	在 Designer 中查看	Launch Stack
亚太地区（香港）	视图	在 Designer 中查看	Launch Stack
亚太地区（东京）	视图	在 Designer 中查看	Launch Stack
亚太地区（首尔）	视图	在 Designer 中查看	Launch Stack
亚太地区（新加坡）	视图	在 Designer 中查看	Launch Stack
亚太地区（悉尼）	视图	在 Designer 中查看	Launch Stack
亚太地区（孟买）	视图	在 Designer 中查看	Launch Stack
中国（北京）	视图	在 Designer 中查看	Launch Stack
中国（宁夏）	视图	在 Designer 中查看	Launch Stack

区域	查看	在 Designer 中查看	发布
AWS GovCloud（美国西部）	视图	在 Designer 中查看	

2. 在 Select Template 页面上，选择 Next。
3. 在指定详细信息页面上，选择下一步。
4. 在选项页面上，选择下一步。
5. 在审核页面上，您需要勾选两个复选框：

   - 第一个复选框确认 AWS CloudFormation 可能会创建具有自定义名称的 IAM 资源。
   - 第二个复选框确认 AWS CloudFormation 可能需要新堆栈的 `CAPABILITY_AUTO_EXPAND` 功能。`CAPABILITY_AUTO_EXPAND` 显式允许 AWS CloudFormation 在创建堆栈时自动扩展宏，无需事先审核。

     客户通常从处理的模板创建更改集，以便在实际创建堆栈之前对宏所做的更改进行审核。有关更多信息，请参阅 AWS CloudFormation CreateStack API。

   然后选择创建。

快速入门模板创建和设置以下内容：

- Neptune 数据库集群。
- 所需的 IAM 角色（并附加它们）。
- 所需的 Amazon EC2 安全组。
- 所需的 SageMaker VPC 端点。
- Neptune ML 的数据库集群参数组。
- 该参数组中的所需参数。
- SageMaker 笔记本，预填充了 Neptune ML 的笔记本样本。请注意，并非每个区域都提供所有实例大小，因此，您需要确保所选的笔记本实例大小是您所在区域支持的大小。
- Neptune-Export 服务。

快速入门堆栈准备就绪后，前往模板创建的 SageMaker 笔记本并查看预先填充的示例。它们将帮助您下载示例数据集，用于体验 Neptune ML 功能。

当您使用 Neptune ML 时，它们还可以为您节省大量时间。例如，请参阅这些笔记本支持的 %neptune_ml 行魔术命令和 %%neptune_ml 单元格魔术命令。

您也可以使用以下 AWS CLI 命令来运行快速入门 AWS CloudFormation 模板：

```
aws cloudformation create-stack \
  --stack-name neptune-ml-fullstack-$(date '+%Y-%m-%d-%H-%M') \
  --template-url https://aws-neptune-customer-samples.s3.amazonaws.com/v2/
cloudformation-templates/neptune-ml-nested-stack.json \
  --parameters ParameterKey=EnableIAMAuthOnExportAPI,ParameterValue=(true if you have
 IAM auth enabled, or false otherwise) \
               ParameterKey=Env,ParameterValue=test$(date '+%H%M')\
  --capabilities CAPABILITY_IAM \
  --region (the AWS region, like us-east-1) \
  --disable-rollback \
  --profile (optionally, a named CLI profile of yours)
```

在不使用快速入门 AWS CloudFormation 模板的情况下设置 Neptune ML

1. 从正常运行的 Neptune 数据库集群开始

如果您不使用 AWS CloudFormation 快速入门模板来设置 Neptune ML，则需要使用现有的 Neptune 数据库集群。如果您愿意，您可以使用已经拥有的一个集群，或者克隆一个您已经在使用的集群，也可以创建一个新集群（请参阅创建数据库集群）。

2. 安装 Neptune-Export 服务

如果您还没有这样做，请安装 Neptune-Export 服务，如使用 Neptune-Export 服务导出 Neptune 数据中所述。

使用以下设置向安装过程创建的 `NeptuneExportSecurityGroup` 安全组添加入站规则：

- 类型：`Custom TCP`
- 协议：`TCP`
- 端口范围：`80 - 443`
- 来源：*`#Neptune ######## ID`*）

3. 创建自定义 **`NeptuneLoadFromS3`** IAM 角色

如果您尚未这样做，请创建自定义 `NeptuneLoadFromS3` IAM 角色，如创建 IAM 角色以访问 Amazon S3中所述。

创建自定义 **`NeptuneSageMakerIAMRole`** 角色

使用 IAM 控制台通过以下策略创建自定义 `NeptuneSageMakerIAMRole`：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:CreateVpcEndpoint",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
```

```
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/*"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com"
          ]
        }
      },
      "Effect": "Allow"
    },
    {
      "Action": [
        "kms:CreateGrant",
        "kms:Decrypt",
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:*:*:key/*",
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/sagemaker/*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:AddTags",
        "sagemaker:CreateEndpoint",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateHyperParameterTuningJob",
        "sagemaker:CreateModel",
        "sagemaker:CreateProcessingJob",
        "sagemaker:CreateTrainingJob",
        "sagemaker:CreateTransformJob",
        "sagemaker:DeleteEndpoint",
        "sagemaker:DeleteEndpointConfig",
        "sagemaker:DeleteModel",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:DescribeHyperParameterTuningJob",
        "sagemaker:DescribeModel",
        "sagemaker:DescribeProcessingJob",
        "sagemaker:DescribeTrainingJob",
        "sagemaker:DescribeTransformJob",
        "sagemaker:InvokeEndpoint",
        "sagemaker:ListTags",
        "sagemaker:ListTrainingJobsForHyperParameterTuningJob",
        "sagemaker:StopHyperParameterTuningJob",
        "sagemaker:StopProcessingJob",
        "sagemaker:StopTrainingJob",
        "sagemaker:StopTransformJob",
        "sagemaker:UpdateEndpoint",
        "sagemaker:UpdateEndpointWeightsAndCapacities"
      ],
```

```
      "Resource": [
        "arn:aws:sagemaker:*:*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:ListEndpointConfigs",
        "sagemaker:ListEndpoints",
        "sagemaker:ListHyperParameterTuningJobs",
        "sagemaker:ListModels",
        "sagemaker:ListProcessingJobs",
        "sagemaker:ListTrainingJobs",
        "sagemaker:ListTransformJobs"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

创建此角色时，请编辑信任关系，使其内容如下所示：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
```

```
          "ec2.amazonaws.com",
          "rds.amazonaws.com",
          "sagemaker.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

最后，复制分配给这个新 NeptuneSageMakerIAMRole 角色的 ARN。

> **Important**
>
> - 请确保 NeptuneSageMakerIAMRole 中的 Amazon S3 权限与上述权限一致。
> - 通用 ARN arn:aws:s3:::* 用于上述策略中的 Amazon S3 资源。如果由于某种原因无法使用通用 ARN，则必须将 arn:aws:s3:::graphlytics* 和 Neptune ML 命令将使用的任何其他客户 Amazon S3 资源的 ARN 添加到资源部分。

配置数据库集群以启用 Neptune ML

要为 Neptune ML 设置数据库集群

1. 在 Neptune 控制台中，导航到参数组，然后导航到与要使用的数据库集群关联的数据库集群参数组。将 neptune_ml_iam_role 参数设置为分配给您刚刚创建的 NeptuneSageMakerIAMRole 角色的 ARN。
2. 导航到数据库，然后选择要用于 Neptune ML 的数据库集群。选择操作，然后选择管理 IAM 角色。
3. 在管理 IAM 角色页面上，选择添加角色并添加 NeptuneSageMakerIAMRole。然后添加 NeptuneLoadFromS3 角色。
4. 重启数据库集群的写入器实例。

在您的 Neptune VPC 中创建两个 SageMaker 端点

最后，要让 Neptune 引擎访问必要的 SageMaker 管理 API，您需要在 Neptune VPC 中创建两个 SageMaker 端点，如在 Neptune VPC 中为 SageMaker 创建两个端点中所述。

手动为 Neptune ML 配置 Neptune 笔记本

Neptune SageMaker 笔记本预装了各种适用于 Neptune ML 的示例笔记本。您可以在开源图形笔记本 GitHub 存储库中预览这些示例。

您可以使用现有的 Neptune 笔记本之一，或者如果您愿意，也可以按照使用 Neptune Workbench 托管 Neptune 笔记本中的说明创建自己的笔记本。

您也可以按照以下步骤配置默认 Neptune 笔记本以与 Neptune ML 结合使用：

修改 Neptune ML 的笔记本

1. 通过 https://console.aws.amazon.com/sagemaker/ 打开 Amazon SageMaker 控制台。
2. 在左侧的导航窗格上，选择笔记本，然后选择笔记本实例。查找要用于 Neptune ML 的 Neptune 笔记本的名称，然后将其选中以进入其详细信息页面。
3. 如果笔记本实例正在运行，请选择笔记本详细信息页面右上角的停止按钮。
4. 在笔记本实例设置中，在生命周期配置下，选择指向打开笔记本生命周期的页面的链接。
5. 选择右上角的编辑，然后选择继续。
6. 在启动笔记本选项卡中，修改脚本以包含其它导出命令，并填写 Neptune ML IAM 角色和导出服务 URI 的字段，具体取决于您的 Shell：

```
echo "export NEPTUNE_ML_ROLE_ARN=(your Neptune ML IAM role ARN)" >> ~/.bashrc
echo "export NEPTUNE_EXPORT_API_URI=(your export service URI)" >> ~/.bashrc
```

7. 选择更新。
8. 返回笔记本实例页面。在权限和加密下，有一个对应于 IAM 角色 ARN 的字段。选择此字段中的链接可转到运行此笔记本实例的 IAM 角色。
9. 创建一个新的内联策略，如下所示：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "arn:aws:cloudwatch:[AWS_REGION]:[AWS_ACCOUNT_ID]:*",
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "arn:aws:logs:[AWS_REGION]:[AWS_ACCOUNT_ID]:*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:Put*",
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "arn:aws:s3:::*",
      "Effect": "Allow"
    },
    {
      "Action": "execute-api:Invoke",
      "Resource": "arn:aws:execute-api:[AWS_REGION]:[AWS_ACCOUNT_ID]:*/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "sagemaker:CreateModel",
        "sagemaker:CreateEndpointConfig",
        "sagemaker:CreateEndpoint",
        "sagemaker:DescribeModel",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:DescribeEndpoint",
        "sagemaker:DeleteModel",
        "sagemaker:DeleteEndpointConfig",
        "sagemaker:DeleteEndpoint"
      ],
      "Resource": "arn:aws:sagemaker:[AWS_REGION]:[AWS_ACCOUNT_ID]:*/*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "iam:PassRole"
      ],
```

```
      "Resource": "[YOUR_NEPTUNE_ML_IAM_ROLE_ARN]"",
      "Effect": "Allow"
    }
  ]
}
```

10. 保存此新策略，并将其附加到步骤 8 中的 IAM 角色。
11. 选择 SageMaker 笔记本实例详细信息页面右上角的启动以启动笔记本实例。

使用 AWS CLI 在数据库集群上设置 Neptune ML

除了 AWS CloudFormation 快速入门模板和 AWS Management Console之外，还可以使用 AWS CLI 设置 Neptune ML。

1. 为 Neptune ML 集群创建数据库集群参数组

以下 AWS CLI 命令创建新的数据库集群参数组并将其设置为与 Neptune ML 结合使用：

为 Neptune ML 创建和配置数据库集群参数组

1. 创建新的数据库集群参数组：

```
aws neptune create-db-cluster-parameter-group \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --db-parameter-group-family neptune1
  --description "(description of your machine learning project)" \
  --region (AWS region, such as us-east-1)
```

2. 创建一个设置为 SageMakerExcecutionIAMRole 的 ARN 的 neptune_ml_iam_role 数据库集群参数，供您的数据库集群在调用 SageMaker 以创建任务和从托管 ML 模型获取预测时使用：

```
aws neptune modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --parameters "ParameterName=neptune_ml_iam_role, \
                ParameterValue=ARN of the SageMakerExcecutionIAMRole, \
                Description=NeptuneMLRole, \
                ApplyMethod=pending-reboot" \
  --region (AWS region, such as us-east-1)
```

   设置此参数允许 Neptune 访问 SageMaker，而无需在每次调用时都传递角色。

   有关如何创建 SageMakerExcecutionIAMRole 的信息，请参阅创建自定义 NeptuneSageMakerIAMRole 角色。

3. 最后，使用 describe-db-cluster-parameters 检查新数据库集群参数组中的所有参数是否都设置为您希望的值：

```
aws neptune describe-db-cluster-parameters \
  --db-cluster-parameter-group-name (name of the new DB cluster parameter group) \
  --region (AWS region, such as us-east-1)
```

将新的数据库集群参数组附加到将与 Neptune ML 一起使用的数据库集群

现在，您可以使用以下命令，将刚刚创建的新数据库集群参数组附加到现有数据库集群：

```
aws neptune modify-db-cluster \
  --db-cluster-identifier (the name of your existing DB cluster) \
  --apply-immediately
  --db-cluster-parameter-group-name (name of your new DB cluster parameter group) \
  --region (AWS region, such as us-east-1)
```

要使所有参数生效，您可以重启数据库集群：

```
aws neptune reboot-db-instance
  --db-instance-identifier (name of the primary instance of your DB cluster) \
  --profile (name of your AWS profile to use) \
  --region (AWS region, such as us-east-1)
```

或者，如果您要创建与 Neptune ML 结合使用的新数据库集群，则可以使用以下命令创建附加了新参数组的集群，然后创建新的主（写入器）实例：

```
cluster-name=(the name of the new DB cluster)
aws neptune create-db-cluster
  --db-cluster-identifier ${cluster-name}
  --engine graphdb \
  --engine-version 1.0.4.1 \
  --db-cluster-parameter-group-name (name of your new DB cluster parameter group) \
  --db-subnet-group-name (name of the subnet to use) \
  --region (AWS region, such as us-east-1)

aws neptune create-db-instance
  --db-cluster-identifier ${cluster-name}
  --db-instance-identifier ${cluster-name}-i \
  --db-instance-class (the instance class to use, such as db.r5.xlarge)
  --engine graphdb \
  --region (AWS region, such as us-east-1)
```

将 NeptuneSageMakerIAMRole 连接到您的数据库集群，以便它可以访问 SageMaker 和 Amazon S3 资源

最后，按照创建自定义 NeptuneSageMakerIAMRole 角色中的说明创建一个 IAM 角色，该角色将允许您的数据库集群与 SageMaker 和 Amazon S3 通信。然后，使用以下命令将您创建的 NeptuneSageMakerIAMRole 角色附加到数据库集群：

```
aws neptune add-role-to-db-cluster
  --db-cluster-identifier ${cluster-name}
  --role-arn arn:aws:iam::(the ARN number of the role's ARN):role/NeptuneMLRole \
  --region (AWS region, such as us-east-1)
```

在 Neptune VPC 中为 SageMaker 创建两个端点

Neptune ML 需要在 Neptune 数据库集群的 VPC 中使用两个 SageMaker 端点：

- com.amazonaws.*(AWS region, like us-east-1)*.sagemaker.runtime
- com.amazonaws.*(AWS region, like us-east-1)*.sagemaker.api

如果您尚未使用快速入门 AWS CloudFormation 模板（它会自动为您创建这些端点），则可以使用以下 AWS CLI 命令来创建它们：

此命令创建 sagemaker.runtime 端点：

```
create-vpc-endpoint
  --vpc-id (the ID of your Neptune DB cluster's VPC)
  --service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.runtime
  --subnet-ids (the subnet ID or IDs that you want to use)
  --security-group-ids (the security group for the endpoint network interface, or omit
 to use the default)
  --private-dns-enabled
```

而此命令创建 sagemaker.api 端点：

```
aws create-vpc-endpoint
  --vpc-id (the ID of your Neptune DB cluster's VPC)
  --service-name com.amazonaws.(AWS region, like us-east-1).sagemaker.api
  --subnet-ids (the subnet ID or IDs that you want to use)
  --security-group-ids (the security group for the endpoint network interface, or omit
 to use the default)
```

```
  --private-dns-enabled
```

您也可以使用 VPC 控制台创建这些端点。请参阅使用 AWS PrivateLink 在 Amazon SageMaker 中保护预测调用和使用 AWS PrivateLink 保护所有 Amazon SageMaker API 调用。

在数据库集群参数组中创建 SageMaker 推理端点参数

为了避免在对您正在使用的模型的 SageMaker 推理端点进行的每次查询中都必须指定该端点，请在 Neptune ML 的数据库集群参数组中创建一个名为 `neptune_ml_endpoint` 的数据库集群参数。将此参数设置为相关实例端点的 `id`。

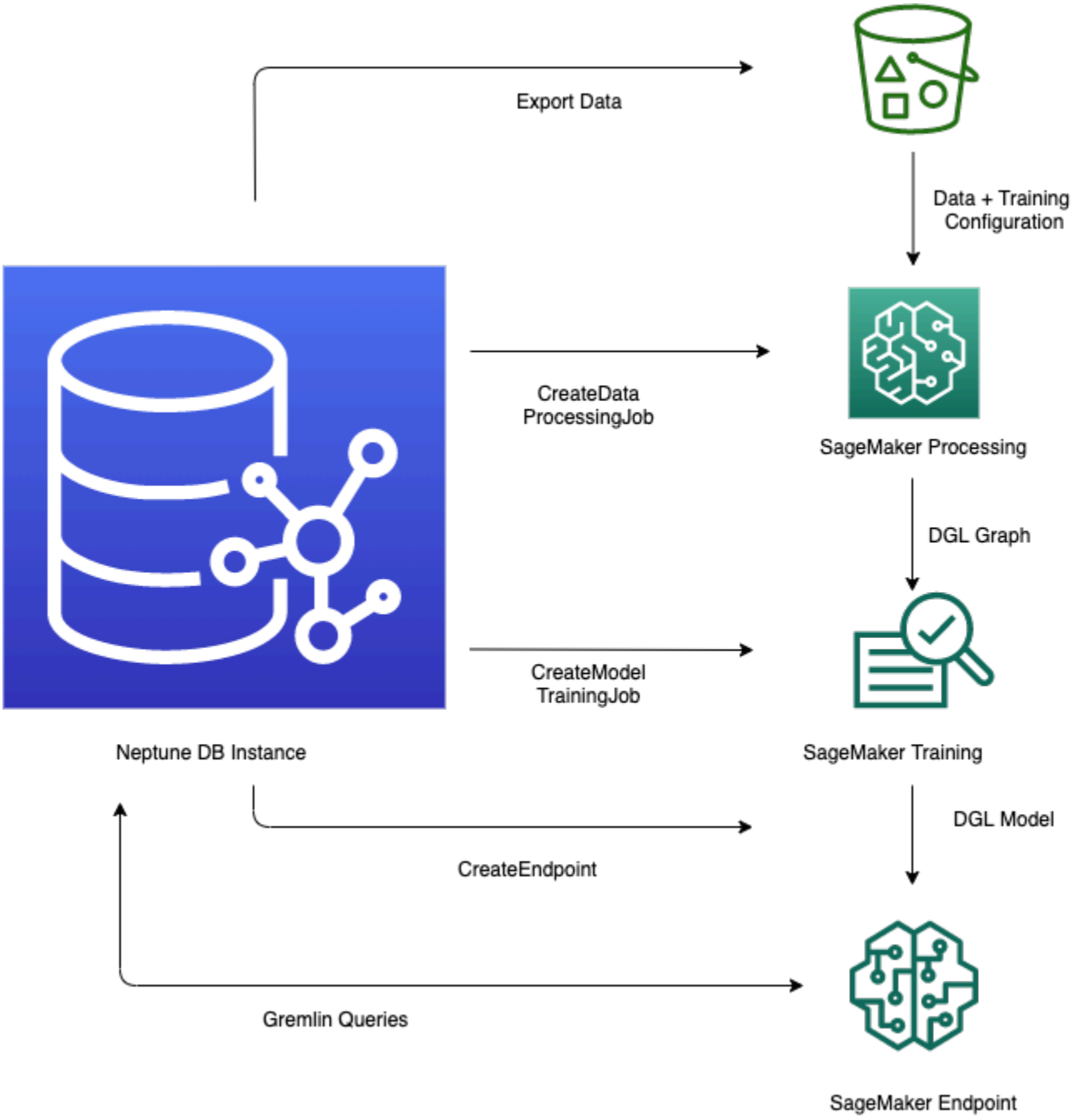
您可以使用以下 AWS CLI 命令来执行此操作：

```
aws neptune modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name neptune-ml-demo \
  --parameters "ParameterName=neptune_ml_endpoint, \
                ParameterValue=(the name of the SageMaker inference endpoint you want
 to query), \
                Description=NeptuneMLEndpoint, \
                ApplyMethod=pending-reboot" \
  --region (AWS region, such as us-east-1)
```

关于如何使用 Neptune ML 特征的概述

开始使用 Neptune ML 的工作流程

在 Amazon Neptune 中使用 Neptune ML 特征通常需要从以下五个步骤开始：



1. 数据导出和配置 – 数据导出步骤使用 Neptune-Export 服务或 `neptune-export` 命令行工具，将数据从 Neptune 以 CSV 格式导出到 Amazon Simple Storage Service (Amazon S3)。同时会自动生成

一个名为 `training-data-configuration.json` 的配置文件，该文件指定如何将导出的数据加载到可训练的图形中。

2. 数据预处理 - 在此步骤中，使用标准技术对导出的数据集进行预处理，准备好数据以进行模型训练。可以对数字数据执行特征标准化，也可以使用 `word2vec` 对文本特征进行编码。在此步骤结束时，将根据导出的数据集生成 DGL（深度图表库）图形供模型训练步骤使用。

   此步骤是使用您账户中的 SageMaker 处理任务来实现的，生成的数据存储在您指定的 Amazon S3 位置。

3. 模型训练 - 模型训练步骤训练将用于预测的机器学习模型。

   模型训练分两个阶段完成：

   - 第一阶段使用 SageMaker 处理任务来生成模型训练策略配置集，该配置集指定模型训练将使用哪种类型的模型和模型超参数范围。
   - 然后，第二阶段使用 SageMaker 模型调整任务来尝试不同的超参数配置，并选择生成了性能最佳模型的训练任务。调整任务对处理后的数据运行预先指定数量的模型训练任务试验。在此阶段结束时，将使用最佳训练任务的训练模型参数来生成模型构件以进行推理。

4. 在 Amazon SageMaker 中创建推理端点 – 推理端点是一个 SageMaker 端点实例，它使用最佳训练任务生成的模型构件启动。每个模型都绑定到单个端点。端点能够接受来自图形数据库的传入请求，并返回请求中输入的模型预测。创建端点后，它会一直处于活动状态，直到您将其删除。

5. 使用 Gremlin 查询机器学习模型 – 您可以使用 Gremlin 查询语言的扩展来查询来自推理端点的预测。

> **Note**
>
> Neptune Workbench 包含行魔术命令和单元格魔术命令，它们可以为您节省大量管理这些步骤的时间，即：
>
> - %neptune_ml
> - %%neptune_ml

根据不断变化的图形数据进行预测

对于不断变化的图形，您可能需要定期使用新数据创建新的批量预测。查询预先计算的预测（转导推理）可能比基于最新数据动态生成新预测（归纳推理）快得多。这两种方法都有其用武之地，具体取决于数据变化的速度和性能要求。

归纳推理和转导推理的区别

执行转导推理时，Neptune 会查找并返回训练时预先计算的预测。

执行归纳推理时，Neptune 会构造相关的子图形并提取其属性。然后，DGL GNN 模型实时应用数据处理和模型评估。

因此，归纳推理可以生成涉及在训练时不存在且反映图形当前状态的节点和边缘的预测。但是，这是以更高的延迟为代价的。

如果您的图形是动态的，则可能需要使用归纳推理来确保将最新数据考虑在内，但是如果您的图形是静态的，则转导推理更快、更高效。

默认情况下，归纳推理处于禁用状态。您可以通过在查询中使用 Gremlin Neptune#ml.inductiveInference 谓词为查询启用它，如下所示：

```
.with( "Neptune#ml.inductiveInference")
```

增量转导工作流程

虽然您只需重新运行第一步到第三步（从数据导出和配置到模型转换）即可更新模型构件，但 Neptune ML 支持更的简单方法以使用新数据更新批量 ML 预测。一种是使用增量模型工作流程，另一种是使用通过热启动进行模型再训练。

增量模型工作流程

在此工作流程中，您无需重新训练机器学习模型即可更新 ML 预测。

> **Note**
>
> 只有在使用新节点和/或边缘更新图形数据后，才能执行此操作。当节点被移除时，它目前无法运行。

1. 数据导出和配置 - 此步骤与主工作流程中的步骤相同。
2. 增量数据预处理 - 此步骤与主工作流程中的数据预处理步骤类似，但使用的处理配置与之前使用的处理配置相同，对应于特定的经过训练的模型。
3. 模型转换 - 此模型转换步骤不是模型训练步骤，而是从主工作流程中获取经过训练的模型和增量数据预处理步骤的结果，并生成用于推理的新模型构件。模型转换步骤启动 SageMaker 处理任务，以执行生成更新后的模型构件的计算。

4. 更新 Amazon SageMaker 推理端点 –（可选）如果您具有现有的推理端点，则此步骤会使用模型转换步骤生成的新模型构件更新端点。或者，也可以使用新的模型构件创建新的推理端点。

通过热启动进行模型再训练

使用此工作流程，您可以训练和部署新的机器学习模型，以便使用增量图形数据进行预测，但要从使用主工作流程生成的现有模型开始：

1. 数据导出和配置 - 此步骤与主工作流程中的步骤相同。
2. 增量数据预处理 - 此步骤与增量模型推理工作流程中的步骤相同。新的图形数据应该使用以前用于模型训练的相同处理方法进行处理。
3. 通过热启动进行模型训练 – 模型训练与主工作流程中发生的情况类似，但您可以利用先前模型训练任务中的信息来加快模型超参数搜索的速度。
4. 更新 Amazon SageMaker 推理端点 – 此步骤与增量模型推理工作流程中的步骤相同。

Neptune ML 中自定义模型的工作流程

Neptune ML 允许您为 Neptune ML 支持的任何任务实现、训练和部署您自己的自定义模型。开发和部署自定义模型的工作流程与内置模型的工作流程基本相同，但有一些区别，如自定义模型工作流程中所述。

Neptune ML 阶段的实例选择

Neptune ML 处理的不同阶段使用不同的 SageMaker 实例。在这里，我们将讨论如何为每个阶段选择合适的实例类型。您可以在 Amazon SageMaker 定价中找到有关 SageMaker 实例类型和定价的信息。

选择进行数据处理的实例

SageMaker 数据处理步骤需要一个具有足够内存和磁盘存储空间的处理实例，用于存放输入、中间和输出数据。所需的具体内存和磁盘存储量取决于 Neptune ML 图形的特性及其导出的特征。

默认情况下，Neptune ML 会选择内存比磁盘上导出的图形数据大小大十倍的最小 `ml.r5` 实例。

为模型训练和模型转换选择实例

为模型训练或模型转换选择正确的实例类型取决于任务类型、图形大小和周转要求。GPU 实例可提供最佳性能。我们通常建议使用 p3 和 g4dn 串行实例。您也可以使用 p2 或 p4d 实例。

默认情况下，Neptune ML 选择其内存超过模型训练和模型转换所需内存的最小 GPU 实例。您可以在 Amazon S3 数据处理输出位置的 `train_instance_recommendation.json` 文件中找到该选择的内容。下面是 `train_instance_recommendation.json` 文件的内容示例：

```
{
  "instance":     "(the recommended instance type for model training and transform)",
  "cpu_instance": "(the recommended instance type for base processing instance)",
  "disk_size":    "(the estimated disk space required)",
  "mem_size":     "(the estimated memory required)"
}
```

为推理端点选择实例

为推理端点选择正确的实例类型取决于任务类型、图形大小和预算。默认情况下，Neptune ML 选择推理端点所需内存较多的最小 `ml.m5d` 实例。

> **Note**
>
> 如果需要超过 384GB 的内存，Neptune ML 将使用 `ml.r5d.24xlarge` 实例。

您可以在位于您用于模型训练的 Amazon S3 位置的 `infer_instance_recommendation.json` 文件中查看 Neptune ML 推荐的实例类型。下面是该文件内容的示例：

```
{
  "instance" :   "(the recommended instance type for an inference endpoint)",
  "disk_size" :  "(the estimated disk space required)",
  "mem_size" :   "(the estimated memory required)"
}
```

使用 `neptune-export` 工具或 Neptune-Export 服务从 Neptune 导出数据用于 Neptune ML

Neptune ML 要求您为深度图表库 (DGL) 提供训练数据，以创建和评估模型。

您可以使用 Neptune-Export 服务或 `neptune-export` 实用程序从 Neptune 导出数据。该服务和命令行工具均以 CSV 格式将数据发布到 Amazon Simple Storage Service (Amazon S3) 的数据，且使用 Amazon S3 服务器端加密 (SSE-S3) 进行加密。请参阅由 Neptune-Export 和 `neptune-export` 导出的文件。

此外，当您为 Neptune ML 配置训练数据的导出时，导出任务会创建并发布加密的模型训练配置文件以及导出的数据。默认情况下，此文件命名为 `training-data-configuration.json`。

使用 Neptune-Export 服务为 Neptune ML 导出训练数据的示例

此请求导出节点分类任务的属性图训练数据：

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-pg",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": {
          "endpoint": "(your Neptune endpoint DNS name)",
          "profile": "neptune_ml"
        },
        "additionalParams": {
          "neptune_ml": {
            "version": "v2.0",
            "targets": [
              {
                "node": "Movie",
                "property": "genre",
                "type": "classification"
              }
            ]
          }
        }
      }'
```

此请求导出节点分类任务的 RDF 训练数据：

```
curl \
  (your NeptuneExportApiUri) \
  -X POST \
  -H 'Content-Type: application/json' \
  -d '{
        "command": "export-rdf",
        "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
        "params": {
          "endpoint": "(your Neptune endpoint DNS name)",
          "profile": "neptune_ml"
        },
        "additionalParams": {
          "neptune_ml": {
            "version": "v2.0",
            "targets": [
              {
                "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
                "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/
genre",
                "type": "classification"
              }
            ]
          }
        }
      }'
```

导出训练数据时要在 **params** 对象中设置的字段

如 params 文档中所述，导出请求中的 params 对象可以包含各种字段。以下字段与导出机器学习训练数据最为相关：

- **endpoint** – 使用 endpoint 指定数据库集群中 Neptune 实例的端点，导出过程可以查询该端点以提取数据。
- **profile** – params 对象中的 profile 字段必须设置为 **neptune-ml**。

  这导致导出过程针对 Neptune ML 模型训练适当地格式化导出的数据，针对属性图数据格式化为 CSV 格式，或针对 RDF 数据格式化为 N-Triples。它还会创建 training-data-configuration.json 文件，并将其写入与导出的训练数据相同的 Amazon S3 位置。

- **cloneCluster** – 如果设置为 `true`，则导出过程将克隆您的数据库集群，从克隆中导出，然后在导出完成后将克隆删除。
- **useIamAuth** – 如果您的数据库集群启用了 IAM 身份验证，则必须包含此字段（设置为 `true`）。

导出过程还提供了多种筛选您导出的数据的方法（请参阅这些示例）。

使用 additionalParams 对象调整模型训练信息的导出

`additionalParams` 对象包含的字段可用于指定用于训练目的的机器学习类别标签和特征，并指导创建训练数据配置文件。

导出过程无法自动推理哪些节点和边缘属性应作为机器学习类别标签以用于训练目的的示例。它也无法自动推理数字、分类和文本属性的最佳特征编码，因此您需要使用 `additionalParams` 对象中的字段提供提示来指定这些编码，或者覆盖默认编码。

对于属性图数据，导出请求中 `additionalParams` 的顶级结构可能如下所示：

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams": {
      "neptune_ml": {
        "version": "v2.0",
        "targets": [ (an array of node and edge class label targets) ],
        "features": [ (an array of node feature hints) ]
    }
  }
}
```

对于 RDF 数据，其顶层结构可能如下所示：

```
{
  "command": "export-rdf",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
```

```
    "profile": "neptune_ml"
  },
  "additionalParams": {
      "neptune_ml": {
        "version": "v2.0",
        "targets": [ (an array of node and edge class label targets) ]
    }
  }
}
```

您还可以使用 jobs 字段提供多种导出配置：

```
{
  "command": "export-pg",
  "outputS3Path": "s3://(your Amazon S3 bucket)/neptune-export",
  "params": {
    "endpoint": "(your Neptune endpoint DNS name)",
    "profile": "neptune_ml"
  },
  "additionalParams" : {
    "neptune_ml" : {
      "version": "v2.0",
      "jobs": [
        {
          "name" : "(training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        },
        {
          "name" : "(another training data configuration name)",
          "targets": [ (an array of node and edge class label targets) ],
          "features": [ (an array of node feature hints) ]
        }
      ]
    }
  }
}
```

additionalParams 中 neptune_ml 字段中的顶级元素

neptune_ml 中的 version 元素

指定要生成的训练数据配置的版本。

（可选），类型：字符串，默认值：“v2.0”。

如果确实包含 `version`，请将其设置为 `v2.0`。

`neptune_ml` 中的 `jobs` 字段

包含训练数据配置对象的数组，其中每个对象都定义了一个数据处理任务，并包含：

- **`name`** – 要创建的训练数据配置的名称。

  例如，名为“job-number-1”的训练数据配置会生成名为 `job-number-1.json` 的训练数据配置文件。
- **`targets`** – 由节点和边缘类别标签目标组成的 JSON 数组，代表用于训练目的的机器学习类别标签。请参阅neptune_ml 对象中的 targets 字段。
- **`features`**– 节点属性特征的 JSON 数组。请参阅neptune_ml 中的 features 字段。

neptune_ml 对象中的 targets 字段

JSON 训练数据导出配置中的 targets 字段包含一组用于指定训练任务的目标对象以及用于训练该任务的机器学习类标签。目标对象的内容会有所不同，具体取决于您是在使用属性图数据还是 RDF 数据进行训练。

对于属性图节点分类和回归任务，数组中的目标对象可以如下所示：

```
{
  "node": "(node property-graph label)",
  "property": "(property name)",
  "type" : "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

对于属性图边缘分类、回归或链接预测任务，它们可以如下所示：

```
{
  "edge": "(edge property-graph label)",
  "property": "(property name)",
  "type" : "(used to specify classification, regression or link_prediction)",
  "split_rate": [0.8,0.2,0.0],
  "separator": ","
}
```

对于 RDF 分类和回归任务，数组中的目标对象可能如下所示：

```
{
  "node": "(node type of an RDF node)",
  "predicate": "(predicate IRI)",
  "type" : "(used to specify classification or regression)",
  "split_rate": [0.8,0.2,0.0]
}
```

对于 RDF 链接预测任务，数组中的目标对象可能如下所示：

```
{
  "subject": "(source node type of an edge)",
  "predicate": "(relation type of an edge)",
```

```
  "object": "(destination node type of an edge)",
  "type" : "link_prediction",
  "split_rate": [0.8,0.2,0.0]
}
```

目标对象可以包含以下字段：

目录

属性图目标对象中的字段

目标对象中的 **node**（顶点）字段

目标节点（顶点）的属性图标签。目标对象必须包含一个 node 元素或一个 edge 元素，但不能同时包含两者。

node 可以取一个单一值，如下所示：

```
  "node": "Movie"
```

或者，对于多标签顶点，它可以采用一个值数组，如下所示：

```
  "node": ["Content", "Movie"]
```

属性图目标对象中的 **edge** 字段

通过目标边缘的起始节点标签、其自己的标签和其结束节点标签来指定目标边缘。目标对象必须包含一个 edge 元素或一个 node 元素，但不能同时包含两者。

edge 字段的值是一个由三个字符串组成的 JSON 数组，这些字符串代表起始节点的属性图标签、边缘本身的属性图标签和结束节点的属性图标签，如下所示：

```
  "edge": ["Person_A", "knows", "Person_B"]
```

如果起始节点和/或结束节点有多个标签，请将它们括在数组中，如下所示：

```
  "edge": [ ["Admin", Person_A"], "knows", ["Admin", "Person_B"] ]
```

属性图目标对象中的 **property** 字段

指定目标顶点或边缘的属性，如下所示：

```
  "property" : "rating"
```

除非目标任务是链接预测，否则此字段为必需字段。

属性图目标对象中的 **type** 字段

表示要在 node 或 edge 上执行的目标任务的类型，如下所示：

```
  "type" : "regression"
```

节点支持的任务类型有：

- `classification`
- `regression`

边缘支持的任务类型有：

- `classification`

- `regression`
- `link_prediction`

此字段为必填。

属性图目标对象中的 **`split_rate`** 字段

（可选）训练、验证和测试阶段将分别使用的节点或边缘的比例的估计值。这些比例由 JSON 数组表示，该数组包含介于零和一之间的三个数字，这些数字加起来为 1：

```
"split_rate": [0.7, 0.1, 0.2]
```

如果您未提供可选 `split_rate` 字段，则默认估计值为 `[0.9, 0.1, 0.0]`。

属性图目标对象中的 **`separator`** 字段

（可选）与分类任务一起使用。

`separator` 字段指定一个字符，当它用于在字符串中存储多个类别值时，用来将目标属性值拆分为多个类别值。例如：

```
"separator": "|"
```

`separator` 字段的存在表示该任务是一项多目标分类任务。

RDF 目标对象中的字段

RDF 目标对象中的 **`node`** 字段

定义目标节点的节点类型。与节点分类任务或节点回归任务一起使用。RDF 中节点的节点类型定义如下：

```
  node_id, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, node_type
```

RDF node 只能取一个值，如下所示：

```
  "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

RDF 目标对象中的 **subject** 字段

对于链接预测任务，subject 定义目标边缘的源节点类型。

```
  "subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director"
```

> **Note**
>
> 对于链接预测任务，subject 应与 predicate 和 object 一起使用。如果未提供这三者中的任何一个，则所有边缘都将被视为训练目标。

RDF 目标对象中的 **predicate** 字段

对于节点分类和节点回归任务，predicate 定义将哪些文本数据用作目标节点的目标节点特征。

```
  "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre"
```

> **Note**
>
> 如果目标节点只有一个定义目标节点特征的谓词，则可以省略 predicate 字段。

对于链接预测任务，predicate 定义目标边缘的关系类型：

```
"predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/direct"
```

> **Note**
>
> 对于链接预测任务，predicate 应与 subject 和 object 一起使用。如果未提供这三者中的任何一个，则所有边缘都将被视为训练目标。

RDF 目标对象中的 **object** 字段

对于链接预测任务，object 定义目标边缘的目标节点类型：

```
  "object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie"
```

> **Note**
>
> 对于链接预测任务，`object` 应与 `subject` 和 `predicate` 一起使用。如果未提供这三者中的任何一个，则所有边缘都将被视为训练目标。

RDF 目标对象中的 **`type`** 字段

表示要执行的目标任务的类型，如下所示：

```
  "type" : "regression"
```

RDF 数据支持的任务类型有：

- `link_prediction`
- `classification`
- `regression`

此字段为必填。

属性图目标对象中的 **`split_rate`** 字段

（可选）训练、验证和测试阶段将分别使用的节点或边缘的比例的估计值。这些比例由 JSON 数组表示，该数组包含介于零和一之间的三个数字，这些数字加起来为 1：

```
"split_rate": [0.7, 0.1, 0.2]
```

如果您未提供可选 `split_rate` 字段，则默认估计值为 `[0.9, 0.1, 0.0]`。

neptune_ml 中的 features 字段

属性值和 RDF 文本值有不同的格式和数据类型。为了在机器学习中获得良好的性能，务必将这些值转换为称为特征的数字编码。

Neptune ML 在数据导出和数据处理步骤中执行特征提取和编码，如Neptune ML 中的特征编码中所述。

对于属性图数据集，导出过程会自动推理字符串属性和包含多个值的数值属性的 auto 特征。对于包含单个值的数值属性，它会推理 numerical 特征。对于日期属性，它会推理 datetime 特征。

如果要覆盖自动推理的特征规范，或者要为属性添加桶数值、TF-IDF、FastText 或 SBERT 规范，则可以使用特征字段控制特征编码。

> **Note**
>
> 您只能使用 features 字段来控制属性图数据的特征规范，而不能用于控制 RDF 数据的特征规范。

对于自由格式文本，Neptune ML 可以使用几种不同的模型将字符串属性值中的令牌序列转换为固定大小的实值向量：

- text_fasttext– 使用 fastText 编码。对于使用 FastText 支持的五种语言中的一种且仅一种的特征，推荐使用这种编码。
- text_sbert– 使用句子 BERT (SBERT) 编码模型。对于 text_fasttext 不支持的文本，建议使用这种编码。
- text_word2vec– 使用最初由 Google 发布的 Word2Vec 算法对文本进行编码。Word2Vec 仅支持英语。
- text_tfidf– 使用术语频率 - 反向文档频率 (TF-IDF) 向量器对文本进行编码。TF-IDF 编码支持其它编码不支持的统计特征。

features 字段包含节点属性特征的 JSON 数组。数组中的对象可以包含以下字段：

目录

- features 中的 property 字段
- 特征的 type 字段的可能值
- norm 字段
- language 字段
- max_length 字段
- separator 字段
- range 字段
- bucket_cnt 字段
- slide_window_size 字段
- imputer 字段
- max_features 字段
- min_df 字段
- ngram_range 字段
- datetime_parts 字段

features 中的 node 字段

node 字段指定特征顶点的属性图标签。例如：

```
  "node": "Person"
```

如果顶点有多个标签，则使用数组来包含它们。例如：

```
  "node": ["Admin", "Person"]
```

features 中的 edge 字段

edge 字段指定特征边缘的边缘类型。边缘类型由一个数组组成，该数组包含源顶点的属性图标签、边缘的属性图标签和目标顶点的属性图标签。指定边缘特征时，必须提供所有三个值。例如：

```
  "edge": ["User", "reviewed", "Movie"]
```

如果边缘类型的源顶点或目标顶点有多个标签，请使用另一个数组来包含它们。例如：

```
"edge": [["Admin", "Person"]. "edited", "Post"]
```

`features` 中的 `property` 字段

使用 property 参数指定由 node 参数标识的顶点的属性。例如：

```
"property" : "age"
```

特征的 `type` 字段的可能值

type 参数指定要定义的特征的类型。例如：

```
"type": "bucket_numerical"
```

type 参数的可能值

- **"auto"** – 指定 Neptune ML 应自动检测属性类型并应用正确的特征编码。auto 特征也可以有一个可选 separator 字段。

  请参阅Neptune ML 中的自动特征编码。
- **"category"**– 此特征编码将属性值表示为多个类别之一。换句话说，该特征可以采用一个或多个离散值。category 特征也可以有一个可选 separator 字段。

  请参阅Neptune ML 中的分类特征。
- **"numerical"**– 此特征编码将数值属性值表示为连续间隔中的数字，其中“大于”和“小于”具有含义。

  numerical 特征也可以具有可选 norm、imputer 和 separator 字段。

  请参阅Neptune ML 中的数值特征。
- **"bucket_numerical"** – 此特征编码将数字属性值划分为一组桶或类别。

  例如，您可以将人们的年龄编码为 4 个桶：儿童 (0-20)、年轻人 (20-40)、中年人 (40-60) 和老年人（60 岁及以上）。

  bucket_numerical 特征需要 range 和 bucket_cnt 字段，也可以选择包含 imputer 和/或 slide_window_size 字段。

  请参阅Neptune ML 中的桶数值特征。

- **"datetime"** – 此特征编码将日期时间属性值表示为以下分类特征的数组：年、月、工作日和小时。

  使用 datetime_parts 参数可以消除这四个类别中的一个或多个类别。

  请参阅Neptune ML 中的日期时间特征。

- **"text_fasttext"** – 此特征编码使用 fastText 模型将由句子或自由格式文本组成的属性值转换为数字向量。它支持五种语言，即英语 (en)、中文 (zh)、印地语 (hi)、西班牙语 (es) 和法语 (fr)。对于这五种语言中任意一种语言的文本属性值，推荐使用 text_fasttext 编码。但是，它无法处理同一个句子包含多种语言的字词的情况。

  对于除 fastText 支持的语言之外的其它语言，请使用 text_sbert 编码。

  如果您有许多长度超过比如 120 个令牌的属性值文本字符串，请使用 max_length 字段来限制 "text_fasttext" 编码的每个字符串中的令牌数量。

  请参阅Neptune ML 中文本属性值的 fastText 编码。

- **"text_sbert"** – 此编码使用句子 BERT (SBERT) 模型将文本属性值转换为数字向量。Neptune 支持两种 SBERT 方法，即 text_sbert128（如果您只指定 text_sbert，则这是默认方法）和 text_sbert512。它们之间的区别在于文本属性中编码的最大令牌数。text_sbert128 编码仅对前 128 个令牌进行编码，而 text_sbert512 最多可对 512 个令牌进行编码。因此，使用 text_sbert512 所需的处理时间可能比 text_sbert128 更长。两种方法都慢于 text_fasttext。

  这些 text_sbert* 方法支持多种语言，并且可以对包含多种语言的句子进行编码。

  请参阅Neptune ML 中文本特征的句子 BERT (SBERT) 编码。

- **"text_word2vec"** – 此编码使用 Word2Vec 算法将文本属性值转换为数字向量。它仅支持英语。

  请参阅Neptune ML 中文本特征的 Word2Vec 编码。

- **"text_tfidf"** – 此编码使用术语频率 - 反向文档频率 (TF-IDF) 向量器将文本属性值转换为数字向量。

  您可以使用 ngram_range 字段、min_df 字段和 max_features 字段来定义 text_tfidf 特征编码的参数。

  请参阅Neptune ML 中文本特征的 TF-IDF 编码。

- **"none"** – 使用 none 类型会导致不进行任何特征编码。而是对原始属性值进行解析并保存。

仅当您计划在自定义模型训练过程中执行自己的自定义特征编码时，才使用 none。

`norm` 字段

此字段是数字特征的必需字段。它指定了一种用于数值的规范化方法：

```
"norm": "min-max"
```

支持以下规范化方法：

- “min-max”– 通过从每个值中减去最小值，然后将其除以最大值和最小值之间的差值来规范化每个值。
- “standard”- 通过将每个值除以所有值的总和来对其进行规范化。
- “none”- 在编码过程中不对数值进行规范化。

请参阅Neptune ML 中的数值特征。

`language` 字段

语言字段指定文本属性值中使用的语言。它的用法取决于文本编码方法：

- 对于 `text_fasttext` 编码，此字段为必需字段，并且必须指定以下语言之一：
  - `en`（英语）
  - `zh`（中文）
  - `hi`（印地语）
  - `es`（西班牙语）
  - `fr`（法语）
- 对于 `text_sbert` 编码，不使用此字段，因为 SBERT 编码是多语言的。
- 对于 `text_word2vec` 编码，此字段是可选的，因为 `text_word2vec` 仅支持英语。如果存在，则它必须指定“英语”语言模型的名称：

  ```
  "language" : "en_core_web_lg"
  ```

- 对于 `text_tfidf` 编码，不使用此字段。

max_length 字段

对于 text_fasttext 特征，max_length 字段是可选的，它指定输入文本特征中要编码的最大令牌数。长度大于 max_length 的输入文本会被截断。例如，将 max_length 设置为 128，表示将忽略文本序列中第 128 个之后的任何令牌：

```
"max_length": 128
```

separator 字段

此字段可选择性地与 category、numerical 和 auto 特征一起使用。它指定一个字符，该字符可用于将一个属性值拆分为多个类别值或数值：

```
"separator": ";"
```

只有当属性在单个字符串中存储多个分隔值（例如 "Actor;Director" 或 "0.1;0.2"）时，才使用 separator 字段。

请参阅分类特征、数值特征和 自动编码。

range 字段

此字段是 bucket_numerical 特征的必需字段。它指定了要分成桶的数值范围，格式为 [*lower-bound*, *upper-bound*]：

```
"range" : [20, 100]
```

如果属性值小于下限，则将其分配给第一个桶；如果值大于上限，则将其分配给最后一个桶。

请参阅Neptune ML 中的桶数值特征。

bucket_cnt 字段

此字段是 bucket_numerical 特征的必需字段。它指定由 range 参数定义的数值范围应划分为的桶数：

```
"bucket_cnt": 10
```

请参阅Neptune ML 中的桶数值特征。

`slide_window_size` 字段

此字段可以选择与 `bucket_numerical` 特征一起使用，以便为多个桶分配值：

```
"slide_window_size": 5
```

滑动窗口的工作方式是，Neptune ML 采用窗口大小 **`s`**，并将属性的每个数值 **`v`** 转换为从 `v - s/2` 到 `v + s/2` 的范围。然后将该值分配给范围重叠的每个桶。

请参阅Neptune ML 中的桶数值特征。

`imputer` 字段

此字段可选择与 `numerical` 和 `bucket_numerical` 特征一起使用，以提供一种用于填充缺失值的插补技术：

```
"imputer": "mean"
```

支持的插补技术有：

- `"mean"`
- `"median"`
- `"most-frequent"`

如果不包含 imputer 参数，则当遇到缺失值时，数据预处理将停止并退出。

请参阅 Neptune ML 中的数值特征 和 Neptune ML 中的桶数值特征。

`max_features` 字段

`text_tfidf` 特征可以选择使用此字段来指定要编码的最大术语数：

```
"max_features": 100
```

设置为 100 会导致 TF-IDF 向量器仅对 100 个最常见的术语进行编码。如果不包括 `max_features`，则默认值为 5000。

请参阅Neptune ML 中文本特征的 TF-IDF 编码。

`min_df` 字段

`text_tfidf` 特征可以选择使用此字段来指定要编码的术语的最小文档频率：

```
"min_df": 5
```

设置为 5 表示术语必须出现在至少 5 个不同的属性值中才能进行编码。

未包含 `min_df` 参数时的默认值为 2。

请参阅Neptune ML 中文本特征的 TF-IDF 编码。

`ngram_range` 字段

`text_tfidf` 特征可以选择使用该字段来指定字词或令牌的哪些大小序列应被视为要编码的潜在单个术语：

```
"ngram_range": [2, 4]
```

值 `[2, 4]` 指定应将 2、3 和 4 个字词的序列视为潜在的单个术语。

如果您未明确设置 `ngram_range`，则默认值为 `[1, 1]`，这意味着只有单个字词或令牌被视为要编码的术语。

请参阅Neptune ML 中文本特征的 TF-IDF 编码。

`datetime_parts` 字段

特征 `datetime` 可以选择使用此字段来指定对日期时间值的哪些部分进行分类编码：

```
"datetime_parts": ["weekday", "hour"]
```

如果不包括 `datetime_parts`，则默认情况下，Neptune ML 会对日期时间值的年、月、工作日和小时部分进行编码。值 `["weekday", "hour"]` 表示只应在该特征中对日期时间值的工作日和小时进行分类编码。

如果其中一个部分在训练集中没有多个唯一值，则不会对其进行编码。

请参阅Neptune ML 中的日期时间特征。

使用 **additionalParams** 中的参数调整模型训练配置的示例

目录

使用 **additionalParams** 的属性图示例

为模型训练配置指定默认分割率

在以下示例中，split_rate 参数设置模型训练的默认分割率。如果未指定默认分割率，则训练将使用 [0.9、0.1、0.0] 的值。您可以通过为每个目标指定一个 split_rate 来覆盖每个目标的默认值。

在以下示例中，default split_rate 字段表示应使用分割率 [0.7,0.1,0.2]，除非针对每个目标进行覆盖：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ],
    "features": [
      (...)
    ]
  }
}
```

为模型训练配置指定节点分类任务

要指明哪个节点属性包含用于训练的带标签的示例，请使用 "type" : "classification" 向 targets 数组中添加节点分类元素。如果您想覆盖默认分割率，则可添加 split_rate 字段。

在以下示例中，node 目标表示应将每个 Movie 节点的 genre 属性视为节点类别标签。split_rate 值将覆盖默认分割率：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "genre",
        "type": "classification",
        "split_rate": [0.7,0.1,0.2]
      }
    ],
```

```
    "features": [
      (...)
    ]
  }
}
```

为模型训练配置指定多类别节点分类任务

要指明哪个节点属性包含用于训练的多个带标签的示例，请使用 "type" : "classification" 以及 separator（指定一个可用于将目标属性值拆分为多个分类值的字符），从而将节点分类元素添加到目标数组中。如果您想覆盖默认分割率，则可添加 split_rate 字段。

在以下示例中，node 目标表示应将每个 Movie 节点的 genre 属性视为节点类别标签。separator 字段表示每个流派属性都包含多个以分号分隔的值：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "genre",
        "type": "classification",
        "separator": ";"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

为模型训练配置指定节点回归任务

要指明哪个节点属性包含用于训练目的的带标签的回归，请使用 "type" : "regression" 将节点回归元素添加到目标数组中。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 node 目标表示应将每个 Movie 节点的 rating 属性视为节点回归标签：

```
    "additionalParams": {
  "neptune_ml": {
```

```
    "version": "v2.0",
    "targets": [
      {
        "node": "Movie",
        "property": "rating",
        "type" : "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ],
    "features": [
      ...
    ]
  }
}
```

为模型训练配置指定边缘分类任务

要指明哪个边缘属性包含用于训练目的的带标签的示例，请使用 `"type" : "regression"` 向 `targets` 数组中添加边缘元素。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 `edge` 目标表示应将每个 `knows` 边缘的 `metAtLocation` 属性视为边缘类别标签：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "knows", "Person"],
        "property": "metAtLocation",
        "type": "classification"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

为模型训练配置指定多类别边缘分类任务

要指明哪个边缘属性包含用于训练目的的多个带标签的示例，请使用 `"type" : "classification"` 和 `separator` 字段（此字段指定用于将目标属性值拆分为多个类别值的字

符），向 targets 数组添加一个边缘元素。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 edge 目标表示应将每个 repliedTo 边缘的 sentiment 属性视为边缘类别标签。分隔符字段表示每个情绪属性都包含多个以逗号分隔的值：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "repliedTo", "Message"],
        "property": "sentiment",
        "type": "classification",
        "separator": ","
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

为模型训练配置指定边缘回归

要指明哪个边缘属性包含用于训练目的的带标签的回归示例，请使用 "type" : "regression" 向 targets 数组中添加 edge 元素。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 edge 目标表示应将每个 reviewed 边缘的 rating 属性视为边缘回归：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Person", "reviewed", "Movie"],
        "property": "rating",
        "type" : "regression"
      }
    ],
    "features": [
      (...)
    ]
```

```
  }
}
```

为模型训练配置指定链接预测任务

要指示应使用哪些边缘进行链接预测训练目的，请使用 "type" : "link_prediction" 向目标数组添加边缘元素。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 edge 目标表示应使用 cites 边缘进行链接预测：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "edge": ["Article", "cites", "Article"],
        "type" : "link_prediction"
      }
    ],
    "features": [
      (...)
    ]
  }
}
```

指定数值桶特征

您可以通过将 "type": "bucket_numerical" 添加到 features 数组中来为节点属性指定数值数据特征。

以下 node 特征表示应将每个 Person 节点的 age 属性视为数字桶特征：

```
  "additionalParams": {
  "neptune_ml": {
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Person",
        "property": "age",
        "type": "bucket_numerical",
        "range": [1, 100],
```

```
        "bucket_cnt": 5,
        "slide_window_size": 3,
        "imputer": "median"
      }
    ]
  }
}
```

指定 **Word2Vec** 特征

您可以通过将 `"type": "text_word2vec"` 添加到 `features` 数组中来为节点属性指定 `Word2Vec` 特征。

以下 node 特征表示应将每个 `Movie` 节点的 `description` 属性视为 `Word2Vec` 特征：

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "description",
        "type": "text_word2vec",
        "language": "en_core_web_lg"
      }
    ]
  }
}
```

指定 **FastText** 特征

您可以通过将 `"type": "text_fasttext"` 添加到 `features` 数组中来为节点属性指定 `FastText` 特征。`language` 字段为必需字段，并且必须指定以下语言之一：

- `en`（英语）
- `zh`（中文）
- `hi`（印地语）
- `es`（西班牙语）

- fr（法语）

请注意，在特征中，text_fasttext 编码不能同时处理多种语言。

以下 node 特征表示应将每个 Movie 节点的法语 description 属性视为 FastText 特征：

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "description",
        "type": "text_fasttext",
        "language": "fr",
        "max_length": 1024
      }
    ]
  }
}
```

指定 Sentence BERT 特征

您可以通过将 "type": "text_sbert" 添加到 features 数组中来为节点属性指定 Sentence BERT 特征。您无需指定语言，因为该方法使用多语言模型自动对文本特征进行编码。

以下 node 特征表示应将每个 Movie 节点的 description 属性视为 Sentence BERT 特征：

```
"additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "description",
        "type": "text_sbert128",
```

```
      }
    ]
  }
}
```

指定 **TF-IDF** 特征

您可以通过将 "type": "text_tfidf" 添加到 features 数组中来为节点属性指定 TF-IDF 特征。

以下 node 特征表示应将每个 Person 节点的 bio 属性视为 TF-IDF 特征：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Movie",
        "property": "bio",
        "type": "text_tfidf",
        "ngram_range": [1, 2],
        "min_df": 5,
        "max_features": 1000
      }
    ]
  }
}
```

指定 **datetime** 特征

导出过程会自动推理日期属性的 datetime 特征。但是，如果要限制用于 datetime 特征的 datetime_parts，或者覆盖特征规范，以便将通常被视为 auto 特征的属性显式地视为 datetime 特征，则可以通过向特征数组中添加 "type": "datetime" 来实现。

以下 node 特征表示应将每个 Post 节点的 createdAt 属性视为 datetime 特征：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
```

```
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Post",
        "property": "createdAt",
        "type": "datetime",
        "datetime_parts": ["month", "weekday", "hour"]
      }
    ]
  }
}
```

指定 **category** 特征

导出过程会自动推理字符串属性和包含多个值的数值属性的 auto 特征。对于包含单个值的数值属性，它会推理 numerical 特征。对于日期属性，它会推理 datetime 特征。

如果要覆盖特征规范以便将属性视为分类特征，请向特征数组中添加 "type": "category"。如果该属性包含多个值，请包含 separator 字段。例如：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Post",
        "property": "tag",
        "type": "category",
        "separator": "|"
      }
    ]
  }
}
```

指定 **numerical** 特征

导出过程会自动推理字符串属性和包含多个值的数值属性的 auto 特征。对于包含单个值的数值属性，它会推理 numerical 特征。对于日期属性，它会推理 datetime 特征。

如果要覆盖特征规范以便将属性视为 numerical 特征，请向特征数组中添加 "type": "numerical"。如果该属性包含多个值，请包含 separator 字段。例如：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "Recording",
        "property": "duration",
        "type": "numerical",
        "separator": ","
      }
    ]
  }
}
```

指定 **auto** 特征

导出过程会自动推理字符串属性和包含多个值的数值属性的 auto 特征。对于包含单个值的数值属性，它会推理 numerical 特征。对于日期属性，它会推理 datetime 特征。

如果要覆盖特征规范以便将属性视为 auto 特征，请向特征数组中添加 "type": "auto"。如果该属性包含多个值，请包含 separator 字段。例如：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      ...
    ],
    "features": [
      {
        "node": "User",
        "property": "role",
        "type": "auto",
        "separator": ","
      }
    ]
  }
```

```
}
```

使用 **additionalParams** 的 RDF 示例

为模型训练配置指定默认分割率

在以下示例中，split_rate 参数设置模型训练的默认分割率。如果未指定默认分割率，则训练将使用 [0.9、0.1、0.0] 的值。您可以通过为每个目标指定一个 split_rate 来覆盖每个目标的默认值。

在以下示例中，default split_rate 字段表示应使用分割率 [0.7,0.1,0.2]，除非针对每个目标进行覆盖：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "split_rate": [0.7,0.1,0.2],
    "targets": [
      (...)
    ]
  }
}
```

为模型训练配置指定节点分类任务

要指明哪个节点属性包含用于训练的带标签的示例，请使用 "type" : "classification" 向 targets 数组中添加节点分类元素。添加节点字段以指示目标节点的节点类型。添加 predicate 字段以定义哪些文本数据用作目标节点的目标节点特征。如果您想覆盖默认分割率，则可添加 split_rate 字段。

在以下示例中，node 目标表示应将每个 Movie 节点的 genre 属性视为节点类别标签。split_rate 值将覆盖默认分割率：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/genre",
        "type": "classification",
        "split_rate": [0.7,0.1,0.2]
```

```
      }
    ]
  }
}
```

为模型训练配置指定节点回归任务

要指明哪个节点属性包含用于训练目的的带标签的回归，请使用 "type" : "regression" 将节点回归元素添加到目标数组中。添加 node 字段以指示目标节点的节点类型。添加 predicate 字段以定义哪些文本数据用作目标节点的目标节点特征。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 node 目标表示应将每个 Movie 节点的 rating 属性视为节点回归标签：

```
    "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "node": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/rating",
        "type": "regression",
        "split_rate": [0.7,0.1,0.2]
      }
    ]
  }
}
```

为特定边缘指定链接预测任务

要指示应使用哪些边缘进行链接预测训练目的，请使用 "type" : "link_prediction" 向目标数组添加边缘元素。添加 subject、predicate 和 object 字段以指定边缘类型。如果您想覆盖默认分割率，则可添加 split_rate 字段。

以下 edge 目标表示应使用将 Directors 连接到 Movies 的 directed 边缘进行链路预测：

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "subject": "http://aws.amazon.com/neptune/csv2rdf/class/Director",
```

```
        "predicate": "http://aws.amazon.com/neptune/csv2rdf/datatypeProperty/directed",
        "object": "http://aws.amazon.com/neptune/csv2rdf/class/Movie",
        "type" : "link_prediction"
      }
    ]
  }
}
```

为所有边缘指定链接预测任务

要指示应使用所有边缘进行链接预测训练目的，请使用 `"type" : "link_prediction"` 向目标数组添加 `edge` 元素。请勿添加 `subject`、`predicate` 或 `object` 字段。如果您想覆盖默认分割率，则可添加 `split_rate` 字段。

```
  "additionalParams": {
  "neptune_ml": {
    "version": "v2.0",
    "targets": [
      {
        "type" : "link_prediction"
      }
    ]
  }
}
```

处理从 Neptune 导出的图形数据以用于训练

数据处理步骤采用导出过程创建的 Neptune 图形数据，并创建深度图表库 (DGL) 在训练期间使用的信息。这包括执行各种数据映射和转换：

- 解析节点和边缘以构造 DGL 所需的图形映射和 ID 映射文件。
- 将节点和边缘属性转换为 DGL 所需的节点和边缘特征。
- 将数据拆分为训练集、验证集和测试集。

管理 Neptune ML 的数据处理步骤

从 Neptune 中导出要用于模型训练的数据后，可以使用 `curl`（或 `awscurl`）命令启动数据处理任务，如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for the new job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)",
        "configFileName" : "training-job-configuration.json"
      }'
```

dataprocessing 命令中解释了如何使用此命令的详细信息，以及有关如何获取正在运行的任务的状态、如何停止正在运行的任务以及如何列出所有正在运行的任务的信息。

处理 Neptune ML 的更新图形数据

您也可以向 API 提供 `previousDataProcessingJobId`，以确保新的数据处理任务使用与先前任务相同的处理方法。当您想通过在新数据上重新训练旧模型，或者在新数据上重新计算模型构件，来获得对 Neptune 中更新的图形数据的预测时，这是必需的。

为此，您可以使用如下所示的 `curl`（或 `awscurl`）命令：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
```

```
 -H 'Content-Type: application/json' \
 -d '{ "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
folder)",
       "id" : "(a job ID for the new job)",
       "processedDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your output
folder)",
       "previousDataProcessingJobId", "(the job ID of the previous data-processing
job)"}'
```

将 previousDataProcessingJobId 参数的值设置为与训练后的模型对应的先前数据处理任务的任务 ID。

> **Note**
>
> 目前不支持在更新的图形中删除节点。如果在更新的图形中移除了节点，则必须启动一个全新的数据处理任务，而不是使用 previousDataProcessingJobId。

Neptune ML 中的特征编码

属性值有不同的格式和数据类型。为了在机器学习中获得良好的性能，务必将这些值转换为称为特征的数字编码。

作为数据导出和数据处理步骤的一部分，Neptune ML 使用此处描述的特征编码技术来执行特征提取和编码。

> **Note**
>
> 如果您计划在自定义模型实现中实现自己的特征编码，则可以在数据预处理阶段，通过选择 none 作为特征编码类型来禁用自动特征编码。然后，在该节点或边缘属性上不会进行任何特征编码，而是解析原始属性值并将其保存在字典中。数据预处理仍会根据导出的数据集创建 DGL 图形，但是构造的 DGL 图形没有用于训练的预处理特征。
> 仅当您计划在自定义模型训练过程中执行自定义特征编码时，才应使用此选项。有关详细信息，请参阅Neptune ML 中的自定义模型。

Neptune ML 中的分类特征

可以从可能值的固定列表中提取一个或多个不同值的属性是分类特征。在 Neptune ML 中，分类特征使用独热码编码进行编码。以下示例显示了不同食物的属性名称是如何根据其类别进行独热码编码的：

```
 Food        Veg.   Meat   Fruit    Encoding
---------    ----   ----   -----    --------
 Apple         0      0      1         001
 Chicken       0      1      0         010
 Broccoli      1      0      0         100
```

> **Note**
>
> 任何类别特征中的最大类别数为 100。如果一个属性的值类别超过 100 个，则只有其中最常见的 99 个被归入不同的类别，其余的则归入名为 OTHER 的特殊类别。

Neptune ML 中的数值特征

在 Neptune ML 中，任何值为实数的属性都可以编码为数值特征。数值特征使用浮点数进行编码。

您可以指定在对数值特征进行编码时要使用的数据规范化方法，如下所示："norm": "*normalization technique*"。支持以下规范化技术：

- “none”- 在编码过程中不对数值进行规范化。
- “min-max”– 通过从每个值中减去最小值，然后将其除以最大值和最小值之间的差值来规范化每个值。
- “standard”- 通过将每个值除以所有值的总和来对其进行规范化。

Neptune ML 中的桶数值特征

与其使用原始数字表示数值属性，不如将数值压缩成类别。例如，您可以将人们的年龄分为几个类别，例如儿童 (0-20)、年轻人 (20-40)、中年人 (40-60) 和老年人（从 60 岁起）。使用这些数值桶，您可以将数值属性转换为一种分类特征。

在 Neptune ML 中，您可以将数值属性编码为桶数值特征，您必须提供两项内容：

- 以 "range": [*a*, *b*] 形式表示的数值范围，其中 a 和 b 是整数。
- 桶计数，形式为 "bucket_cnt": *c* ，其中 c 是桶的数量，也是一个整数。

然后，Neptune ML 将每个桶的大小计算为 (b - a) / c ，并将每个数值编码为它归入的任何桶的数量。任何小于 a 的值都被视为属于第一个桶，任何大于 b 的值都被视为属于最后一个桶。

您也可以选择通过指定滑动窗口大小将数值归入多个桶中，如下所示： "slide_window_size": *s* ，其中 s 是一个数字。然后，Neptune ML 将属性的每个数值 v 转换为从 v - s/2 到 v + s/2 的范围，并将值 v 分配给该范围涵盖的每个桶。

最后，您还可以选择提供一种填充数值特征和桶数值特征的缺失值的方法。为此，您可以使用 "imputer": "*imputation technique* "，其中插补技术为 "mean"、"median" 或 "most-frequent" 之一。如果不指定插补器，则缺失值可能会导致处理停止。

Neptune ML 中的文本特征编码

对于自由格式文本，Neptune ML 可以使用几种不同的模型将属性值字符串中的令牌序列转换为固定大小的实值向量：

- `text_fasttext` – 使用 fastText 编码。对于使用 fastText 支持的五种语言中的一种且仅一种的特征，推荐使用这种编码。

- `text_sbert`– 使用句子 BERT (SBERT) 编码模型。对于 `text_fasttext` 不支持的文本，建议使用这种编码。
- `text_word2vec`– 使用最初由 Google 发布的 Word2Vec 算法对文本进行编码。Word2Vec 仅支持英语。
- `text_tfidf`– 使用术语频率 - 反向文档频率 (TF-IDF) 向量器对文本进行编码。TF-IDF 编码支持其它编码不支持的统计特征。

Neptune ML 中文本属性值的 fastText 编码

Neptune ML 可以使用 fastText 模型将文本属性值转换为固定大小的实值向量。以下是 fastText 支持的五种语言中任意一种的文本属性值的推荐编码方法：

- `en`（英语）
- `zh`（中文）
- `hi`（印地语）
- `es`（西班牙语）
- `fr`（法语）

请注意，fastText 无法处理包含多种语言字词的句子。

`text_fasttext` 方法可以选择使用一个 `max_length` 字段，该字段指定要编码的文本属性值中的最大令牌数，之后字符串将被截断。当文本属性值包含长字符串时，这可以提高性能，因为如果未指定 `max_length`，则无论字符串长度如何，fastText 都会对所有令牌进行编码。

此示例指定法语电影标题使用 fastText 进行编码：

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      {
        "feature": ["title", "title", "text_fasttext"],
        "language": "fr",
        "max_length": 1024
      }
    ]
```

```
  }
```

Neptune ML 中文本特征的句子 BERT (SBERT) 编码

Neptune ML 可以使用句子 BERT (SBERT) 模型将字符串属性值中的令牌序列转换为固定大小的实值向量。Neptune 支持两种 SBERT 方法：`text_sbert128`（如果您只指定 `text_sbert`，则这是默认方法）和 `text_sbert512`。两者的区别在于编码后的文本属性值字符串的最大长度。`text_sbert128` 编码在编码 128 个令牌后截断文本字符串，而 `text_sbert512` 在编码 512 个令牌后截断文本字符串。因此，`text_sbert512` 所需的处理时间比 `text_sbert128` 更多。两种方法都慢于 `text_fasttext`。

SBERT 编码是多语言的，因此，无需为正在编码的属性值文本指定语言。SBERT 支持多种语言，并且可以对包含多种语言的句子进行编码。如果您使用 fastText 不支持的一种或多种语言对包含文本的属性值进行编码，则推荐使用 SBERT 编码方法。

以下示例指定将电影标题编码为 SBERT，最多 128 个令牌：

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      { "feature": ["title", "title", "text_sbert128"] }
    ]
  }
```

Neptune ML 中文本特征的 Word2Vec 编码

Neptune ML 可以将字符串属性值编码为 Word2Vec 特征（Word2Vec 算法最初由 Google 发布）。`text_word2vec` 方法使用 spaCy 训练模型之一将字符串中的令牌编码为密集向量。这仅支持使用 en_core_web_lg 模型的“英语”语言。

以下示例指定电影标题使用 Word2Vec 进行编码：

```
{
    "file_name" : "nodes/movie.csv",
    "separator" : ",",
    "node" : ["~id", "movie"],
    "features" : [
      {
```

```
        "feature": ["title", "title", "text_word2vec"],
        "language": "en_core_web_lg"
      }
    ]
  }
```

请注意，语言字段是可选的，因为英语 en_core_web_lg 模型是 Neptune 唯一支持的模型。

Neptune ML 中文本特征的 TF-IDF 编码

Neptune ML 可以将文本属性值编码为 text_tfidf 特征。这种编码使用术语频率 - 反向文档频率 (TF-IDF) 向量器将文本中的字词序列转换为数字向量，然后进行降维操作。

TF-IDF（术语频率 – 反向文档频率）是一个数值，旨在衡量字词在文档集中的重要性。它的计算方法是将字词在给定属性值中出现的次数除以该词出现在此类属性值中的总次数。

例如，如果“kiss”一词在给定的电影标题中出现两次（比如“kiss kiss bang bang”），并且“kiss”总共出现在 4 部电影的标题中，那么“kiss kiss bang bang”标题中“kiss”的 TF-IDF 值将为 2 / 4 。

最初创建的向量具有 d 个维度，其中 d 是该类型的所有属性值中唯一术语的数量。降维操作使用随机稀疏投影将该数字减少到最大值 100。然后，通过合并图形中的所有 text_tfidf 特征来生成图形的词汇表。

您可以通过多种方式控制 TF-IDF 向量器：

- **max_features** – 使用 max_features 参数，您可以将 text_tfidf 特征中的术语数量限制为最常见的术语。例如，如果您将 max_features 设置为 100，则仅包括前 100 个最常用的术语。如果您未显式设置 max_features，则其默认值为 5000。
- **min_df** – 使用 min_df 参数，您可以将 text_tfidf 特征中的术语数量限制为至少具有指定文档频率的术语数量。例如，如果您将 min_df 设置为 5，则仅使用出现在至少 5 个不同属性值中的术语。如果您未显式设置 min_df，则其默认值为 2。
- **ngram_range** – ngram_range 参数确定将哪些字词组合视为术语。例如，如果您将 ngram_range 设置为 [2, 4]，则可以在“kiss kiss bang bang”标题中找到以下 6 个术语：
  - 2 个字词的术语：“kiss kiss”、“kiss bang”和“bang bang”。
  - 3 个字词的术语：“kiss kiss bang”和“kiss bang bang”。
  - 4 个字词的术语：“kiss kiss bang bang”。

  ngram_range 的默认设置是 [1, 1]。

Neptune ML 中的日期时间特征

Neptune ML 可以通过将部分 `datetime` 属性值编码为独热码数组来将它们转换为分类特征。使用 `datetime_parts` 参数指定要编码的以下一个或多个部分：`["year", "month", "weekday", "hour"]`。如果未设置 `datetime_parts`，则默认情况下，所有四个部分均编码。

例如，如果日期时间值的范围跨越 2010 年到 2012 年，则日期时间条目 `2011-04-22 01:16:34` 的四个部分如下所示：

- year – `[0, 1, 0]`。

  由于这段时间跨度只有 3 年（2010 年、2011 年和 2012 年），因此独热码数组有三个条目，每年一个。
- month – `[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]`。

  在这里，独热码数组对于一年中的每个月份有一个条目。
- weekday – `[0, 0, 0, 0, 1, 0, 0]`。

  ISO 8601 标准规定，星期一是一周的第一天，由于 2011 年 4 月 22 日是星期五，因此相应的独热码工作日数组在第五个位置是热的。
- hour – `[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`。

  凌晨 1 点是在具有 24 个成员的独热码数组中设置的。

月份中的某天、分钟和秒没有明确编码。

如果所讨论的总 `datetime` 范围仅包括一年内的日期，则不对 `year` 数组进行编码。

您可以使用 `imputer` 参数和一种可用于数值特征的策略来指定填补缺失 `datetime` 值的插补策略。

Neptune ML 中的自动特征编码

您可以将 `auto` 设置为特征编码方法，而不必手动指定要用于图形中属性的特征编码方法。然后，Neptune ML 尝试根据每个属性的底层数据类型推理其最佳特征编码。

以下是 Neptune ML 在选择相应的特征编码时使用的一些启发式方法：

- 如果该属性只有数值并且可以转换为数值数据类型，那么 Neptune ML 通常将其编码为数值。但是，如果该属性的唯一值数量小于值总数的 10%，并且这些唯一值的基数小于 100，则 Neptune ML 将使用分类编码。
- 如果可以将属性值强制转换为 `datetime` 类型，则 Neptune ML 将它们编码为 `datetime` 特征。
- 如果可以将属性值强制为布尔值（1/0 或 True/False），那么 Neptune ML 将使用类别编码。
- 如果属性是一个字符串，其值的 10% 以上是唯一的，并且每个值的平均令牌数大于或等于 3，则 Neptune ML 推断属性类型为文本，并自动检测所使用的语言。如果检测到的语言是 fastText 支持的语言之一，即英语、中文、印地语、西班牙语和法语，则 Neptune ML 使用 `text_fasttext` 对文本进行编码。否则，Neptune ML 使用 `text_sbert`。
- 如果该属性是未归类为文本特征的字符串，则 Neptune ML 会将其假定为分类特征并使用类别编码。
- 如果每个节点对推理为类别特征的属性都有其自己的唯一值，则 Neptune ML 会将该属性从训练图形中删除，因为它可能是一个无法提供学习信息的 ID。
- 如果已知该属性包含有效的 Neptune 分隔符，例如分号 (";")，则 Neptune ML 只能将该属性视为 `MultiNumerical` 或 `MultiCategorical`。
  - Neptune ML 首先尝试将值编码为数字特征。如果成功，Neptune ML 将使用数字编码来创建数字向量特征。
  - 否则，Neptune ML 会将这些值编码为多类别。
- 如果 Neptune ML 无法推理属性值的数据类型，则 Neptune ML 会从训练图形中删除该属性。

编辑训练数据配置文件

Neptune 导出过程将 Neptune ML 数据从 Neptune 数据库集群导出到 S3 桶中。它将节点和边缘分别导出到 nodes/ 和 edges/ 文件夹中。它还会创建一个 JSON 训练数据配置文件，默认情况下命名为 training-data-configuration.json。此文件包含有关图形的架构、其特征的类型、特征转换和规范化操作以及分类或回归任务的目标特征的信息。

在某些情况下，您可能想要直接修改配置文件。其中一种情况是，当您想要更改特征的处理方式或图形的构造方式时，而不必在每次想要修改正在解决的机器学习任务的规范时都重新运行导出。

编辑训练数据配置文件

1. 将文件下载到本地计算机上。

   除非您在传递给导出过程的 additionalParams/neptune_ml 参数中指定了一个或多个命名任务，否则该文件将具有默认名称，即 training-data-configuration.json。您可以使用如下的 AWS CLI 命令来下载此文件：

   ```
   aws s3 cp \
     s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-
   configuration.json \
     ./
   ```

2. 使用文本编辑器编辑文件。
3. 上传修改后的文件。使用如下所示的 AWS CLI 命令，将修改后的文件上传回 Amazon S3 中您从中下载该文件的相同位置：

   ```
   aws s3 cp \
     training-data-configuration.json \
     s3://(your Amazon S3 bucket)/(path to your export folder)/training-data-
   configuration.json
   ```

JSON 训练数据配置文件示例

以下是描述节点分类任务图形的示例训练数据配置文件：

```
{
  "version" : "v2.0",
  "query_engine" : "gremlin",
  "graph" : [
```

```
{
  "edges" : [
    {
      "file_name" : "edges/(movie)-included_in-(genre).csv",
      "separator" : ",",
      "source" : ["~from", "movie"],
      "relation" : ["", "included_in"],
      "dest" : [ "~to", "genre" ]
    },
    {
      "file_name" : "edges/(user)-rated-(movie).csv",
      "separator" : ",",
      "source" : ["~from", "movie"],
      "relation" : ["rating", "prefixname"], # [prefixname#value]
      "dest" : ["~to", "genre"],
      "features" : [
        {
          "feature" : ["rating", "rating", "numerical"],
          "norm" : "min-max"
        }
      ]
    }
  ],
  "nodes" : [
    {
      "file_name" : "nodes/genre.csv",
      "separator" : ",",
      "node" : ["~id", "genre"],
      "features" : [
        {
          "feature": ["name", "genre", "category"],
          "separator": ";"
        }
      ]
    },
    {
      "file_name" : "nodes/movie.csv",
      "separator" : ",",
      "node" : ["~id", "movie"],
      "features" : [
        {
          "feature": ["title", "title", "word2vec"],
          "language": ["en_core_web_lg"]
        }
```

```
          ]
        },
        {
          "file_name" : "nodes/user.csv",
          "separator" : ",",
          "node" : ["~id", "user"],
          "features" : [
            {
              "feature": ["age", "age", "numerical"],
              "norm" : "min-max",
              "imputation": "median",
            },
            {
              "feature": ["occupation", "occupation", "category"],
            }
          ],
          "labels" : [
            {
              "label": ["gender", "classification"],
              "split_rate" : [0.8, 0.2, 0.0]
            }
          ]
        }
      ]
    },
    "warnings" : [ ]
  ]
}
```

JSON 训练数据配置文件的结构

训练配置文件是指由导出过程保存在 nodes/ 和 edges/ 文件夹中的 CSV 文件。

nodes/ 下的每个文件都存储有关具有相同属性图节点标签的节点的信息。节点文件中的每一列都存储节点 ID 或节点属性。文件的第一行包含一个标题，该标题为每列指定 ~id 或属性名称。

edges/ 下的每个文件都存储有关具有相同属性图边缘标签的节点的信息。节点文件中的每列都存储源节点 ID、目标节点 ID 或边缘属性。文件的第一行包含一个标题，该标题指定每列的 ~from、~to 或属性名称。

训练数据配置文件包含三个顶级元素：

```
{
```

```
  "version" : "v2.0",
  "query_engine" : "gremlin",
  "graph" : [ ... ]
}
```

- `version` – (字符串) 正在使用的配置文件的版本。
- `query_engine` – (字符串) 用于导出图形数据的查询语言。目前，只有“gremlin”是有效的。
- `graph` – (JSON 数组) 列出一个或多个配置对象，这些对象包含将要使用的每个节点和边缘的模型参数。

  图形数组中的配置对象具有下一节中描述的结构。

`graph` 数组中列出的配置对象的内容

`graph` 数组中的配置对象可以包含三个顶级节点：

```
  {
    "edges"    : [ ... ],
    "nodes"    : [ ... ],
    "warnings" : [ ... ],
  }
```

- `edges` – (JSON 对象数组) 每个 JSON 对象指定一组参数，用于定义在模型处理和训练期间如何处理图形中的边缘。这仅用于 Gremlin 引擎。
- `nodes` – (JSON 对象数组) 每个 JSON 对象指定一组参数，用于定义在模型处理和训练期间如何处理图形中的节点。这仅用于 Gremlin 引擎。
- `warnings` – (JSON 对象数组) 每个对象都包含在数据导出过程中生成的警告。

`edges` 数组中列出的边缘配置对象的内容

`edges` 数组中列出的边缘配置对象可以包含以下顶级字段：

```
    {
      "file_name" : "(path to a CSV file)",
      "separator" : "(separator character)",
      "source"    : ["(column label for starting node ID)", "(starting node type)"],
      "relation"  : ["(column label for the relationship name)", "(the prefix name
for the relationship name)"],
```

```
    "dest"      : ["(column label for ending node ID)", "(ending node type)"],
    "features"  : [(array of feature objects)],
    "labels"    : [(array of label objects)]
  }
```

- **file_name**– 一个字符串，用于指定 CSV 文件的路径，该文件存储有关具有相同属性图标签的边缘的信息。

  该文件的第一行包含列标签的标题行。

  前两个列标签为 ~from 和 ~to。第一列（~from 列）存储边缘的起始节点的 ID，第二列（~to 列）存储边缘的结束节点的 ID。

  标题行中的其余列标签为剩下的每列指定边缘属性的名称，该属性的值已导出到该列中。
- **separator** - 包含分隔该 CSV 文件中各列的分隔符的字符串。
- **source** – 一个 JSON 数组，其中包含两个用于指定边缘的起始节点的字符串。第一个字符串包含存储起始节点 ID 的列的标题名称。第二个字符串指定节点类型。
- **relation** – 一个 JSON 数组，其中包含两个用于指定边缘关系类型的字符串。第一个字符串包含存储关系名称 (relname) 的列的标题名称。第二个字符串包含关系名称的前缀 (prefixname)。

  完整的关系类型由两个字符串组合而成，它们之间有一个连字符，如下所示：*prefixname*-*relname*。

  如果第一个字符串为空，则所有边缘都具有相同的关系类型，即 prefixname 字符串。
- **dest** – 一个 JSON 数组，其中包含两个用于指定边缘的结束节点的字符串。第一个字符串包含存储结束节点 ID 的列的标题名称。第二个字符串指定节点类型。
- **features** – 属性值特征对象的 JSON 数组。每个属性值特征对象包含以下字段：
  - 特征 – 由三个字符串组成的 JSON 数组。第一个字符串包含一列的标题名称，该列包含属性值。第二个字符串包含特征名称。第三个字符串包含特征类型。
  - norm –（可选）指定要应用于属性值的规范化方法。
- **labels** – 一个由对象组成的 JSON 数组。其中的每个对象都定义了边缘的一个目标特征，并指定了训练和验证阶段应采用的边缘比例。每个对象都包含以下字段：
  - 标签 – 由两个字符串组成的 JSON 数组。第一个字符串包含一列的标题名称，该列包含目标特征属性值。第二个字符串指定以下目标任务类型之一：
    - "classification" – 边缘分类任务。由 label 数组中第一个字符串标识的列中提供的属性值被视为分类值。对于边缘分类任务，label 数组中的第一个字符串不能为空。

  - `"regression"` – 边缘回归任务。由 `label` 数组中第一个字符串标识的列中提供的属性值被视为数值。对于边缘回归任务，`label` 数组中的第一个字符串不能为空。
  - `"link_prediction"` – 链接预测任务。不需要任何属性值。对于链接预测任务，`label` 数组中的第一个字符串将被忽略。
- **`split_rate`** – 一个 JSON 数组，包含三个介于零和一之间的数字，这些数字加起来为 1，分别表示训练、验证和测试阶段将使用的节点比例的估计值。可以定义此字段或 `custom_split_filenames`，但不能同时定义这两者。请参阅 split_rate。
- **`custom_split_filenames`** – 一个 JSON 对象，它为定义训练、验证和测试群体的文件指定文件名。可以定义此字段或 `split_rate`，但不能同时定义这两者。参阅 自定义训练-验证-测试比例 了解更多信息。

`nodes` 数组中列出的节点配置对象的内容

`nodes` 数组中列出的节点配置对象可以包含以下字段：

```
    {
      "file_name" : "(path to a CSV file)",
      "separator" : "(separator character)",
      "node"      : ["(column label for the node ID)", "(node type)"],
      "features"  : [(feature array)],
      "labels"    : [(label array)],
    }
```

- **`file_name`**– 一个字符串，用于指定 CSV 文件的路径，该文件存储有关具有相同属性图标签的节点的信息。

  该文件的第一行包含列标签的标题行。

  第一列的标签为 `~id`，第一列（`~id` 列）存储节点 ID。

  标题行中的其余列标签为剩下的每列指定节点属性的名称，该属性的值已导出到该列中。
- **`separator`** - 包含分隔该 CSV 文件中各列的分隔符的字符串。
- **`node`**– 包含两个字符串的 JSON 数组。第一个字符串包含存储节点 ID 的列的标题名称。第二个字符串指定图形中的节点类型，该类型对应于该节点的属性图标签。
- **`features`** – 节点特征对象的 JSON 数组。请参阅在节点或边缘的 `features` 数组中列出的特征对象的内容。
- **`labels`** – 节点标签对象的 JSON 数组。请参阅节点 `labels` 数组中列出的节点标签对象的内容。

在节点或边缘的 **features** 数组中列出的特征对象的内容

节点 features 数组中列出的节点特征对象可以包含以下顶级字段：

- **feature** – 由三个字符串组成的 JSON 数组。第一个字符串包含一列的标题名称，该列包含特征的属性值。第二个字符串包含特征名称。

  第三个字符串包含特征类型。特征的 type 字段的可能值中列出了有效的特征类型。
- **norm** – 此字段是数字特征的必需字段。它指定了一种用于数值的规范化方法。有效值为 "none"、"min-max" 和“standard”。有关详细信息，请参阅norm 字段。
- **language** – 语言字段指定文本属性值中使用的语言。它的用法取决于文本编码方法：
  - 对于 text_fasttext 编码，此字段为必需字段，并且必须指定以下语言之一：
    - en（英语）
    - zh（中文）
    - hi（印地语）
    - es（西班牙语）
    - fr（法语）

    但是，text_fasttext 无法同时处理多种语言。
  - 对于 text_sbert 编码，不使用此字段，因为 SBERT 编码是多语言的。
  - 对于 text_word2vec 编码，此字段是可选的，因为 text_word2vec 仅支持英语。如果存在，则它必须指定“英语”语言模型的名称：

    ```
    "language" : "en_core_web_lg"
    ```
  - 对于 tfidf 编码，不使用此字段。
- **max_length** – 对于 text_fasttext 特征，此字段是可选字段，它指定输入文本特征中将要编码的最大令牌数。达到 max_length 后的输入文本将被忽略。例如，将 max_length 设置为 128 表示将忽略文本序列中第 128 个之后的任何令牌。
- **separator** – 此字段可选择性地与 category、numerical 和 auto 特征一起使用。它指定一个字符，该字符可用于将一个属性值拆分为多个类别值或数值。

  请参阅separator 字段。
- **range** – 此字段是 bucket_numerical 特征的必需字段。它指定了要分成桶的数值范围。

  请参阅range 字段。

- **bucket_cnt** – 此字段是 bucket_numerical 特征的必需字段。它指定由 range 参数定义的数值范围应划分为的桶数。

  请参阅Neptune ML 中的桶数值特征。

- **slide_window_size** – 此字段可以选择与 bucket_numerical 特征一起使用，以便为多个桶分配值。

  请参阅slide_window_size 字段。

- **imputer** – 此字段可选择与 numerical、bucket_numerical 和 datetime 特征一起使用，以提供一种用于填充缺失值的插补技术。支持的插补技术有 "mean"、"median" 和 "most_frequent"。

  请参阅imputer 字段。

- **max_features** – text_tfidf 特征可以选择使用此字段来指定要编码的最大术语数。

  请参阅max_features 字段。

- **min_df** – text_tfidf 特征可以选择使用此字段来指定要编码的术语的最小文档频率

  请参阅min_df 字段。

- **ngram_range** – text_tfidf 特征可以选择使用此字段来指定一系列字词或令牌，这些字词或令牌被视为要编码的潜在单个术语

  请参阅ngram_range 字段。

- **datetime_parts** – datetime 特征可以选择使用此字段来指定对日期时间值的哪些部分进行分类编码。

  请参阅datetime_parts 字段。

节点 **labels** 数组中列出的节点标签对象的内容

节点 labels 数组中列出的标签对象定义节点目标特征，并指定训练、验证和测试阶段将使用的节点比例。每个对象可以包含以下字段：

```
    {
      "label"      : ["(column label for the target feature property value)", "(task
type)"],
      "split_rate" : [(training proportion), (validation proportion), (test
proportion)],
```

```
        "custom_split_filenames" : {"train": "(training file name)", "valid":
"(validation file name)", "test": "(test file name)"},
        "separator"  : "(separator character for node-classification category values)",
      }
```

- **label** – 包含两个字符串的 JSON 数组。第一个字符串包含一列的标题名称，该列存储特征的属性值。第二个字符串指定目标任务类型，可以是：
  - "classification" – 节点分类任务。指定列中的属性值用于创建分类特征。
  - "regression" – 节点回归任务。指定列中的属性值用于创建数值特征。
- **split_rate** – 一个 JSON 数组，包含三个介于零和一之间的数字，这些数字加起来为 1，分别表示训练、验证和测试阶段将使用的节点比例的估计值。请参阅split_rate。
- **custom_split_filenames** – 一个 JSON 对象，它为定义训练、验证和测试群体的文件指定文件名。可以定义此字段或 split_rate，但不能同时定义这两者。参阅 自定义训练-验证-测试比例 了解更多信息。
- **separator** – 包含分隔符的字符串，分隔符用于分隔分类任务的分类特征值。

> **Note**
>
> 如果没有为边缘和节点提供标签对象，则会自动假定该任务是链接预测，并且边缘会随机拆分成 90% 用于训练，10% 用于验证。

自定义训练-验证-测试比例

默认情况下，Neptune ML 使用 split_rate 参数，通过在此参数中定义的比例，将图形随机拆分为训练、验证和测试群体。为了更精确地控制在这些不同的群体中使用哪些实体，可以创建显式定义它们的文件，然后可以编辑训练数据配置文件以将这些索引文件映射到各个群体。此映射由训练配置文件中 custom_split_filesnames 键的 JSON 对象指定。如果使用此选项，则必须为 train 和 validation 键提供文件名，而对于 test 键是可选的。

这些文件的格式应与 Gremlin 数据格式相匹配。具体而言，对于节点级任务，每个文件都应包含一列，其 ~id 标题中列出了节点 ID；对于边缘级任务，文件应分别指定 ~from 和 ~to 以指明边缘的源节点和目标节点。这些文件需要放在与用于数据处理的导出数据相同的 Amazon S3 位置（请参阅：outputS3Path）。

对于属性分类或回归任务，这些文件可以选择为机器学习任务定义标签。在这种情况下，文件需要有一个属性列，其标题名称与在训练数据配置文件中定义的名称相同。如果在导出的节点和边缘文件以及自定义拆分文件中都定义了属性标签，则优先考虑自定义拆分文件。

使用 Neptune ML 训练模型

处理完从 Neptune 导出的用于模型训练的数据后，可以使用 curl（或 awscurl）命令启动模型训练任务，如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
      }'
```

modeltraining 命令中解释了如何使用此命令的详细信息，以及有关如何获取正在运行的任务的状态、如何停止正在运行的任务以及如何列出所有正在运行的任务的信息。

您还可以提供 previousModelTrainingJobId 以使用已完成的 Neptune ML 模型训练任务中的信息，从而在新的训练任务中加快超参数搜索的速度。这在对新的图形数据进行模型再训练以及对相同图形数据进行增量训练期间非常有用。使用如下命令：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "previousModelTrainingJobId" : "(the model-training job-id of a completed job)"
      }'
```

您可以通过提供 customModelTrainingParameters 对象在 Neptune ML 训练基础设施上训练自己的模型实现，如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
```

```
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

有关更多信息，例如如何获取正在运行的任务的状态、如何停止正在运行的任务以及如何列出所有正在运行的任务，请参阅modeltraining 命令。有关如何实现和使用自定义模型的信息，请参阅Neptune ML 中的自定义模型。

主题

- Amazon Neptune ML 中的模型和模型训练
- 在 Neptune ML 中自定义模型超参数配置
- 模型训练最佳实践

Amazon Neptune ML 中的模型和模型训练

Neptune ML 使用图形神经网络 (GNN) 为各种机器学习任务创建模型。事实证明，图形神经网络可以为图形机器学习任务获得最先进的结果，并且非常擅长从图形结构化数据中提取信息模式。

Neptune ML 中的图形神经网络 (GNN)

图形神经网络 (GNN) 属于一个神经网络系列，它们可通过考虑附近节点的结构和特征来计算节点表示形式。GNN 是对其它不太适合图形数据的传统机器学习和神经网络方法的补充。

GNN 用于解决机器学习任务，例如节点分类和回归（预测节点的属性）、边缘分类和回归（预测边缘的属性）或链接预测（预测图形中的两个节点是否应连接）。

通常，使用 GNN 执行机器学习任务涉及两个阶段：

- 编码阶段，其中 GNN 计算图形中每个节点的 D 维向量。这些向量也称为表示形式或嵌入。
- 解码阶段，它根据编码后的表示形式进行预测。

对于节点分类和回归，节点表示形式直接用于分类和回归任务。对于边缘分类和回归，边缘上事件节点的节点表示形式用作分类或回归的输入。对于链接预测，边缘似然性分数是通过使用一对节点表示形式和一种节点类型表示形式来计算的。

深度图表库 (DGL) 有助于为这些任务高效定义和训练 GNN。

不同的 GNN 模型在消息传递的公式下是统一的。在此视图中，图形中节点的表示形式是使用该节点的邻居表示形式（消息）以及该节点的初始表示形式来计算的。在 Neptune ML 中，节点的初始表示形式来自从其节点属性中提取的特征，或者是可学习的并取决于节点的身份。

Neptune ML 还提供了连接节点特征和可学习节点表示形式以用作原始节点表示形式的选项。

对于 Neptune ML 中涉及具有节点属性的图形的各种任务，我们使用关系图表卷积网络 (R-GCN) 来执行编码阶段。R-GCN 是一种 GNN 架构，非常适合具有多个节点和边缘类型的图形（这些图形称为异构图形）。

R-GCN 网络由固定数量的层组成，一层接一层地堆叠。R-GCN 的每一层都使用其可学习的模型参数来聚合来自节点的直接 1 跳邻域的信息。由于后续层使用前一层的输出表示形式作为输入，因此，影响节点最终嵌入的图形邻域的半径取决于 R-GCN 网络的层数 (`num-layer`)。

例如，这意味着 2 层网络使用来自 2 跳之外的节点的信息。

要了解有关 GNN 的更多信息，请参阅图形神经网络综合调查。有关深度图表库 (DGL) 的更多信息，请访问 DGL 网页。有关在 GNN 中使用 DGL 的实操教程，请参阅使用深度图表库学习图形神经网络。

训练图形神经网络

在机器学习中，让模型学习如何对任务做出正确预测的过程称为模型训练。这通常是通过指定要优化的特定目标以及用于执行此优化的算法来实现的。

此过程用于训练 GNN，以学习下游任务的良好表示形式。我们为该任务创建了一个目标函数，该函数在模型训练期间最小化。例如，对于节点分类，我们使用 CrossEntropyLoss 作为目标，这会惩罚错误的分类；对于节点回归，我们最小化 MeanSquareError。

目标通常是一个损失函数，它获取特定数据点的模型预测，并将其与该数据点的真实情况值进行比较。它返回损失值，该值显示模型的预测接近真实情况的程度。训练过程的目标是最大限度地减少损失，并确保模型预测接近真实情况。

深度学习中用于训练过程的优化算法通常是梯度下降的一种变体。在 Neptune ML 中，我们使用 Adam，这是一种基于低阶矩的自适应估计值对随机目标函数进行基于一阶梯度的优化的算法。

虽然模型训练过程试图确保学习到的模型参数接近目标函数的最小值，但模型的整体性能还取决于模型的超参数，这些超参数是训练算法无法学习的模型设置。例如，学习到的节点表示形式的维度 `num-hidden` 是一个影响模型性能的超参数。因此，在机器学习中，通常会执行超参数优化 (HPO) 来选择合适的超参数。

Neptune ML 使用 SageMaker 超参数调整任务来启动多个具有不同超参数配置的模型训练实例，以尝试为一系列超参数设置找到最佳模型。请参阅在 Neptune ML 中自定义模型超参数配置。

Neptune ML 中的知识图谱嵌入模型

知识图谱 (KG) 是对有关不同实体（节点）及其关系（边缘）的信息进行编码的图形。在 Neptune ML 中，当图形不包含节点属性，而仅包含与其它节点的关系时，默认应用知识图谱嵌入模型来执行链接预测。尽管通过将模型类型指定为 `"rgcn"`，也可以将具有可学习的嵌入内容的 R-GCN 模型用于这些图形，但知识图谱嵌入模型更简单，旨在有效地学习大规模知识图谱的表示形式。

知识图谱嵌入模型用于链接预测任务中，以预测完成三元组 (**h**, **r**, **t**) 的节点或关系，其中 **h** 是源节点，**r** 是关系类型，**t** 是目标节点。

在 Neptune ML 中实现的知识图谱嵌入模型是 `distmult`、`transE` 和 `rotatE`。要了解有关知识图谱嵌入模型的更多信息，请参阅 DGL-KE。

在 Neptune ML 中训练自定义模型

Neptune ML 可让您针对特定场景定义和实现自己的自定义模型。有关如何实现自定义模型以及如何使用 Neptune ML 基础设施对其进行训练的信息，请参阅Neptune ML 中的自定义模型。

在 Neptune ML 中自定义模型超参数配置

启动 Neptune ML 模型训练任务时，Neptune ML 会自动使用从之前的数据处理任务中推理的信息。它使用这些信息生成超参数配置范围，这些范围用于创建 SageMaker 超参数调整任务，以便为您的任务训练多个模型。这样，您就不必为要训练的模型指定一长串超参数值。相反，模型超参数范围和默认值是根据任务类型、图形类型和调整任务设置来选择的。

但是，也可以通过修改数据处理任务生成的 JSON 配置文件，来覆盖默认的超参数配置并提供自定义超参数。

使用 Neptune ML modelTraining API，您可以控制多个高级超参数调整任务设置，例如 `maxHPONumberOfTrainingJobs`、`maxHPOParallelTrainingJobs` 和 `trainingInstanceType`。要更精细地控制模型超参数，可以自定义数据处理任务生成的 `model-HPO-configuration.json` 文件。该文件保存在您为处理任务输出指定的 Amazon S3 位置。

您可以下载该文件，对其进行编辑以覆盖默认的超参数配置，然后将其上传回相同的 Amazon S3 位置。请勿更改文件名，编辑时请注意按照以下说明进行操作。

从 Amazon S3 下载该文件

```
aws s3 cp \
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json \
  ./
```

完成编辑后，将文件上传回原处：

```
aws s3 cp \
  model-HPO-configuration.json \
  s3://(bucket name)/(path to output folder)/model-HPO-configuration.json
```

`model-HPO-configuration.json` 文件的结构

`model-HPO-configuration.json` 文件指定了要训练的模型、机器学习 `task_type` 以及应在模型训练的不同运行中改变或固定的超参数。

超参数归类为属于不同的层，这些层表示调用超参数调整任务时赋予超参数的优先级：

- 第 1 层超参数具有最高优先级。如果将 `maxHPONumberOfTrainingJobs` 设置为小于 10 的值，则只调整第 1 层超参数，其余超参数采用其默认值。

- 第 2 层超参数的优先级较低，因此，如果调整任务的训练任务总数超过 10 但少于 50，则会调整第 1 层和第 2 层超参数。
- 只有当训练任务总数超过 50 时，第 3 层超参数才会与第 1 层和第 2 层一起调整。
- 最后，固定的超参数根本不会进行调整，并且始终采用其默认值。

model-HPO-configuration.json 文件的示例

下面是一个示例 model-HPO-configuration.json 文件。

```
{
  "models": [
    {
      "model": "rgcn",
      "task_type": "node_class",
      "eval_metric": {
        "metric": "acc"
      },
      "eval_frequency": {
          "type":  "evaluate_every_epoch",
          "value":  1
      },
      "1-tier-param": [
        {
            "param": "num-hidden",
            "range": [16, 128],
            "type": "int",
            "inc_strategy": "power2"
        },
        {
          "param": "num-epochs",
          "range": [3,30],
          "inc_strategy": "linear",
          "inc_val": 1,
          "type": "int",
          "node_strategy": "perM"
        },
        {
          "param": "lr",
          "range": [0.001,0.01],
          "type": "float",
          "inc_strategy": "log"
        }
```

```
    ],
    "2-tier-param": [
      {
        "param": "dropout",
        "range": [0.0,0.5],
        "inc_strategy": "linear",
        "type": "float",
        "default": 0.3
      },
      {
        "param": "layer-norm",
        "type": "bool",
        "default": true
      }
    ],
    "3-tier-param": [
      {
        "param": "batch-size",
        "range": [128, 4096],
        "inc_strategy": "power2",
        "type": "int",
        "default": 1024
      },
      {
        "param": "fanout",
        "type": "int",
        "options": [[10, 30],[15, 30], [15, 30]],
        "default": [10, 15, 15]
      },
      {
        "param": "num-layer",
        "range": [1, 3],
        "inc_strategy": "linear",
        "inc_val": 1,
        "type": "int",
        "default": 2
      },
      {
        "param": "num-bases",
        "range": [0, 8],
        "inc_strategy": "linear",
        "inc_val": 2,
        "type": "int",
        "default": 0
```

```
        }
      ],
      "fixed-param": [
        {
          "param": "concat-node-embed",
          "type": "bool",
          "default": true
        },
        {
          "param": "use-self-loop",
          "type": "bool",
          "default": true
        },
        {
          "param": "low-mem",
          "type": "bool",
          "default": true
        },
        {
          "param": "l2norm",
          "type": "float",
          "default": 0
        }
      ]
    }
  ]
}
```

model-HPO-configuration.json 文件的元素

该文件包含一个 JSON 对象，带有一个名为 `models` 的顶级数组，其中包含单个模型配置对象。自定义此文件时，请确保 `models` 数组中只有一个模型配置对象。如果文件包含多个模型配置对象，则调整任务将失败并显示警告。

模型配置对象包含以下顶级元素：

- **`model`** – （字符串）要训练的模型类型（请勿修改）。有效值为：
  - `"rgcn"` – 这是节点分类和回归任务以及异构链接预测任务的默认设置。
  - `"transe"` – 这是 KGE 链接预测任务的默认设置。
  - `"distmult"` – 这是 KGE 链接预测任务的替代模型类型。
  - `"rotate"` – 这是 KGE 链接预测任务的替代模型类型。

通常，不要直接修改 model 值，因为不同的模型类型通常具有明显不同的适用超参数，这可能会在启动训练任务后导致解析错误。

要更改模型类型，请使用 modelTraining API 中的 modelName 参数，而不是在 model-HPO-configuration.json 文件中对其进行更改。

更改模型类型并对超参数进行精细更改的一种方法是：复制要使用的模型的默认模型配置模板并将其粘贴到 model-HPO-configuration.json 文件中。如果推理的任务类型支持多个模型，那么在与 model-HPO-configuration.json 文件相同的 Amazon S3 位置中有一个名为 hpo-configuration-templates 的文件夹。此文件夹包含适用于该任务的其它模型的所有默认超参数配置。

例如，如果要将 KGE 链接预测任务的模型和超参数配置从默认 transe 模型更改为 distmult 模型，只需将 hpo-configuration-templates/distmult.json 文件的内容粘贴到 model-HPO-configuration.json 文件中，然后根据需要编辑超参数即可。

> **Note**
>
> 如果您在 modelTraining API 中设置 modelName 参数，此外还更改 model-HPO-configuration.json 文件中的 model 和超参数规范，但它们不同，则 model-HPO-configuration.json 文件中的 model 值优先，modelName 值将被忽略。

- **task_type** – （字符串）由数据处理任务推理或直接传递给数据处理任务的机器学习任务类型（请勿修改）。有效值为：
  - "node_class"
  - "node_regression"
  - "link_prediction"

  数据处理任务通过检查导出的数据集和生成的训练任务配置文件中的数据集属性来推理任务类型。

  不应更改该值。如果要训练其它任务，则需要运行新的数据处理任务。如果 task_type 值不是您预期的值，则应检查数据处理任务的输入以确保它们正确无误。这包括 modelTraining API 的参数，以及数据导出过程生成的训练任务配置文件中的参数。
- **eval_metric** – （字符串）评估指标应用来评估模型性能和选择在 HPO 运行中表现最佳的模型。有效值为：
  - "acc" – 标准分类精度。这是单标签分类任务的默认设置，除非在数据处理过程中发现不平衡的标签，在这种情况下，默认值为 "F1"。

- "acc_topk"– 正确的标签出现在前 **k** 个预测中的次数。您也可以通过传入 topk 作为额外键来设置值 **k**。
- "F1" – F1 分数。
- "mse" – 回归任务的均方误差指标。
- "mrr" – 平均倒数排名指标。
- "precision"– 模型查准率，计算方法为真阳性与预测阳性的比率：= true-positives / (true-positives + false-positives)。
- "recall" – 模型查全率，计算方法为真阳性与实际阳性的比率：= true-positives / (true-positives + false-negatives)。
- "roc_auc" – ROC 曲线下方的区域。这是多标签分类的默认设置。

例如，要将指标更改为 F1，请按如下方式更改 eval_metric 值：

```
"  eval_metric": {
    "metric": "F1",
  },
```

或者，要将指标更改为 topk 精度分数，您可以按以下方式更改 eval_metric：

```
  "eval_metric": {
    "metric": "acc_topk",
    "topk": 2
  },
```

- **eval_frequency** –（对象）指定在训练期间应多久检查一次模型在验证集上的性能。根据验证性能，可以启动提前停止，并保存最佳模型。

  eval_frequency 对象包含两个元素，即 "type" 和 "value"。例如：

```
  "eval_frequency": {
    "type":  "evaluate_every_pct",
    "value":  0.1
  },
```

  有效的 type 值为：

  - **evaluate_every_pct** – 指定每次评估要完成的训练百分比。

对于 evaluate_every_pct，"value" 字段包含一个介于零和一之间的浮点数，用于表示该百分比。

- **evaluate_every_batch** – 指定每次评估要完成的训练批次数。

  对于 evaluate_every_batch，"value" 字段包含一个表示该批次计数的整数。

- **evaluate_every_epoch** – 指定每次评估的纪元数，其中新纪元从午夜开始。

  对于 evaluate_every_epoch，"value" 字段包含一个表示该纪元计数的整数。

eval_frequency 的默认设置是：

```
"eval_frequency": {
  "type":  "evaluate_every_epoch",
  "value":  1
},
```

- **1-tier-param** – （必需）第 1 层超参数的数组。

  如果您不想调整任何超参数，则可以将其设置为空数组。这不会影响 SageMaker 超参数调整任务启动的训练任务总数。这只是意味着所有训练任务（如果大于 1 但小于 10）都将使用相同的超参数集运行。

  另一方面，如果您想以同等的重要性对待所有可调整的超参数，您可以将所有超参数放在这个数组中。

- **2-tier-param** – （必需）第 2 层超参数的数组。

  仅当 maxHPONumberOfTrainingJobs 的值大于 10 时，才调整这些参数。否则，它们将固定为默认值。

  如果您的训练预算最多为 10 个训练任务，或者出于任何其它原因不想要第 2 层超参数，但您想调整所有可调整的超参数，则可以将其设置为空数组。

- **3-tier-param** – （必需）第 3 层超参数的数组。

  仅当 maxHPONumberOfTrainingJobs 的值大于 50 时，才调整这些参数。否则，它们将固定为默认值。

  如果您不想要第 3 层超参数，可以将其设置为空数组。

- **fixed-param** – (必需) 固定超参数的数组，这些超参数仅采用其默认值，且在不同的训练任务中没有变化。

  如果您想改变所有超参数，您可以将其设置为一个空数组，然后将 `maxHPONumberOfTrainingJobs` 的值设置为足够大以改变所有层，或者使所有超参数成为第 1 层。

表示 `1-tier-param`、`2-tier-param`、`3-tier-param` 和 `fixed-param` 中每个超参数的 JSON 对象包含以下元素：

- **param** – (字符串) 超参数的名称 (请勿更改)。

  请参阅 Neptune ML 中的有效超参数名称列表。

- **type** – (字符串) 超参数类型 (请勿更改)。

  有效类型为：`bool`、`int` 和 `float`。

- **default** – (字符串) 超参数的默认值。

  您可以设置新的默认值。

可调整超参数也可以包含以下元素：

- **range** – (数组) 连续可调整超参数的范围。

  这应该是一个包含两个值的数组，即范围的最小值和最大值 (`[min, max]`)。

- **options** – (数组) 分类可调整超参数的选项。

  此数组应包含所有需要考虑的选项：

  ```
  "options" : [value1, value2, ... valuen]
  ```

- **inc_strategy** – (字符串) 连续可调整超参数范围的增量更改的类型 (请勿更改)。

  有效值包括 `log`、`linear` 和 `power2`。仅当设置范围键时适用。

  修改此设置可能会导致无法使用超参数的完整范围进行调整。

- **inc_val** – (浮点数) 连续可调整超参数的连续增量差异量 (请勿更改)。

  仅当设置范围键时适用。

修改此设置可能会导致无法使用超参数的完整范围进行调整。

- **node_strategy** – (字符串) 表示此超参数的有效范围应根据图形中的节点数而变化 (请勿更改)。

  有效值为 "perM" (每百万)、"per10M" (每千万) 和 "per100M" (每 1 亿)。

  不更改此值，而应更改 range。

- **edge_strategy** – (字符串) 表示此超参数的有效范围应根据图形中的边缘数而变化 (请勿更改)。

  有效值为 "perM" (每百万)、"per10M" (每千万) 和 "per100M" (每 1 亿)。

  不更改此值，而应更改 range。

Neptune ML 中所有超参数的列表

以下列表包含可以在 Neptune ML 中的任何位置为任何模型类型和任务设置的所有超参数。由于它们并非全部适用于每种模型类型，因此务必仅在所用模型的模板中出现的 model-HPO-configuration.json 文件中设置超参数。

- **batch-size** – 一次向前传递中使用的目标节点批次的大小。类型：int。

  将其设置为更大的值可能会导致在 GPU 实例上进行训练时出现内存问题。

- **concat-node-embed** – 指示是否通过将节点的处理后特征与可学习的初始节点嵌入相连接来获得节点的初始表示形式，以改善模型的表现力。类型：bool。

- **dropout** – 应用于丢弃层的丢弃概率。类型：float。

- **edge-num-hidden**– 边缘特征模块的隐藏层大小或单位数量。仅在 use-edge-features 设置为 True 时使用。类型：浮点数。

- **enable-early-stop** – 切换是否使用提前停止特征。类型：bool。原定设置: true。

  使用此布尔参数可关闭提前停止特征。

- **fanout** – 在邻居采样期间要为目标节点采样的邻居数量。类型：int。

  此值与 num-layers 紧密耦合，应始终位于同一个超参数层中。这是因为您可以为每个潜在的 GNN 层指定扇出。

由于此超参数可能导致模型性能变化很大，因此应将其固定或设置为第 2 层或第 3 层超参数。将其设置为较大的值可能会导致在 GPU 实例上训练时出现内存问题。

- **gamma** – 分数函数中的边距值。类型：float。

  这仅适用于 KGE 链接预测模型。
- **l2norm** – 优化程序中使用的权重衰减值，该值对权重施加 L2 规范化惩罚。类型：bool。
- **layer-norm** – 表示是否对 rgcn 模型使用层规范化。类型：bool。
- **low-mem** – 指示是否以牺牲速度为代价使用关系消息传递函数的低内存实现。类型：bool。
- **lr** – 学习率。类型：float。

  应将其设置为第 1 层超参数。
- **neg-share** – 在链接预测中，表示正采样的边缘是否可以共享负边缘样本。类型：bool。
- **num-bases** – rgcn 模型中用于基本分解的基数。使用小于图形中边缘类型数量的 num-bases 值可以作为 rgcn 模型的正则化项。类型：int。
- **num-epochs** – 要运行的训练纪元数。类型：int。

  纪元是通过图形的完整训练过程。
- **num-hidden** – 隐藏的层大小或单位数量。类型：int。

  这还会设置无特征节点的初始嵌入大小。

  将其设置为大得多的值而不减小 batch-size，可能会导致在 GPU 实例上训练时出现内存不足问题。
- **num-layer** – 模型中 GNN 层的数量。类型：int。

  该值与扇出参数紧密耦合，应在同一个超参数层中设置扇出之后出现。

  由于这可能导致模型性能变化很大，因此应将其固定或设置为第 2 层或第 3 层超参数。
- **num-negs** – 在链接预测中，每个正样本的负样本数。类型：int。
- **per-feat-name-embed** – 表示是否在组合特征之前通过单独转换每个特征来嵌入每个特征。类型：bool。

  如果设置为 true，则每个节点的每个特征都将独立转换为固定的维度大小，然后将该节点的所有已转换特征串联起来，并进一步转换为 num_hidden 维度。

  如果设置为 false，则在不进行任何特定于特征的转换的情况下串联特征。

- **regularization-coef** – 在链接预测中，正则化损失的系数。类型：float。
- **rel-part** – 表示是否使用关系分区进行 KGE 链接预测。类型：bool。
- **sparse-lr** – 可学习节点嵌入的学习率。类型：float。

  可学习的初始节点嵌入用于没有特征或设置 concat-node-embed 时的节点。稀疏可学习节点嵌入层的参数使用单独的优化程序进行训练，该优化程序可以具有单独的学习速率。
- **use-class-weight** – 表示是否对不平衡的分类任务应用类别权重。如果设置为 true，则使用标签计数为每个类别标签设置权重。类型：bool。
- **use-edge-features** – 表示在消息传递过程中是否使用边缘特征。如果设置为 true，则将为具有特征的边缘类型向 RGCN 层添加自定义边缘特征模块。类型：bool。
- **use-self-loop**– 表示是否在训练 rgcn 模型时包含自循环。类型：bool。
- **window-for-early-stop** – 控制最新验证分数的平均值，以决定是否提前停止。默认值为 3。类型 = int。另请参阅 在 Neptune ML 中提前停止模型训练过程。类型：int。原定设置: 3。

  请参阅 。

在 Neptune ML 中自定义超参数

编辑 model-HPO-configuration.json 文件时，以下是最常见的更改类型：

- 编辑 range 超参数的最小值和/或最大值。
- 通过将超参数移到 fixed-param 部分并将其默认值设置为您想要的固定值，将其设置为固定值。
- 更改超参数的优先级，方法是将其置于特定层，编辑其范围，并确保正确设置其默认值。

模型训练最佳实践

您可以采取一些措施来提高 Neptune ML 模型的性能。

选择正确的节点属性

并非图形中的所有属性都可能有意义或与您的机器学习任务相关。在数据导出过程中，应排除任何不相关的属性。

以下是一些最佳实践：

- 借助领域专家来帮助评估特征的重要性以及使用它们进行预测的可行性。
- 移除您确定为冗余或不相关的特征，以减少数据中的噪音和不重要的相关性。
- 在构建模型时进行迭代。随心所欲地调整特征、特征组合和调整目标。

《Amazon 机器学习开发人员指南》中的特征处理提供了与 Neptune ML 相关的其它特征处理指南。

处理异常值数据点

异常值是指与其余数据明显不同的数据点。数据异常值可能会破坏或误导训练过程，从而导致训练时间延长或模型精度降低。除非异常值确实很重要，否则应在导出数据之前消除异常值。

移除重复的节点和边缘

存储在 Neptune 中的图形可能有重复的节点或边缘。这些冗余元素将为 ML 模型训练引入噪声。在导出数据之前，请消除重复的节点或边缘。

调整图形结构

导出图形时，您可以更改处理特征的方式和构造图形的方式，以提高模型性能。

以下是一些最佳实践：

- 当边缘属性具有边缘类别的含义时，在某些情况下值得将其转换为边缘类型。
- 用于数值属性的默认规范化策略是 `min-max`，但在某些情况下，其它规范化策略效果更好。您可以预处理该属性并更改规范化策略，如model-HPO-configuration.json 文件的元素中所述。
- 导出过程会根据属性类型自动生成特征类型。例如，它将 `String` 属性视为类别特征，并将 `Float` 和 `Int` 属性视为数值特征。如果需要，可以在导出后修改特征类型（请参阅model-HPO-configuration.json 文件的元素）。

调整超参数范围和默认值

数据处理操作从图形中推理超参数配置范围。如果生成的模型超参数范围和默认值不很适用于您的图形数据，则可以编辑 HPO 配置文件以创建自己的超参数调整策略。

以下是一些最佳实践：

- 当图形变大时，默认的隐藏维度大小可能不足以包含所有信息。您可以更改 `num-hidden` 超参数来控制隐藏的维度大小。
- 对于知识图谱嵌入 (KGE) 模型，您可能需要根据图形结构和预算更改正在使用的特定模型。

  `TrainsE` 模型难以处理一对多 (1-N)、多对一 (N-1) 和多对多 (N-N) 关系。`DistMult` 模型难以处理对称关系。`RotatE` 擅长对各种关系进行建模，但在训练期间比 `TrainsE` 和 `DistMult` 更昂贵。
- 在某些情况下，当节点标识和节点特征信息都很重要时，您应该使用 `concat-node-embed` 告诉 Neptune ML 模型，通过将节点的特征与其初始嵌入项串联来获得节点的初始表示形式。
- 当您在某些超参数上获得相当不错的性能时，您可以根据这些结果调整超参数搜索空间。

在 Neptune ML 中提前停止模型训练过程

提前停止可以在不降低模型性能的情况下，显著减少模型训练的运行时间和关联成本。它还可以防止模型对训练数据进行过度拟合。

提前停止取决于对验证集性能的定期测量。最初，性能会随着训练进行而提高，但是当模型开始过度拟合时，它再次开始下降。提前停止特征可识别模型开始过度拟合的点并在该点停止模型训练。

Neptune ML 监控验证指标调用，并将最新的验证指标与上次 **n** 评估中验证指标的平均值进行比较，其中 **n** 是使用 `window-for-early-stop` 参数设置的数字。一旦验证指标比该平均值差，Neptune ML 就会停止模型训练并保存迄今为止最好的模型。

您可以使用以下参数控制提前停止：

- **`window-for-early-stop`** – 此参数的值是一个整数，用于指定在决定提前停止时要求取平均值的最近验证分数的数量。默认值为 `3`。
- **`enable-early-stop`** – 使用此布尔参数可关闭提前停止特征。默认情况下，其值为 `true`。

在 Neptune ML 中提前停止 HPO 过程

Neptune ML 中的提前停止特征还可以使用 SageMaker HPO 热启动特征停止与其它训练任务相比性能不佳的训练。这也可以降低成本并提高 HPO 的质量。

有关其工作原理的描述，请参阅运行热启动超参数调整任务。

热启动提供了将从以前的训练工作中学习到的信息传递给后续训练任务的能力，并具有两个明显的好处：

- 首先，使用先前训练任务的结果来选择超参数的良好组合，以在新的调整任务中进行搜索。
- 其次，它允许提前停止以访问更多的模型运行，从而缩短调整时间。

此特征在 Neptune ML 中自动启用，允许您在模型训练时间和性能之间取得平衡。如果您对当前模型的性能感到满意，则可以使用该模型。否则，您会运行更多的 HPO，这些 HPO 使用先前运行的结果进行热启动，以便发现更好的模型。

获取专业支持服务

AWS 提供专业支持服务，以帮助您解决在 Neptune 项目上进行机器学习时遇到的问题。如果您遇到困难，请联系 AWS Support。

使用经过训练的模型生成新的模型构件

使用 Neptune ML 模型转换命令，您可以使用预训练的模型参数计算模型构件，例如已处理的图形数据上的节点嵌入。

用于增量推理的模型转换

在增量模型推理工作流程中，在处理完从 Neptune 导出的已更新的图形数据后，可以使用 curl（或 awscurl）命令启动模型转换任务，如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "mlModelTrainingJobId": "(the ML model training job-id)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
      }'
```

然后，您可以将此任务的 ID 传递给 create-endpoints API 调用，以创建新的端点，或者使用此任务生成的新模型构件更新现有端点。这允许新的或更新的端点为更新后的图形数据提供模型预测。

适用于任何训练任务的模型转换

还可以提供一个 `trainingJobName` 参数，为在 Neptune ML 模型训练期间启动的任何 SageMaker 训练任务生成模型构件。由于 Neptune ML 模型训练任务可能会潜在启动许多 SageMaker 训练任务，因此您可以灵活地基于任何 SageMaker 训练任务创建推理端点。

例如：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
```

```
    }'
```

如果原始训练任务是针对用户提供的自定义模型，则在调用模型转换时必须包含一个 `customModelTransformParameters` 对象。有关如何实现和使用自定义模型的信息，请参阅Neptune ML 中的自定义模型。

> **Note**
>
> `modeltransform` 命令始终在最适合该训练的 SageMaker 训练任务上运行模型转换。

有关模型转换任务的更多信息，请参阅modeltransform 命令。

在 Neptune ML 中通过模型训练产生的构件

模型训练结束后，Neptune ML 使用经训练的最佳模型参数来生成启动推理端点和提供模型预测所需的模型构件。这些构件由训练任务打包并存储在最佳 SageMaker 训练任务的 Amazon S3 输出位置。

以下各节描述了各种任务的模型构件中包含的内容，以及模型转换命令如何使用预先存在的训练模型生成构件，即使是在新的图形数据上也是如此。

为不同的任务生成的构件

训练过程生成的模型构件的内容取决于目标机器学习任务：

- 节点分类和回归 – 对于节点属性预测，构件包括模型参数、来自 GNN 编码器的节点嵌入、训练图形中节点的模型预测以及推理端点的一些配置文件。在节点分类和节点回归任务中，预先计算训练期间存在的节点的模型预测，以减少查询延迟。
- 边缘分类和回归 – 对于边缘属性预测，构件还包括模型参数和节点嵌入。模型解码器的参数对于推理尤其重要，因为我们通过将模型解码器应用于边缘的源顶点和目标顶点的嵌入，来计算边缘分类或边缘回归预测。
- 链接预测 - 对于链接预测，除了为边缘属性预测生成的构件外，DGL 图形还作为构件包含在内，因为链接预测需要训练图形才能执行预测。链接预测的目标是预测可能与源顶点组合以在图形中形成特定类型的边缘的目标顶点。为此，将源顶点的节点嵌入和边缘类型的学习表示形式与所有可能的目标顶点的节点嵌入相结合，以便为每个目标顶点生成边缘似然性分数。然后对分数进行排序，以对潜在的目标顶点进行排名并返回排名靠前的候选顶点。

对于每种任务类型，来自 DGL 的图形神经网络模型权重都保存在模型构件中。这允许 Neptune ML 在图形变化时计算新的模型输出（归纳推理），此外，还可以使用预先计算的预测和嵌入（转导推理）来减少延迟。

生成新的模型构件

在 Neptune ML 中训练模型后生成的模型构件与训练过程直接相关。这意味着预先计算的嵌入和预测仅对原始训练图形中的实体存在。尽管 Neptune ML 端点的归纳推理模式可以实时计算新实体的预测，但您可能需要在不查询端点的情况下对新实体生成批量预测。

为了获得已添加到图形中的新实体的批量模型预测，需要为新的图形数据重新计算新的模型构件。这是使用 `modeltransform` 命令完成的。当您只想在不设置端点的情况下进行批量预测，或者您想生成所有预测以便将它们写回到图形时，您可以使用 `modeltransform` 命令。

由于模型训练在训练过程结束时隐式执行模型转换，因此，训练任务总是根据训练图形数据重新计算模型构件。但是，`modeltransform` 命令还可以在未用于训练模型的图形数据上计算模型构件。为此，必须使用与原始图形数据相同的特征编码来处理新的图形数据，并且必须遵循相同的图形架构。

为此，您可以先创建一个新的数据处理任务，该任务是在原始训练图形数据上运行的数据处理任务的克隆，然后对新的图形数据运行该任务（请参阅处理 Neptune ML 的更新图形数据）。然后，使用新的 `dataProcessingJobId` 和旧的 `modelTrainingJobId` 调用 `modeltransform` 命令，以针对更新后的图形数据重新计算模型构件。

对于节点属性预测，节点嵌入和预测是在新的图形数据上重新计算的，甚至对于原始训练图形中存在的节点也是如此。

对于边缘属性预测和链接预测，还会重新计算节点嵌入，并类似地覆盖任何现有的节点嵌入。为了重新计算节点嵌入，Neptune ML 将从先前训练的模型中学习到的 GNN 编码器应用于具有新特征的新图形数据的节点。

对于没有特征的节点，可以重用从原始模型训练中学习到的初始表示形式。对于没有特征且原始训练图形中不存在的新节点，Neptune ML 将其表示形式初始化为原始训练图形中存在的该节点类型的已学习初始节点表示形式的平均值。如果您有许多没有特征的新节点，这可能会导致模型预测的性能有所下降，因为它们都将初始化为该节点类型的平均初始嵌入。

如果您的模型在 `concat-node-embed` 设置为 true 的情况下进行训练，则初始节点表示形式是通过将节点特征与可学习的初始表示形式串联起来创建的。因此，对于更新后的图形，新节点的初始节点表示形式还使用平均初始节点嵌入，并与新的节点特征串联。

此外，目前不支持删除节点。如果在更新后的图形中移除了节点，则必须根据更新后的图形数据重新训练模型。

重新计算模型构件在新的图形上重用学习到的模型参数，并且只有在新图形与旧图形非常相似时才应这样做。如果您的新图形不够相似，则需要重新训练模型，以便在新图形数据上获得相似的模型性能。什么构成足够相似取决于图形数据的结构，但根据经验，如果您的新数据与原始训练图形数据相差超过 10-20%，则应重新训练模型。

对于所有节点都有特征的图形，阈值的高限（相差 20%）适用，但是对于许多节点没有特征而添加到图形中的新节点没有属性的图形，则阈值的低限（相差 10%）甚至可能过高。

有关模型转换任务的更多信息，请参阅modeltransform 命令。

Neptune ML 中的自定义模型

Neptune ML 允许您使用 Python 定义自己的自定义模型实现。您可以使用 Neptune ML 基础设施训练和部署自定义模型，就像对内置模型所做的那样，并使用它们通过图形查询获得预测。

> **Note**
>
> 自定义模型目前不支持实时归纳推理。

您可以按照 Neptune ML 工具包示例以及使用 Neptune ML 工具包中提供的模型组件，开始在 Python 中实现自己的自定义模型。以下各节提供了更多详细信息。

目录

Neptune ML 中自定义模型的概述

何时在 Neptune ML 中使用自定义模型

Neptune ML 的内置模型可处理 Neptune ML 支持的所有标准任务，但在某些情况下，您可能希望对特定任务的模型进行更精细的控制，或者需要自定义模型训练过程。例如，自定义模型适用于以下情况：

- 非常大的文本模型的文本特征的特征编码需要在 GPU 上运行。
- 您想使用自己在深度图表库 (DGL) 中开发的自定义图形神经网络 (GNN) 模型。
- 您想使用表格模型或集合模型进行节点分类和回归。

在 Neptune ML 中开发和使用自定义模型的工作流程

Neptune ML 中的自定义模型支持旨在无缝集成到现有的 Neptune ML 工作流程中。它的工作原理是在 Neptune ML 基础设施上的源模块中运行自定义代码来训练模型。与内置模式一样，Neptune ML 会自动启动 SageMaker 超参数调整任务，并根据评估指标选择最佳模型。然后，它使用源模块中提供的实现来生成模型构件以进行部署。

自定义模型的数据导出、训练配置和数据预处理与内置模型相同。

在数据预处理之后，您可以使用 Python 以迭代和交互方式开发和测试您的自定义模型实现。当您的模型可以投入生产时，您可以将生成的 Python 模块上传到 Amazon S3，如下所示：

```
aws s3 cp --recursive (source path to module) s3://(bucket name)/(destination path for
 your module)
```

然后，您可以使用普通的默认数据或增量数据工作流程将模型部署到生产环境中，但有一些区别。

要使用自定义模型进行模型训练，必须向 Neptune ML 模型训练 API 提供 `customModelTrainingParameters` JSON 对象，以确保使用您的自定义代码。`customModelTrainingParameters` 对象中的字段如下所示：

- **`sourceS3DirectoryPath`** – （必需）实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 `model-hpo-configuration.json` 文件。
- **`trainingEntryPointScript`** – （可选）执行模型训练并将超参数作为命令行参数（包括固定的超参数）的脚本模块中入口点的名称。

原定设置: `training.py`。

- **`transformEntryPointScript`** – （可选）脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。

  原定设置: `transform.py`。

例如：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

同样，要启用自定义模型转换，您必须向 Neptune ML 模型转换 API 提供一个 `customModelTransformParameters` JSON 对象，其字段值必须与训练任务中保存的模型参数兼容。`customModelTransformParameters` 对象包含以下字段：

- **`sourceS3DirectoryPath`** – （必需）实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 `model-hpo-configuration.json` 文件。
- **`transformEntryPointScript`** – （可选）脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。

原定设置: transform.py。

例如：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
        "customModelTransformParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

在 Neptune ML 中开发自定义模型

开始开发自定义模型的一个好方法是按照 Neptune ML 工具包示例来构造和编写训练模块。Neptune ML 工具包还在 modelzoo 中实现了模块化图形 ML 模型组件，您可以将其堆叠并用于创建自定义模型。

此外，该工具包还提供实用程序函数，帮助您在模型训练和模型转换期间生成必要的构件。您可以在您的自定义实现中导入这个 Python 软件包。该工具包中提供的任何功能或模块也可在 Neptune ML 训练环境中使用。

如果您的 Python 模块具有其它外部依赖项，则可以通过在模块的目录中创建 `requirements.txt` 文件来包含这些额外的依赖项。然后，将在运行训练脚本之前安装 `requirements.txt` 文件中列出的软件包。

实现您的自定义模型的 Python 模块至少需要包含以下内容：

- 训练脚本入口点
- 转换脚本入口点
- 一个 `model-hpo-configuration.json` 文件

在 Neptune ML 中开发自定义模型训练脚本

自定义模型训练脚本应该是可执行的 Python 脚本，就像 Neptune ML 工具包的 `train.py` 示例一样。它必须接受超参数名称和值作为命令行参数。在模型训练期间，超参数名称是从 `model-hpo-configuration.json` 文件中获取的。如果超参数是可调整的，则超参数值在有效的超参数范围内，或者，如果超参数值不可调整，则采用默认的超参数值。

训练脚本使用如下语法在 SageMaker 训练实例上运行：

```
python3 (script entry point) --(1st parameter) (1st value) --(2nd parameter) (2nd
 value) (...)
```

对于所有任务，除了您指定的超参数外，Neptune ML AutoTrainer 还会向您的训练脚本发送几个必需的参数，并且您的脚本必须能够处理这些额外参数才能正常工作。

这些额外的必需参数因任务而稍有差异：

对于节点分类或节点回归

- **task** – Neptune ML 在内部使用的任务类型。对于节点分类，这是 node_class；对于节点回归，这是 node_regression。
- **model** – Neptune ML 在内部使用的模型名称，在本例中为 custom。
- **name** – Neptune ML 在内部使用的任务名称，在本例中，node_class-custom 用于节点分类，node_regression-custom 用于节点回归。
- **target_ntype** – 用于分类或回归的节点类型的名称。
- **property** – 用于分类或回归的节点属性的名称。

对于链接预测

- **task** – Neptune ML 在内部使用的任务类型。对于链接预测，这是 link_predict。
- **model** – Neptune ML 在内部使用的模型名称，在本例中为 custom。
- **name** – Neptune ML 在内部使用的任务名称，在本例中为 link_predict-custom。

对于边缘分类或边缘回归

- **task** – Neptune ML 在内部使用的任务类型。对于边缘分类，这是 edge_class；对于边缘回归，这是 edge_regression。
- **model** – Neptune ML 在内部使用的模型名称，在本例中为 custom。
- **name** – Neptune ML 在内部使用的任务名称，在本例中，edge_class-custom 用于边缘分类，edge_regression-custom 用于边缘回归。
- **target_etype** – 用于分类或回归的边缘类型的名称。
- **property** – 用于分类或回归的边缘属性的名称。

您的脚本应保存模型参数以及训练结束时所需的任何其它构件。

您可以使用 Neptune ML 工具包实用程序函数来确定处理后的图形数据的位置、模型参数应保存的位置以及训练实例上有哪些 GPU 设备可用。有关如何使用这些实用程序函数的示例，请参阅 train.py 示例训练脚本。

在 Neptune ML 中开发自定义模型转换脚本

需要一个转换脚本来利用 Neptune ML 增量工作流程在不断演变的图形上进行模型推理，而无需重新训练模型。即使模型部署所需的所有构件都是由训练脚本生成的，但如果您想在不重新训练模型的情况下生成更新的模型，您仍然需要提供转换脚本。

> **Note**
>
> 自定义模型目前不支持实时归纳推理。

自定义模型转换脚本应该是可执行的 Python 脚本，就像 Neptune ML 工具包的 transform.py 示例脚本一样。由于此脚本是在模型训练期间调用的，没有命令行参数，因此该脚本接受的任何命令行参数都必须具有默认值。

该脚本在 SageMaker 训练实例上运行，其语法如下所示：

```
python3 (your transform script entry point)
```

您的转换脚本需要各种信息，例如：

- 处理的图形数据的位置。
- 保存模型参数的位置以及应保存新模型构件的位置。
- 实例上可用的设备。
- 生成了最佳模型的超参数。

这些输入是使用脚本可以调用的 Neptune ML 实用程序函数获得的。有关如何执行此操作的示例，请参阅该工具包的示例 transform.py 脚本。

该脚本应保存节点嵌入、节点 ID 映射以及每个任务的模型部署所需的任何其它构件。有关不同 Neptune ML 任务所需的模型构件的更多信息，请参阅模型构件文档。

Neptune ML 中的自定义 `model-hpo-configuration.json` 文件

`model-hpo-configuration.json` 文件为您的自定义模型定义超参数。它的格式与 Neptune ML 内置模型中使用的 `model-hpo-configuration.json` 文件格式相同，并且优先于由 Neptune ML 自动生成并上传到处理过的数据所在位置的版本。

向模型添加新的超参数时，还必须在此文件中为该超参数添加一个条目，以便将超参数传递给您的训练脚本。

如果您希望超参数可调整，则必须为其提供一个范围，并将其设置为 `tier-1`、`tier-2` 或 `tier-3` 参数。如果配置的训练任务总数允许在其层中调整超参数，则将调整超参数。对于不可调整的参数，必须提供默认值并将超参数添加到文件的 `fixed-param` 部分。有关如何执行此操作的示例，请参阅工具包的示例 `model-hpo-configuration.json` 文件。

您还必须提供指标定义，SageMaker 超参数优化任务将使用该定义来评估经过训练的候选模型。为此，您需要向 `model-hpo-configuration.json` 文件中添加一个 `eval_metric` JSON 对象，如下所示：

```
"eval_metric": {
  "tuning_objective": {
      "MetricName": "(metric_name)",
      "Type": "Maximize"
  },
  "metric_definitions": [
    {
      "Name": "(metric_name)",
      "Regex": "(metric regular expression)"
    }
  ]
},
```

`eval_metric` 对象中的 `metric_definitions` 数组列出了您希望 SageMaker 从训练实例中提取的每个指标的指标定义对象。每个指标定义对象都有一个 `Name` 键，它允许您为指标提供名称（例如“accuracy”、“f1”等）。`Regex` 键允许您提供与该特定指标在训练日志中的输出方式相匹配的正则表达式字符串。有关如何定义指标的更多详细信息，请参阅 SageMaker 超参数调整页面。

然后，`eval_metric` 中的 `tuning_objective` 对象允许您指定应将 `metric_definitions` 中的哪些指标用作评估指标（作为超参数优化的目标指标）。`MetricName` 的值必须与 `metric_definitions` 中定义之一的 `Name` 的值相匹配。`Type` 的值应为“Maximize”或“Minimize”，具体取决于该指标应解释为越大越好（如“accuracy”），还是应解释为越小越好（例如“mean-squared-error”）。

`model-hpo-configuration.json` 文件的此部分中的错误可能导致 Neptune ML 模型训练 API 任务失败，因为 SageMaker 超参数调整任务将无法选择最佳模型。

在 Neptune ML 中对您的自定义模型实现进行本地测试

您可以使用 Neptune ML 工具包 Conda 环境在本地运行代码，以便测试和验证您的模型。如果您在 Neptune 笔记本实例上进行开发，那么这个 Conda 环境将预安装在 Neptune 笔记本实例上。如果您在其它实例上进行开发，则需要按照 Neptune ML 工具包中的本地设置说明进行操作。

当您调用模型训练 API 时，Conda 环境可以准确地重现模型的运行环境。所有示例训练脚本和转换脚本都允许您传递命令行 `--local` 标志，以便在本地环境中运行脚本来便于调试。在开发自己的模型时，这是一种很好的做法，因为它允许您以交互方式和迭代方式测试模型实现。在 Neptune ML 生产训练环境中进行模型训练期间，忽略此参数。

创建要查询的推理端点

推理端点允许您查询模型训练过程构造的一个特定模型。端点附加到训练过程能够生成的给定类型中性能最佳的模型。然后，端点能够接受来自 Neptune 的 Gremlin 查询，并返回该模型对查询中输入的预测。创建推理端点后，它会一直处于活动状态，直到您将其删除。

管理 Neptune ML 的推理端点

对从 Neptune 导出的数据完成模型训练后，可以使用 `curl`（或 `awscurl`）命令创建推理端点，如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

您也可以通过已完成的模型转换任务创建的模型创建推理端点，方法大致相同：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTransformJobId": "(the model-transform job-id of a completed job)"
      }'
```

endpoints 命令中解释了如何使用这些命令的详细信息，以及有关如何获取端点状态、如何删除端点以及如何列出所有推理端点的信息。

Neptune ML 中的推理查询

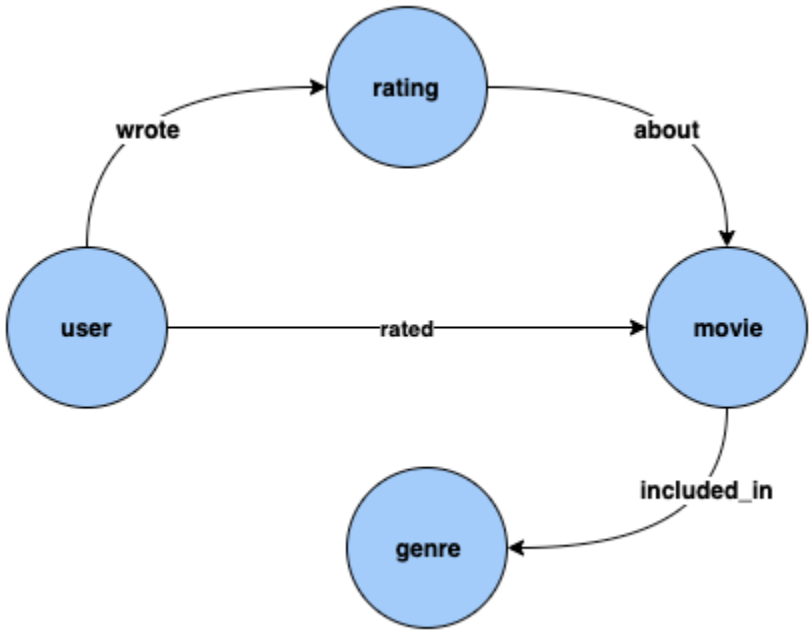
您可以使用 Gremlin 或 SPARQL 来查询 Neptune ML 推理端点。但是，目前仅 Gremlin 查询支持实时归纳推理。

Neptune ML 中的 Gremlin 推理查询

如Neptune ML 功能中所述，Neptune ML 支持可以执行以下类型的推理任务的训练模型：

- 节点分类 - 预测顶点属性的分类特征。
- 节点回归 - 预测顶点的数值属性。
- 边缘分类 - 预测边缘属性的分类特征。
- 边缘回归 - 预测边缘的数值属性。
- 链接预测 - 根据源节点和传出边缘预测目标节点，或者根据目标节点和传入边缘预测源节点。

我们可以借助使用 MovieLens 100k 数据集（由 GroupLens Research 提供）的示例来说明这些不同的任务。这个数据集由电影、用户和用户对电影的评分组成，我们从中创建了如下所示的属性图：



节点分类：在上面的数据集中，Genre 是一种通过 included_in 边缘与顶点类型 Movie 相连的顶点类型。但是，如果我们调整数据集以使 Genre 成为顶点类型 Movie 的分类特征，则可以使用节点分类模型来解决对于添加到知识图谱中的新电影推理 Genre 的问题。

节点回归：如果我们考虑具有如 timestamp 和 score 等属性的顶点类型 Rating，则可以使用节点回归模型来解决对于 Rating 推理数值 Score 的问题。

边缘分类：同样，对于 Rated 边缘，如果我们有一个属性 Scale 可以具有值（Love、Like、Dislike、Neutral、Hate）之一，就可以使用边缘分类模型来解决对于新电影/评分的 Rated 边缘推理 Scale 的问题。

边缘回归：同样，对于同一 `Rated` 边缘，如果我们有一个属性 `Score` 可以保存评分的数值，那么可以从边缘回归模型中推理出来。

链接预测：诸如查找最有可能对给定电影进行评分的前十名用户，或者找到给定用户最有可能评分的前十部电影之类的问题，属于链接预测的范围。

> **Note**
>
> 对于 Neptune ML 用例，我们有一套非常丰富的笔记本，旨在让您亲身了解每个用例。当您使用 Neptune ML AWS CloudFormation 模板创建 Neptune ML 集群时，您可以将这些笔记本与 Neptune 集群一起创建。这些笔记本也可以在 github 上找到。

主题

Gremlin 推理查询中使用的 Neptune ML 谓词

`Neptune#ml.deterministic`

此谓词是归纳推理查询的选项，也就是说，适用于包含 `Neptune#ml.inductiveInference` 谓词的查询。

使用归纳推理时，Neptune 引擎会创建相应的子图形来评估经过训练的 GNN 模型，该子图形的要求取决于最终模型的参数。具体而言，`num-layer` 参数确定从目标节点或边缘开始的遍历跳数，而 `fanouts` 参数指定在每跳处要对多少邻居进行采样（请参阅 HPO 参数）。

默认情况下，归纳推理查询在非确定性模式下运行，在这种模式下，Neptune 会随机构建邻域。在进行预测时，这种正常的随机邻居采样有时会产生不同的预测。

当您在归纳推理查询中包含 Neptune#ml.deterministic 时，Neptune 引擎尝试以确定性的方式对邻居进行采样，这样对同一查询的多次调用每次都会返回相同的结果。但是，不能保证结果是完全确定的，因为分布式系统的底层图形和构件的变化仍然会带来波动。

您可以在查询中加入 Neptune#ml.deterministic 谓词，如下所示：

```
.with("Neptune#ml.deterministic")
```

如果 Neptune#ml.deterministic 谓词包含在也不包含 Neptune#ml.inductiveInference 的查询中，则直接将其忽略。

Neptune#ml.disableInductiveInferenceMetadataCache

此谓词是归纳推理查询的选项，也就是说，适用于包含 Neptune#ml.inductiveInference 谓词的查询。

对于归纳推理查询，Neptune 使用存储在 Amazon S3 中的元数据文件来决定构建邻域时的跳数和扇出。Neptune 通常会缓存此模型的元数据，以避免重复从 Amazon S3 获取文件。可以通过在查询中包含 Neptune#ml.disableInductiveInferenceMetadataCache 谓词来禁用缓存。尽管 Neptune 直接从 Amazon S3 获取元数据可能会比较慢，但当重新训练或转换后更新了 SageMaker 端点并且缓存已过时时，这会很有用。

您可以在查询中加入 Neptune#ml.disableInductiveInferenceMetadataCache 谓词，如下所示：

```
.with("Neptune#ml.disableInductiveInferenceMetadataCache")
```

以下是 Jupyter 笔记本中示例查询的样子：

```
%%gremlin
g.with("Neptune#ml.endpoint", "ep1")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .with("Neptune#ml.disableInductiveInferenceMetadataCache")
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

Neptune#ml.endpoint

必要时，在 with() 步骤中使用 Neptune#ml.endpoint 谓词来指定推理端点：

```
  .with("Neptune#ml.endpoint", "the model's SageMaker inference endpoint")
```

您可以通过端点的 id 或其 URL 来识别端点。例如：

```
  .with( "Neptune#ml.endpoint", "node-classification-movie-lens-endpoint" )
```

或者：

```
  .with( "Neptune#ml.endpoint", "https://runtime.sagemaker.us-east-1.amazonaws.com/
endpoints/node-classification-movie-lens-endpoint/invocations" )
```

> **Note**
>
> 如果您将 Neptune 数据库集群参数组中的 neptune_ml_endpoint 参数设置为端点 id 或 URL，则无需在每个查询中都包含 Neptune#ml.endpoint 谓词。

Neptune#ml.iamRoleArn

在 with() 步骤中使用 Neptune#ml.iamRoleArn 来指定 SageMaker 执行 IAM 角色的 ARN（如有必要）：

```
  .with("Neptune#ml.iamRoleArn", "the ARN for the SageMaker execution IAM role")
```

有关如何创建 SageMaker 执行 IAM 角色的信息，请参阅创建自定义 NeptuneSageMakerIAMRole 角色。

> **Note**
>
> 如果您将 Neptune 数据库集群参数组中的 neptune_ml_iam_role 参数设置为 SageMaker 执行 IAM 角色的 ARN，则无需在每个查询中都包含 Neptune#ml.iamRoleArn 谓词。

Neptune#ml.inductiveInference

Gremlin 中已默认启用转导推理。要进行实时归纳推理查询，请加入 Neptune#ml.inductiveInference 谓词，如下所示：

```
.with("Neptune#ml.inductiveInference")
```

如果您的图形是动态的，则归纳推理通常是最佳选择；但如果图形是静态的，则转导推理会更快、更高效。

Neptune#ml.limit

Neptune#ml.limit 谓词可以选择限制每个实体返回的结果数：

```
.with( "Neptune#ml.limit", 2 )
```

默认情况下，限制为 1，可设置的最大数量为 100。

Neptune#ml.threshold

Neptune#ml.threshold 谓词可以选择为结果分数设定截止阈值：

```
.with( "Neptune#ml.threshold", 0.5D )
```

这使您可以丢弃分数低于指定阈值的所有结果。

Neptune#ml.classification

Neptune#ml.classification 谓词附加到 properties() 步骤，以确定需要从节点分类模型的 SageMaker 端点获取属性：

```
.properties( "property key of the node classification
model" ).with( "Neptune#ml.classification" )
```

Neptune#ml.regression

Neptune#ml.regression 谓词附加到 properties() 步骤，以确定需要从节点回归模型的 SageMaker 端点获取属性：

```
.properties( "property key of the node regression
model" ).with( "Neptune#ml.regression" )
```

Neptune#ml.prediction

Neptune#ml.prediction 谓词附加到 in() 和 out() 步骤，以确定这是一个链接预测查询：

```
  .in("edge label of the link prediction
  model").with("Neptune#ml.prediction").hasLabel("target node label")
```

Neptune#ml.score

在 Gremlin 节点或边缘分类查询中使用 Neptune#ml.score 谓词来获取机器学习置信度分数。在 properties() 步骤中应将 Neptune#ml.score 谓词与查询谓词一起传递，以获得节点或边缘分类查询的 ML 置信度分数。

您可以找到一个包含其它节点分类示例的节点分类示例，以及边缘分类部分中的一个边缘分类示例。

Neptune ML 中的 Gremlin 节点分类查询

对于 Neptune ML 中的 Gremlin 节点分类：

- 模型是在顶点的一个属性上训练的。此属性的唯一值集合称为一组节点类，或者简称为类。
- 可以从节点分类模型中推理顶点属性的节点类或分类属性值。当此属性尚未附加到顶点时，这很有用。
- 要从节点分类模型中获取一个或多个类，您需要使用带有谓词 Neptune#ml.classification 的 with() 步骤来配置 properties() 步骤。如果这些是顶点属性，则输出格式与您期望的格式类似。

> **Note**
>
> 节点分类仅适用于字符串属性值。这意味着不支持诸如 0 或 1 之类的数值属性值，但支持等同的字符串 "0" 和 "1"。同样，布尔属性值 true 和 false 不起作用，但 "true" 和 "false" 起作用。

以下是一个示例节点分类查询：

```
g.with( "Neptune#ml.endpoint","node-classification-movie-lens-endpoint" )
 .with( "Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role" )
 .with( "Neptune#ml.limit", 2 )
 .with( "Neptune#ml.threshold", 0.5D )
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
```

此查询的输出如下所示：

```
==>vp[genre->Action]
==>vp[genre->Crime]
==>vp[genre->Comedy]
```

在上面的查询中，V() 和 properties() 步骤的使用方式如下：

V() 步骤包含要从节点分类模型中获取类的一组顶点：

```
  .V( "movie_1", "movie_2", "movie_3" )
```

properties() 步骤包含训练模型所依据的键，并具有 .with("Neptune#ml.classification") 以表明这是节点分类机器学习推理查询。

目前不支持在 properties().with("Neptune#ml.classification") 步骤中使用多个属性键。例如，以下查询会导致异常：

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre", "other_label").with("Neptune#ml.classification")
```

有关具体的错误消息，请参阅 Neptune ML 异常列表。

properties().with("Neptune#ml.classification") 步骤可以与以下任何步骤结合使用：

- value()
- value().is()
- hasValue()
- has(value,"")
- key()
- key().is()
- hasKey()
- has(key,"")
- path()

其它节点分类查询

如果推理端点和相应的 IAM 角色均已保存在数据库集群参数组中，则节点分类查询可以像这样简单：

```
g.V("movie_1", "movie_2",
 "movie_3").properties("genre").with("Neptune#ml.classification")
```

您可以使用 union() 步骤在查询中混用顶点属性和类：

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .union(
   properties("genre").with("Neptune#ml.classification"),
   properties("genre")
 )
```

您也可以进行无界查询，如下所示：

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V()
 .properties("genre").with("Neptune#ml.classification")
```

您可以使用 select() 步骤以及 as() 步骤一起检索节点类和顶点：

```
g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" ).as("vertex")
 .properties("genre").with("Neptune#ml.classification").as("properties")
 .select("vertex","properties")
```

还可以按节点类进行筛选，如以下示例所示：

```
g.with("Neptune#ml.endpoint", "node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
 .has(value, "Horror")

g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
```

```
  .has(value, P.eq("Action"))

g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre").with("Neptune#ml.classification")
 .has(value, P.within("Action", "Horror"))
```

您可以使用 Neptune#ml.score 谓词获得节点分类置信度分数：

```
 g.with("Neptune#ml.endpoint","node-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V( "movie_1", "movie_2", "movie_3" )
 .properties("genre", "Neptune#ml.score").with("Neptune#ml.classification")
```

响应将如下所示：

```
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.01234567]
==>vp[genre->Crime]
==>vp[Neptune#ml.score->0.543210]
==>vp[genre->Comedy]
==>vp[Neptune#ml.score->0.10101]
```

在节点分类查询中使用归纳推理

假设您要在 Jupyter 笔记本的现有图形中添加一个新节点，如下所示：

```
%%gremlin
g.addV('label1').property(id,'101').as('newV')
 .V('1').as('oldV1')
 .V('2').as('oldV2')
 .addE('eLabel1').from('newV').to('oldV1')
 .addE('eLabel2').from('oldV2').to('newV')
```

然后，您可以使用归纳推理查询来获得反映了新节点的流派和置信度分数：

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("genre", "Neptune#ml.score")
```

```
  .with("Neptune#ml.classification")
  .with("Neptune#ml.inductiveInference")
```

但是，如果您多次运行此查询，您得到的结果可能会有所不同：

```
# First time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.21365921]
```

您可以将同样的查询设定为确定性：

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
  .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
  .V('101').properties("genre", "Neptune#ml.score")
  .with("Neptune#ml.classification")
  .with("Neptune#ml.inductiveInference")
  .with("Neptune#ml.deterministic")
```

在这种情况下，每次的结果都大致相同：

```
# First time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]
# Second time
==>vp[genre->Action]
==>vp[Neptune#ml.score->0.12345678]
```

Neptune ML 中的 Gremlin 节点回归查询

节点回归与节点分类类似，不同之处在于从回归模型中为每个节点推理的值都是数值。除了以下区别之外，您可以对节点回归使用与节点分类相同的 Gremlin 查询：

- 同样，在 Neptune ML 中，节点指的是顶点。
- `properties()` 步骤采用形式 `properties().with("Neptune#ml.regression")`，而不是 `properties().with("Neptune#ml.classification")`。

- "Neptune#ml.limit" 和 "Neptune#ml.threshold" 谓词不适用。
- 对值进行筛选时，必须指定一个数值。

以下是一个示例顶点分类查询：

```
g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
```

您可以使用回归模型对推理的值进行筛选，如以下示例所示：

```
g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
 .value().is(P.gte(1600000))

g.with("Neptune#ml.endpoint","node-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1","movie_2","movie_3")
 .properties("revenue").with("Neptune#ml.regression")
 .hasValue(P.lte(1600000D))
```

在节点回归查询中使用归纳推理

假设您要在 Jupyter 笔记本的现有图形中添加一个新节点，如下所示：

```
%%gremlin
g.addV('label1').property(id,'101').as('newV')
 .V('1').as('oldV1')
 .V('2').as('oldV2')
 .addE('eLabel1').from('newV').to('oldV1')
 .addE('eLabel2').from('oldV2').to('newV')
```

然后，您可以使用归纳推理查询来获得考虑了新节点的评级：

```
%%gremlin
g.with("Neptune#ml.endpoint", "nr-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
```

```
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

由于查询不是确定性的，因此如果您根据邻域运行它多次，它返回的结果可能会有所不同：

```
# First time
==>vp[rating->9.1]

# Second time
==>vp[rating->8.9]
```

如果您需要更一致的结果，可以使查询具有确定性：

```
%%gremlin
g.with("Neptune#ml.endpoint", "nc-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').properties("rating")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

现在，每次的结果将大致相同：

```
# First time
==>vp[rating->9.1]

# Second time
==>vp[rating->9.1]
```

Neptune ML 中的 Gremlin 边缘分类查询

对于 Neptune ML 中的 Gremlin 边缘分类：

- 模型是在边缘的一个属性上训练的。此属性的唯一值集合称为一组类。
- 可以从边缘分类模型中推理边缘的类或分类属性值，当该属性尚未附加到边缘时，该模型很有用。
- 要从边缘分类模型中获取一个或多个类，您需要使用带有谓词 `"Neptune#ml.classification"` 的 `with()` 步骤来配置 `properties()` 步骤。如果这些是边缘属性，则输出格式与您期望的格式类似。

> **Note**
>
> 边缘分类仅适用于字符串属性值。这意味着不支持诸如 0 或 1 之类的数值属性值，但支持等同的字符串 "0" 和 "1"。同样，布尔属性值 true 和 false 不起作用，但 "true" 和 "false" 起作用。

以下是使用 Neptune#ml.score 谓词请求置信度分数的边缘分类查询的示例：

```
g.with("Neptune#ml.endpoint","edge-classification-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by", "Neptune#ml.score").with("Neptune#ml.classification")
```

响应将如下所示：

```
==>p[knows_by->"Family"]
==>p[Neptune#ml.score->0.01234567]
==>p[knows_by->"Friends"]
==>p[Neptune#ml.score->0.543210]
==>p[knows_by->"Colleagues"]
==>p[Neptune#ml.score->0.10101]
```

Gremlin 边缘分类查询的语法

对于简单的图形，其中 User 是头部和尾部节点，Relationship 是连接它们的边缘，边缘分类查询的示例为：

```
g.with("Neptune#ml.endpoint","edge-classification-social-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by").with("Neptune#ml.classification")
```

此查询的输出如下所示：

```
==>p[knows_by->"Family"]
==>p[knows_by->"Friends"]
==>p[knows_by->"Colleagues"]
```

在上面的查询中，E() 和 properties() 步骤的使用方式如下：

- E() 步骤包含要从边缘分类模型中获取类的一组边缘：

```
.E("relationship_1","relationship_2","relationship_3")
```

- properties() 步骤包含训练模型所依据的键，并具有 .with("Neptune#ml.classification") 以表明这是边缘分类机器学习推理查询。

目前不支持在 properties().with("Neptune#ml.classification") 步骤中使用多个属性键。例如，以下查询会导致引发异常：

```
g.with("Neptune#ml.endpoint","edge-classification-social-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("relationship_1","relationship_2","relationship_3")
 .properties("knows_by", "other_label").with("Neptune#ml.classification")
```

有关具体的错误消息，请参阅Neptune ML Gremlin 推理查询的异常列表。

properties().with("Neptune#ml.classification") 步骤可以与以下任何步骤结合使用：

- value()
- value().is()
- hasValue()
- has(value,"")
- key()
- key().is()
- hasKey()
- has(key,"")
- path()

在边缘分类查询中使用归纳推理

假设您要在 Jupyter 笔记本的现有图形中添加一个新边缘，如下所示：

```
%%gremlin
g.V('1').as('fromV')
.V('2').as('toV')
```

```
.addE('eLabel1').from('fromV').to('toV').property(id, 'e101')
```

然后，您可以使用归纳推理查询来获得一个考虑了新边缘的比例：

```
%%gremlin
g.with("Neptune#ml.endpoint", "ec-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("scale", "Neptune#ml.score")
 .with("Neptune#ml.classification")
 .with("Neptune#ml.inductiveInference")
```

由于查询不是确定性的，因此，如果您根据随机邻域多次运行该查询，结果会有所不同：

```
# First time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.21365921]
```

如果您需要更一致的结果，可以使查询具有确定性：

```
%%gremlin
g.with("Neptune#ml.endpoint", "ec-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("scale", "Neptune#ml.score")
 .with("Neptune#ml.classification")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

现在，每次运行查询时，结果都会大致相同：

```
# First time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]

# Second time
==>vp[scale->Like]
==>vp[Neptune#ml.score->0.12345678]
```

Neptune ML 中的 Gremlin 边缘回归查询

边缘回归与边缘分类类似，不同之处在于从 ML 模型推理的值是数值。对于边缘回归，Neptune ML 支持与分类相同的查询。

需要注意的要点是：

- 您需要使用 ML 谓词 "Neptune#ml.regression" 来配置此用例的 properties() 步骤。
- "Neptune#ml.limit" 和 "Neptune#ml.threshold" 谓词不适用于此用例。
- 要对值进行筛选，您需要将该值指定为数值。

Gremlin 边缘回归查询的语法

对于一个简单的图形，其中 User 是头节点，Movie 是尾节点，Rated 是连接它们的边缘，下面是一个边缘回归查询示例，它可以找到边缘 Rated 的数值评分值，此处称为分数：

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("rating_1","rating_2","rating_3")
 .properties("score").with("Neptune#ml.regression")
```

也可以根据从 ML 回归模型推理的值进行筛选。对于由 "rating_1"、"rating_2" 和 "rating_3" 标识的现有 Rated 边缘（从 User 到 Movie），如果这些评分不存在边缘属性 Score，则可以使用如下查询对于其大于或等于 9 的边缘推理 Score：

```
g.with("Neptune#ml.endpoint","edge-regression-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .E("rating_1","rating_2","rating_3")
 .properties("score").with("Neptune#ml.regression")
 .value().is(P.gte(9))
```

在边缘回归查询中使用归纳推理

假设您要在 Jupyter 笔记本的现有图形中添加一个新边缘，如下所示：

```
%%gremlin
g.V('1').as('fromV')
.V('2').as('toV')
.addE('eLabel1').from('fromV').to('toV').property(id, 'e101')
```

然后，您可以使用归纳推理查询来获得一个考虑了新边缘的分数：

```
%%gremlin
g.with("Neptune#ml.endpoint", "er-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("score")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
```

由于查询不是确定性的，因此，如果您根据随机邻域多次运行该查询，结果会有所不同：

```
# First time
==>ep[score->96]

# Second time
==>ep[score->91]
```

如果您需要更一致的结果，可以使查询具有确定性：

```
%%gremlin
g.with("Neptune#ml.endpoint", "er-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .E('e101').properties("score")
 .with("Neptune#ml.regression")
 .with("Neptune#ml.inductiveInference")
 .with("Neptune#ml.deterministic")
```

现在，每次运行查询时，结果都会大致相同：

```
# First time
==>ep[score->96]

# Second time
==>ep[score->96]
```

在 Neptune ML 中使用链接预测模型的 Gremlin 链接预测查询

链接预测模型可以解决如下问题：

- 头节点预测：给定顶点和边缘类型，该顶点可能从哪些顶点链接？

- 尾节点预测：给定顶点和边缘标签，该顶点可能链接到哪些顶点？

> **Note**
>
> Neptune ML 尚不支持边缘预测。

对于下面的示例，请考虑一个包含顶点 `User` 和 `Movie` 的简单图形，这些顶点通过边缘 `Rated` 相连。

以下是头节点预测查询示例，用于预测最有可能对电影 `"movie_1"`、`"movie_2"` 和 `"movie_3"` 进行评分的前五位用户：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .with("Neptune#ml.limit", 5)
 .V("movie_1", "movie_2", "movie_3")
 .in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

以下是类似的尾节点预测示例，用于预测用户 `"user_1"` 可能评分的前五部电影：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
 .out("rated").with("Neptune#ml.prediction").hasLabel("movie")
```

边缘标签和预测的顶点标签都是必需的。如果省略其中一个，则会引发异常。例如，以下没有预测顶点标签的查询会引发异常：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
 .out("rated").with("Neptune#ml.prediction")
```

同样，以下没有边缘标签的查询也会引发异常：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
```

```
  .out().with("Neptune#ml.prediction").hasLabel("movie")
```

有关这些异常返回的具体错误消息，请参阅 Neptune ML 异常列表。

其它链接预测查询

可以将 `select()` 步骤与 `as()` 步骤一起使用来输出预测的顶点以及输入顶点：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1").as("source")
 .in("rated").with("Neptune#ml.prediction").hasLabel("user").as("target")
 .select("source","target")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1").as("source")
 .out("rated").with("Neptune#ml.prediction").hasLabel("movie").as("target")
 .select("source","target")
```

您可以进行无界查询，如下所示：

```
g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("user_1")
 .out("rated").with("Neptune#ml.prediction").hasLabel("movie")

g.with("Neptune#ml.endpoint","node-prediction-movie-lens-endpoint")
 .with("Neptune#ml.iamRoleArn","arn:aws:iam::0123456789:role/sagemaker-role")
 .V("movie_1")
 .in("rated").with("Neptune#ml.prediction").hasLabel("user")
```

在链接预测查询中使用归纳推理

假设您要在 Jupyter 笔记本的现有图形中添加一个新节点，如下所示：

```
%%gremlin
g.addV('label1').property(id,'101').as('newV1')
 .addV('label2').property(id,'102').as('newV2')
 .V('1').as('oldV1')
 .V('2').as('oldV2')
 .addE('eLabel1').from('newV1').to('oldV1')
```

```
  .addE('eLabel2').from('oldV2').to('newV2')
```

然后，您可以使用归纳推理查询来预测头节点，同时考虑到新节点：

```
%%gremlin
g.with("Neptune#ml.endpoint", "lp-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('101').out("eLabel1")
 .with("Neptune#ml.prediction")
 .with("Neptune#ml.inductiveInference")
 .hasLabel("label2")
```

结果：

```
==>V[2]
```

同样，您可以使用归纳推理查询来预测尾节点，同时考虑到新节点：

```
%%gremlin
g.with("Neptune#ml.endpoint", "lp-ep")
 .with("Neptune#ml.iamRoleArn", "arn:aws:iam::123456789012:role/NeptuneMLRole")
 .V('102').in("eLabel2")
 .with("Neptune#ml.prediction")
 .with("Neptune#ml.inductiveInference")
 .hasLabel("label1")
```

结果：

```
==>V[1]
```

Neptune ML Gremlin 推理查询的异常列表

- **BadRequestException** – 无法加载所提供角色的凭证。

  消息：`Unable to load credentials for role: the specified IAM Role ARN.`
- **BadRequestException** – 指定的 IAM 角色无权调用 SageMaker 端点。

  消息：`User: the specified IAM Role ARN is not authorized to perform: sagemaker:InvokeEndpoint on resource: the specified endpoint.`
- **BadRequestException** – 指定的端点不存在。

消息：`Endpoint *the specified endpoint* not found.`

- **InternalFailureException** – 无法从 Amazon S3 获取 Neptune ML 实时归纳推理元数据。

  消息：`Unable to fetch Neptune ML - Real-Time Inductive Inference metadata from S3. Check the permissions of the S3 bucket or if the Neptune instance can connect to S3.`

- **InternalFailureException** – Neptune ML 无法在 Amazon S3 中找到用于实时归纳推理的元数据文件。

  消息：`Neptune ML cannot find the metadata file for Real-Time Inductive Inference in S3.`

- **InvalidParameterException** – 指定的端点在语法上无效。

  消息：`Invalid endpoint provided for external service query.`

- **InvalidParameterException** – 指定的 SageMaker 执行 IAM 角色 ARN 在语法上无效。

  消息：`Invalid IAM role ARN provided for external service query.`

- **InvalidParameterException** – 在查询的 `properties()` 步骤中指定了多个属性键。

  消息：`ML inference queries are currently supported for one property key.`

- **InvalidParameterException** – 在查询中指定了多个边缘标签。

  消息：`ML inference are currently supported only with one edge label.`

- **InvalidParameterException** – 在查询中指定了多个顶点标签约束。

  消息：`ML inference are currently supported only with one vertex label constraint.`

- **InvalidParameterException** – `Neptune#ml.classification` 和 `Neptune#ml.regression` 谓词都存在于同一个查询中。

  消息：`Both regression and classification ML predicates cannot be specified in the query.`

- **InvalidParameterException** – 在链接预测查询的 `in()` 或 `out()` 步骤中指定了多个边缘标签。

  消息：`ML inference are currently supported only with one edge label.`

- **InvalidParameterException** – 使用 Neptune#ml.score 指定了多个属性键。

消息：Neptune ML inference queries are currently supported for one property key and one Neptune#ml.score property key.

- **MissingParameterException** – 端点未在查询中指定，也未指定为数据库集群参数。

  消息：No endpoint provided for external service query.

- MissingParameterException – SageMaker 执行 IAM 角色未在查询中指定，也未指定为数据库集群参数。

  消息：No IAM role ARN provided for external service query.

- **MissingParameterException** – 查询的 properties() 步骤中缺少属性键。

  消息：Property key needs to be specified using properties() step for ML inference queries.

- **MissingParameterException** – 在链接预测查询的 in() 或 out() 步骤中未指定边缘标签。

  消息：Edge label needs to be specified while using in() or out() step for ML inference queries.

- **MissingParameterException** – 未使用 Neptune#ml.score 指定任何属性键。

  消息：Property key needs to be specified along with Neptune#ml.score property key while using the properties() step for Neptune ML inference queries.

- **UnsupportedOperationException** – both() 步骤用于链接预测查询。

  消息：ML inference queries are currently not supported with both() step.

- **UnsupportedOperationException**– 在链接预测查询中，没有在带有 in() 或 out() 步骤的 has() 步骤中指定预测顶点标签。

  消息：Predicted vertex label needs to be specified using has() step for ML inference queries.

- **UnsupportedOperationException** – 未优化的步骤目前不支持 Gremlin ML 归纳推理查询。

  消息：Neptune ML - Real-Time Inductive Inference queries are currently not supported with Gremlin steps which are not optimized for Neptune. Check the Neptune User Guide for a list of Neptune-optimized steps.

- **UnsupportedOperationException** – repeat 步骤中目前不支持 Neptune ML 推理查询。

消息：Neptune ML inference queries are currently not supported inside a repeat step.

- **UnsupportedOperationException** – 每个 Gremlin 查询目前仅支持一个 Neptune ML 推理查询。

  消息：Neptune ML inference queries are currently supported only with one ML inference query per gremlin query.

Neptune ML 中的 SPARQL 推理查询

Neptune ML 将 RDF 图形映射到属性图，以便对 ML 任务建模。目前，它支持以下用例：

- 对象分类 - 预测对象的分类特征。
- 对象回归 - 预测对象的数值属性。
- 对象预测 - 在给定主题和关系的情况下预测对象。
- 主题预测 – 在给定对象和关系的情况下预测主题。

> **Note**
>
> Neptune ML 不支持 SPARQL 的主题分类和回归用例。

SPARQL 推理查询中使用的 Neptune ML 谓词

以下谓词用于 SPARQL 推理：

`neptune-ml:timeout` 谓词

指定与远程服务器连接的超时。不应与查询请求超时混淆，查询请求超时是服务器满足请求所花费的最大时间量。

请注意，如果查询超时在 `neptune-ml:timeout` 谓词指定的服务超时之前发生，则服务连接也会被取消。

`neptune-ml:outputClass` 谓词

`neptune-ml:outputClass` 谓词仅用于定义用于对象预测的预测对象或用于主题预测的预测主题的类别。

`neptune-ml:outputScore` 谓词

`neptune-ml:outputScore` 谓词是一个正数，它表示机器学习模型的输出正确的可能性。

`neptune-ml:modelType` 谓词

`neptune-ml:modelType` 谓词指定正在训练的机器学习模型的类型：

- `OBJECT_CLASSIFICATION`

- OBJECT_REGRESSION
- OBJECT_PREDICTION
- SUBJECT_PREDICTION

neptune-ml:input 谓词

neptune-ml:input 谓词是指用作 Neptune ML 的输入的 URI 列表。

neptune-ml:output 谓词

neptune-ml:output 谓词是指 Neptune ML 返回结果的绑定集列表。

neptune-ml:predicate 谓词

根据正在执行的任务，neptune-ml:predicate 谓词的使用方式有所不同：

- 对于对象或主题预测：定义谓词的类型（边缘或关系类型）。
- 对于对象分类和回归：定义我们要预测的文本（属性）。

neptune-ml:batchSize 谓词

neptune-ml:batchSize 指定远程服务调用的输入大小。

SPARQL 对象分类示例

对于 Neptune ML 中的 SPARQL 对象分类，模型是根据其中一个谓词值进行训练的。当给定主题中还没有该谓词时，这很有用。

使用对象分类模型只能推理分类谓词值。

下面的查询试图预测类型为 foaf:Person 的所有输入的 <http://www.example.org/team> 谓词值：

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/team> ;
                      neptune-ml:output ?output .
  }
}
```

可以按如下方式自定义此查询：

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;

                      neptune-ml:batchSize "40"^^xsd:integer ;
                      neptune-ml:timeout "1000"^^xsd:integer ;

                      neptune-ml:modelType 'OBJECT_CLASSIFICATION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/team> ;
                      neptune-ml:output ?output .
  }
}
```

SPARQL 对象回归示例

对象回归与对象分类类似，不同之处在于从回归模型中推理的每个节点的数值谓词值。除了 the Neptune#ml.limit 和 Neptune#ml.threshold 谓词不适用之外，您可以对于对象回归和对象分类使用相同的 SPARQL 查询。

下面的查询试图预测类型为 foaf:Person 的所有输入的 <http://www.example.org/accountbalance> 谓词值：

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_REGRESSION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/accountbalance> ;
                      neptune-ml:output ?output .
  }
}
```

可以按如下方式自定义此查询：

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-account-balance-endpoint' ;
```

```
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;

                      neptune-ml:batchSize "40"^^xsd:integer ;
                      neptune-ml:timeout "1000"^^xsd:integer ;

                      neptune-ml:modelType 'OBJECT_REGRESSION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/accountbalance> ;
                      neptune-ml:output ?output .
  }
}
```

SPARQL 对象预测示例

对象预测预测给定主题和谓词的对象值。

以下对象预测查询试图预测类型为 `foaf:Person` 的输入喜欢什么电影：

```
?x a foaf:Person .
?x   <http://www.example.org/likes> ?m .
?m a <http://www.example.org/movie> .

## Query
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:modelType 'OBJECT_PREDICTION' ;
                      neptune-ml:input ?input ;
                      neptune-ml:predicate <http://www.example.org/likes> ;
                      neptune-ml:output ?output ;
                      neptune-ml:outputClass <http://www.example.org/movie> .
  }
}
```

查询本身可以按如下方式自定义：

```
SELECT * WHERE { ?input a foaf:Person .
  SERVICE neptune-ml:inference {
    neptune-ml:config neptune-ml:endpoint 'node-prediction-user-movie-prediction-
endpoint' ;
                      neptune-ml:iamRoleArn 'arn:aws:iam::0123456789:role/sagemaker-
role' ;
```

```
                    neptune-ml:limit "5"^^xsd:integer ;
                    neptune-ml:batchSize "40"^^xsd:integer ;
                    neptune-ml:threshold "0.1"^^xsd:double ;
                    neptune-ml:timeout "1000"^^xsd:integer ;
                    neptune-ml:outputScore ?score ;

                    neptune-ml:modelType 'OBJECT_PREDICTION' ;
                    neptune-ml:input ?input ;
                    neptune-ml:predicate <http://www.example.org/likes> ;
                    neptune-ml:output ?output ;
                    neptune-ml:outputClass <http://www.example.org/movie> .
  }
}
```

SPARQL 主题预测示例

主题预测在给定谓语和对象的情况下预测主题。

例如，以下查询预测谁（类型为 `foaf:User`）将观看给定的电影：

```
SELECT * WHERE { ?input (a foaf:Movie) .
    SERVICE neptune-ml:inference {
        neptune-ml:config neptune-ml:modelType 'SUBJECT_PREDICTION' ;
                          neptune-ml:input ?input ;
                          neptune-ml:predicate <http://aws.amazon.com/neptune/csv2rdf/
object_Property/rated> ;
                          neptune-ml:output ?output ;
                          neptune-ml:outputClass <http://aws.amazon.com/neptune/
csv2rdf/class/User> ;        }
}
```

Neptune ML SPARQL 推理查询的异常列表

- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects at least 1 value for the parameter *(parameter name)*, found zero.
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects at most 1 value for the parameter *(parameter name)*, found *(a number)* values.

- **BadRequestException** – 消息：Invalid predicate *(predicate name)* provided for external service http://aws.amazon.com/neptune/vocab/v01/services/ml#inference query.
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the predicate *(predicate name)* to be defined。
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the value of (parameter) *(parameter name)* to be a variable, found: *(type)*"
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference expects the input *(parameter name)* to be a constant, found: *(type)*。
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference is expected to return only 1 value。
- **BadRequestException** – 消息："The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference only allows StatementPatternNodes。
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference does not allow the predicate *(predicate name)*。
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference predicates cannot be variables, found: *(type)*。
- **BadRequestException** – 消息：The SERVICE http://aws.amazon.com/neptune/vocab/v01/services/ml#inference predicates are expected to be part of the namespace *(namespace name)*, found: *(namespace name)*。

Neptune ML 管理 API 参考

目录

使用 **dataprocessing** 命令处理数据

您可以使用 Neptune ML `dataprocessing` 命令创建数据处理任务、检查其状态、停止它或列出所有活动的数据处理任务。

使用 Neptune ML **dataprocessing** 命令创建数据处理任务

用于创建新任务的典型 Neptune ML `dataprocessing` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for the new job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)"
      }'
```

启动增量重新处理的命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/dataprocessing \
  -H 'Content-Type: application/json' \
  -d '{
        "inputDataS3Location" : "s3://(Amazon S3 bucket name)/(path to your input
 folder)",
        "id" : "(a job ID for this job)",
        "processedDataS3Location" : "s3://(S3 bucket name)/(path to your output
 folder)"
        "previousDataProcessingJobId" : "(the job ID of a previously completed job to
 update)"
}'
```

用于创建 **dataprocessing** 任务的参数

- **id** – （可选）新任务的唯一标识符。

  类型：字符串。默认：自动生成的 UUID。
- **previousDataProcessingJobId** – （可选）在较早版本的数据上运行的已完成数据处理任务的任务 ID。

类型：字符串。默认值：无。

注意：使用它进行增量数据处理，以便在图形数据发生变化（但不是在数据已被删除）时更新模型。

- **`inputDataS3Location`** –（必需）您希望 SageMaker 下载运行数据处理任务所需数据的 Amazon S3 位置的 URI。

  类型：字符串。

- **`processedDataS3Location`** –（必需）您希望 SageMaker 保存数据处理任务结果的 Amazon S3 位置的 URI。

  类型：字符串。

- **`sagemakerIamRoleArn`** –（可选）用于执行 SageMaker 的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **`neptuneIamRoleArn`** –（可选）SageMaker 可以代入以代表您执行任务的 IAM 角色的 Amazon 资源名称 (ARN)。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **`processingInstanceType`** –（可选）在数据处理期间使用的机器学习实例的类型。它的内存应该足够大，可以容纳处理后的数据集。

  类型：字符串。默认：内存比磁盘上导出的图形数据大小大十倍的最小 `ml.r5` 类型。

  注意：Neptune ML 可以自动选择实例类型。请参阅选择进行数据处理的实例。

- **`processingInstanceVolumeSizeInGB`** –（可选）处理实例的磁盘卷大小。输入数据和处理后的数据都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。

  类型：整数。默认值：`0`。

  注意：如果未指定或为 0，则 Neptune ML 会根据数据大小自动选择卷大小。

- **`processingTimeOutInSeconds`** –（可选）数据处理任务的超时（以秒为单位）。

  类型：整数。默认值：`86,400`（1 天）。

- **`modelType`** –（可选）Neptune ML 当前支持的两种模型类型之一：异构图模型 (`heterogeneous`) 和知识图谱 (`kge`)。

  类型：字符串。默认值：无。

注意：如果未指定，Neptune ML 会根据数据自动选择模型类型。

- **configFileName** – （可选）描述如何加载导出的图形数据进行训练的数据规范文件。该文件由 Neptune 导出工具包自动生成。

  类型：字符串。默认值：`training-data-configuration.json`。

- **subnets** – （可选）Neptune VPC 中子网的 ID。

  类型：字符串列表。默认值：无。

- **securityGroupIds** – （可选）VPC 安全组 ID。

  类型：字符串列表。默认值：无。

- **volumeEncryptionKMSKey** – （可选）AWS Key Management Service (AWS KMS) 密钥，SageMaker 使用它来加密附加到运行处理任务的 ML 计算实例的存储卷上的数据。

  类型：字符串。默认值：无。

- **enableInterContainerTrafficEncryption** – （可选）在训练或超参数调整任务中启用或禁用容器间流量加密。

  类型：布尔值。默认值：True。

  > **Note**
  >
  > `enableInterContainerTrafficEncryption` 参数仅在引擎版本 1.2.0.2.R3 中可用。

- **s3OutputEncryptionKMSKey**– （可选）SageMaker 用来加密训练任务输出的 AWS Key Management Service (AWS KMS) 密钥。

  类型：字符串。默认值：无。

使用 Neptune ML **dataprocessing** 命令获取数据处理任务的状态

用于显示任务状态的示例 Neptune ML `dataprocessing` 命令如下所示：

```
curl -s \
  "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)" \
  | python -m json.tool
```

dataprocessing 任务状态的参数

- **id** – (必需) 数据处理任务的唯一标识符。

  类型：字符串。

- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 Neptune ML **dataprocessing** 命令停止数据处理任务

用于停止任务的示例 Neptune ML `dataprocessing` 命令如下所示：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)"
```

或者：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/dataprocessing/(the job ID)?clean=true"
```

dataprocessing 停止任务的参数

- **id** – (必需) 数据处理任务的唯一标识符。

  类型：字符串。

- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **clean** – (可选) 此标志指定在任务停止时应删除所有 Amazon S3 构件。

  类型：布尔值。默认值：FALSE。

使用 Neptune ML **dataprocessing** 命令列出处于活动状态的数据处理任务

用于列出活动任务的示例 Neptune ML `dataprocessing` 命令如下所示：

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing"
```

或者：

```
curl -s "https://(your Neptune endpoint)/ml/dataprocessing?maxItems=3"
```

dataprocessing 列出任务的参数

- **maxItems** – (可选) ，表示要返回的最大项目数。

  类型：整数。默认值：10。允许的最大值：1024。

- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 **modeltraining** 命令进行模型训练

您可以使用 Neptune ML `modeltraining` 命令创建模型训练任务、检查其状态、停止它或列出所有活动的模型训练任务。

使用 Neptune ML **modeltraining** 命令创建模型训练任务

用于创建全新任务的 Neptune ML `modeltraining` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
      }'
```

用于为增量模型训练创建更新任务的 Neptune ML `modeltraining` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "previousModelTrainingJobId" : "(the job ID of a completed model-training job
 to update)",
      }'
```

使用用户提供的自定义模型实现创建新任务的 Neptune ML `modeltraining` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltraining
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "dataProcessingJobId" : "(the data-processing job-id of a completed job)",
```

```
        "trainModelS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-graph-
autotrainer"
        "modelName": "custom",
        "customModelTrainingParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "trainingEntryPointScript": "(your training script entry-point name in the
 Python module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

用于创建 **modeltraining** 任务的参数

- **id** – (可选) 新任务的唯一标识符。

  类型：字符串。默认：自动生成的 UUID。

- **dataProcessingJobId** – (必需) 已完成的数据处理任务的任务 ID，该任务已创建训练将使用的数据。

  类型：字符串。

- **trainModelS3Location** – (必需) Amazon S3 中要存储模型构件的位置。

  类型：字符串。

- **previousModelTrainingJobId** – (可选) 已完成的模型训练任务的任务 ID，您要根据更新的数据以递增方式更新该任务。

  类型：字符串。默认值：无。

- **sagemakerIamRoleArn** – (可选) 用于执行 SageMaker 的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **modelName** – (可选) 用于训练的模型类型。默认情况下，机器学习模型是根据数据处理中使用的 `modelType` 自动生成的，但您可以在此处指定不同的模型类型。

类型：字符串。默认：rgcn 用于异构图形，kge 用于知识图谱。有效值：对于异构图形：rgcn。对于 kge 图形：transe、distmult 或 rotate。对于自定义模型实现：custom。

- **baseProcessingInstanceType** – （可选）用于准备和管理机器学习模型训练的机器学习实例的类型。

  类型：字符串。注意：这是根据用于处理训练数据和模型的内存要求选择的 CPU 实例。请参阅为模型训练和模型转换选择实例。

- **trainingInstanceType** – （可选）用于模型训练的 ML 实例的类型。所有 Neptune ML 模型都支持 CPU、GPU 和多 GPU 训练。

  类型：字符串。默认值：ml.p3.2xlarge。

  注意：为训练选择正确的实例类型取决于任务类型、图形大小和预算。请参阅为模型训练和模型转换选择实例。

- **trainingInstanceVolumeSizeInGB** – （可选）训练实例的磁盘卷大小。输入数据和输出模型都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。

  类型：整数。默认值：0。

  注意：如果未指定或为 0，Neptune ML 会根据数据处理步骤中生成的建议选择磁盘卷大小。请参阅为模型训练和模型转换选择实例。

- **trainingTimeOutInSeconds** – （可选）训练任务的超时（以秒为单位）。

  类型：整数。默认值：86,400（1 天）。

- **maxHPONumberOfTrainingJobs** - 超参数调整任务要启动的最大训练任务总数。

  类型：整数。默认值：2。

  注意：Neptune ML 会自动调整机器学习模型的超参数。要获得性能良好的模型，请至少使用 10 个任务（换句话说，将 maxHPONumberOfTrainingJobs 设置为 10）。通常，调整次数越多，结果越好。

- **maxHPOParallelTrainingJobs** – 为超参数调整任务启动的最大并行训练任务数。

  类型：整数。默认值：2。

  注意：您可以运行的并行任务数量受训练实例上可用资源的限制。

- **subnets** – （可选）Neptune VPC 中子网的 ID。

类型：字符串列表。默认值：无。

- **securityGroupIds** – （可选）VPC 安全组 ID。

  类型：字符串列表。默认值：无。

- **volumeEncryptionKMSKey** – （可选）AWS Key Management Service (AWS KMS) 密钥，SageMaker 使用它来加密附加到运行训练任务的 ML 计算实例的存储卷上的数据。

  类型：字符串。默认值：无。

- **s3OutputEncryptionKMSKey** – （可选）SageMaker 用来加密处理任务输出的 AWS Key Management Service (AWS KMS) 密钥。

  类型：字符串。默认值：无。

- **enableInterContainerTrafficEncryption** – （可选）在训练或超参数调整任务中启用或禁用容器间流量加密。

  类型：布尔值。默认值：True。

  > **Note**
  >
  > `enableInterContainerTrafficEncryption` 参数仅在引擎版本 1.2.0.2.R3 中可用。

- **enableManagedSpotTraining** – （可选）使用 Amazon Elastic Compute Cloud 竞价型实例优化训练机器学习模型的成本。有关更多信息，请参阅 Amazon SageMaker 中的托管式竞价型训练。

  类型：布尔值。默认值：False。

- **customModelTrainingParameters** – （可选）自定义模型训练的配置。这是具有以下字段的 JSON 对象：
  - **sourceS3DirectoryPath** – （必需）实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 `model-hpo-configuration.json` 文件。
  - **trainingEntryPointScript** – （可选）执行模型训练并将超参数作为命令行参数（包括固定的超参数）的脚本模块中入口点的名称。

    默认值：`training.py`。
  - **transformEntryPointScript** – （可选）脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。

默认值：transform.py。

- **maxWaitTime** – （可选）使用竞价型实例执行模型训练时等待的最长时间，以秒为单位。应大于 trainingTimeOutInSeconds。

  类型：整数。

使用 Neptune ML **modeltraining** 命令获取模型训练任务的状态

用于显示任务状态的示例 Neptune ML modeltraining 命令如下所示：

```
curl -s \
  "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)" \
  | python -m json.tool
```

modeltraining 任务状态的参数

- **id** – （必需）模型训练任务的唯一标识符。

  类型：字符串。

- **neptuneIamRoleArn** – （可选）向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 Neptune ML **modeltraining** 命令停止模型训练任务

用于停止任务的示例 Neptune ML modeltraining 命令如下所示：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)"
```

或者：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltraining/(the job ID)?clean=true"
```

`modeltraining` 停止任务的参数

- **`id`** – (必需) 模型训练任务的唯一标识符。

  类型：字符串。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。
- **`clean`** – (可选) 此标志指定在任务停止时应删除所有 Amazon S3 构件。

  类型：布尔值。默认值：FALSE。

使用 Neptune ML `modeltraining` 命令列出活动的模型训练任务

用于列出活动任务的示例 Neptune ML `modeltraining` 命令如下所示：

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining" | python -m json.tool
```

或者：

```
curl -s "https://(your Neptune endpoint)/ml/modeltraining?maxItems=3" | python -m
 json.tool
```

`modeltraining` 列出任务的参数

- **`maxItems`** – (可选) ，表示要返回的最大项目数。

  类型：整数。默认值：10。允许的最大值：1024。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 **modeltransform** 命令进行模型转换

您可以使用 Neptune ML `modeltransform` 命令创建模型转换任务、检查其状态、停止它或列出所有活动的模型转换任务。

使用 Neptune ML **modeltransform** 命令创建模型转换任务

用于创建增量转换任务的 Neptune ML `modeltransform` 命令（无需重新训练模型）如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-transform job ID)",
        "dataProcessingJobId" : "(the job-id of a completed data-processing job)",
        "mlModelTrainingJobId" : "(the job-id of a completed model-training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform"
      }'
```

用于从已完成的 SageMaker 训练任务中创建任务的 Neptune ML `modeltransform` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-transform job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform",
        "baseProcessingInstanceType" : ""
      }'
```

用于创建使用自定义模型实现的任务的 Neptune ML `modeltransform` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/modeltransform
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique model-training job ID)",
        "trainingJobName" : "(name of a completed SageMaker training job)",
```

```
        "modelTransformOutputS3Location" : "s3://(your Amazon S3 bucket)/neptune-model-
transform/"
        "customModelTransformParameters" : {
          "sourceS3DirectoryPath": "s3://(your Amazon S3 bucket)/(path to your Python
 module)",
          "transformEntryPointScript": "(your transform script entry-point name in the
 Python module)"
        }
      }'
```

用于创建 **modeltransform** 任务的参数

- **id** – (可选) 新任务的唯一标识符。

  类型：字符串。默认：自动生成的 UUID。
- **dataProcessingJobId**– 已完成的数据处理任务的任务 ID。

  类型：字符串。

  注意：必须同时包含 dataProcessingJobId 和 mlModelTrainingJobId 或 trainingJobName。
- **mlModelTrainingJobId** – 已完成的模型训练任务的任务 ID。

  类型：字符串。

  注意：必须同时包含 dataProcessingJobId 和 mlModelTrainingJobId 或 trainingJobName。
- **trainingJobName** – 已完成的 SageMaker 训练任务的名称。

  类型：字符串。

  注意：必须同时包含 dataProcessingJobId 和 mlModelTrainingJobId 参数或 trainingJobName 参数。
- **sagemakerIamRoleArn** – (可选) 用于执行 SageMaker 的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。
- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

- **customModelTransformParameters** – (可选) 使用自定义模型进行模型转换的配置信息。customModelTransformParameters 对象包含以下字段，这些字段的值必须与训练任务中保存的模型参数兼容：
  - **sourceS3DirectoryPath** – (必需) 实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 model-hpo-configuration.json 文件。
  - **transformEntryPointScript** – (可选) 脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。

    默认值：transform.py。
- **baseProcessingInstanceType** – (可选) 用于准备和管理机器学习模型训练的机器学习实例的类型。

  类型：字符串。注意：这是根据用于处理转换数据和模型的内存要求选择的 CPU 实例。请参阅为模型训练和模型转换选择实例。
- **baseProcessingInstanceVolumeSizeInGB** – (可选) 训练实例的磁盘卷大小。输入数据和输出模型都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。

  类型：整数。默认值：0。

  注意：如果未指定或为 0，Neptune ML 会根据数据处理步骤中生成的建议选择磁盘卷大小。请参阅为模型训练和模型转换选择实例。
- **subnets** – (可选) Neptune VPC 中子网的 ID。

  类型：字符串列表。默认值：无。
- **securityGroupIds** – (可选) VPC 安全组 ID。

  类型：字符串列表。默认值：无。
- **volumeEncryptionKMSKey** – (可选) AWS Key Management Service (AWS KMS) 密钥，SageMaker 使用它来加密附加到运行转换任务的 ML 计算实例的存储卷上的数据。

  类型：字符串。默认值：无。
- **enableInterContainerTrafficEncryption** – (可选) 在训练或超参数调整任务中启用或禁用容器间流量加密。

  类型：布尔值。默认值：True。

> **Note**
>
> `enableInterContainerTrafficEncryption` 参数仅在引擎版本 1.2.0.2.R3 中可用。

- **`s3OutputEncryptionKMSKey`** – （可选）SageMaker 用来加密处理任务输出的 AWS Key Management Service (AWS KMS) 密钥。

  类型：字符串。默认值：无。

使用 Neptune ML **`modeltransform`** 命令获取模型转换任务的状态

用于显示任务状态的示例 Neptune ML `modeltransform` 命令如下所示：

```
curl -s \
  "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)" \
  | python -m json.tool
```

`modeltransform` 任务状态的参数

- **`id`** – （必需）模型转换任务的唯一标识符。

  类型：字符串。

- **`neptuneIamRoleArn`** – （可选）向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 Neptune ML **`modeltransform`** 命令停止模型转换任务

用于停止任务的示例 Neptune ML `modeltransform` 命令如下所示：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)"
```

或者：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/modeltransform/(the job ID)?clean=true"
```

`modeltransform` 停止任务的参数

- **`id`** – (必需) 模型转换任务的唯一标识符。

  类型：字符串。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。
- **`clean`** – (可选) 此标志指定在任务停止时应删除所有 Amazon S3 构件。

  类型：布尔值。默认值：FALSE。

使用 Neptune ML `modeltransform` 命令列出活动的模型转换任务

用于列出活动任务的示例 Neptune ML `modeltransform` 命令如下所示：

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform" | python -m json.tool
```

或者：

```
curl -s "https://(your Neptune endpoint)/ml/modeltransform?maxItems=3" | python -m
 json.tool
```

`modeltransform` 列出任务的参数

- **`maxItems`** – (可选) ，表示要返回的最大项目数。

  类型：整数。默认值：10。允许的最大值：1024。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将发生错误。

使用 **endpoints** 命令管理推理端点

您可以使用 Neptune ML `endpoints` 命令创建推理端点、检查其状态、将其删除或列出现有推理端点。

使用 Neptune ML **endpoints** 命令创建推理端点

用于根据训练任务创建的模型创建推理端点的 Neptune ML `endpoints` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

用于根据训练任务创建的模型更新现有推理端点的 Neptune ML `endpoints` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "update" : "true",
        "mlModelTrainingJobId": "(the model-training job-id of a completed job)"
      }'
```

用于根据模型转换任务创建的模型创建推理端点的 Neptune ML `endpoints` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "mlModelTransformJobId": "(the model-training job-id of a completed job)"
      }'
```

用于根据模型转换任务创建的模型更新现有推理端点的 Neptune ML `endpoints` 命令如下所示：

```
curl \
  -X POST https://(your Neptune endpoint)/ml/endpoints
```

```
  -H 'Content-Type: application/json' \
  -d '{
        "id" : "(a unique ID for the new endpoint)",
        "update" : "true",
        "mlModelTransformJobId": "(the model-training job-id of a completed job)"
      }'
```

用于创建 **endpoints** 推理端点的参数

- **id** – (可选) 新推理端点的唯一标识符。

  类型：字符串。默认：自动生成的带有时间戳的名称。
- **mlModelTrainingJobId** – 已完成的模型训练任务的任务 ID，该任务创建了推理端点将指向的模型。

  类型：字符串。

  注意：您必须提供 mlModelTrainingJobId 或 mlModelTransformJobId。
- **mlModelTransformJobId** – 已完成的模型转换任务的任务 ID。

  类型：字符串。

  注意：您必须提供 mlModelTrainingJobId 或 mlModelTransformJobId。
- **update** – (可选) 如果存在，则此参数表示这是更新请求。

  类型：布尔值。默认值：false

  注意：您必须提供 mlModelTrainingJobId 或 mlModelTransformJobId。
- **neptuneIamRoleArn** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将引发错误。
- **modelName** – (可选) 训练的模型类型。默认情况下，机器学习模型是根据数据处理中使用的 modelType 自动生成的，但您可以在此处指定不同的模型类型。

  类型：字符串。默认：rgcn 用于异构图形，kge 用于知识图谱。有效值：对于异构图形：rgcn。对于知识图谱：kge、transe、distmult 或 rotate。
- **instanceType** – (可选) 用于在线服务的机器学习实例的类型。

  类型：字符串。默认值：ml.m5.xlarge。

注意：为推理端点选择 ML 实例取决于任务类型、图形大小和预算。请参阅为推理端点选择实例。

- **instanceCount** – （可选）部署到端点以进行预测的最少 Amazon EC2 实例数量。

  类型：整数。默认值：1。

- **volumeEncryptionKMSKey** – （可选）AWS Key Management Service (AWS KMS) 密钥，SageMaker 使用它来加密附加到运行端点的 ML 计算实例的存储卷上的数据。

  类型：字符串。默认值：无。

使用 Neptune ML **endpoints** 命令获取推理端点的状态

用于显示实例端点状态的示例 Neptune ML `endpoints` 命令如下所示：

```
curl -s \
  "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)" \
  | python -m json.tool
```

endpoints 实例端点状态的参数

- **id** – （必需）推理端点的唯一标识符。

  类型：字符串。

- **neptuneIamRoleArn** – （可选）向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将引发错误。

使用 Neptune ML **endpoints** 命令删除实例端点

用于删除实例端点的 Neptune ML `endpoints` 命令示例如下所示：

```
curl -s \
  -X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)"
```

或者：

```
curl -s \
```

```
  -X DELETE "https://(your Neptune endpoint)/ml/endpoints/(the inference endpoint ID)?
clean=true"
```

`endpoints` 删除推理端点的参数

- **`id`** – (必需) 推理端点的唯一标识符。

  类型：字符串。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将引发错误。
- **`clean`** – (可选) 表示还应删除与此端点相关的所有构件。

  类型：布尔值。默认值：FALSE。

使用 Neptune ML **`endpoints`** 命令列出推理端点

用于列出推理端点的 Neptune ML `endpoints` 命令如下所示：

```
curl -s "https://(your Neptune endpoint)/ml/endpoints" \
  | python -m json.tool
```

或者：

```
curl -s "https://(your Neptune endpoint)/ml/endpoints?maxItems=3" \
  | python -m json.tool
```

`dataprocessing` 列出推理端点的参数

- **`maxItems`** – (可选) ，表示要返回的最大项目数。

  类型：整数。默认值：`10`。允许的最大值：`1024`。
- **`neptuneIamRoleArn`** – (可选) 向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。

  类型：字符串。注意：必须将其列在您的数据库集群参数组中，否则将引发错误。

Neptune ML 管理 API 错误和异常

所有 Neptune ML 管理 API 异常都会返回 400 HTTP 代码。在收到这些异常中的任何一个之后，不应重试生成了异常的命令。

- **MissingParameterException** – 错误消息：

  ```
  Required credentials are missing. Please add IAM role to the cluster or pass as a parameter to this request.
  ```

- **InvalidParameterException** – 错误消息：
  - `Invalid ML instance type.`
  - `Invalid ID provided. ID can be 1-48 alphanumeric characters.`
  - `Invalid ID provided. Must contain only letters, digits, or hyphens.`
  - `Invalid ID provided. Please check whether a resource with the given ID exists.`
  - `Another resource with same ID already exists. Please use a new ID.`
  - `Failed to stop the job because it has already completed or failed.`
- **BadRequestException** – 错误消息：
  - `Invalid S3 URL or incorrect S3 permissions. Please check your S3 configuration.`
  - `Provided ModelTraining job has not completed.`
  - `Provided SageMaker Training job has not completed.`
  - `Provided MLDataProcessing job is not completed.`
  - `Provided MLModelTraining job doesn't exist.`
  - `Provided ModelTransformJob doesn't exist.`
  - `Unable to find SageMaker resource. Please check your input.`

Neptune ML 限制

- 目前支持的推理类型包括节点分类、节点回归、边缘分类、边缘回归和链接预测（请参阅Neptune ML 功能）。
- Neptune ML 可以支持的最大图形大小取决于在数据准备、模型训练和推理期间所需的内存和存储量。
  - SageMaker 数据处理实例的最大内存大小为 768GB。因此，如果需要超过 768GB 的内存，数据处理阶段就会失败。
  - SageMaker 训练实例的最大内存大小为 732GB。因此，如果需要超过 732GB 的内存，则训练阶段就会失败。
- SageMaker 端点的推理负载的最大大小为 6MiB。因此，如果子图形负载超过此大小，则归纳推理就会失败。
- Neptune ML 目前仅在全部支持 Neptune 及其所依赖的其它服务（例如 AWS Lambda、Amazon API Gateway 和 Amazon SageMaker）的区域可用。

  中国（北京）和中国（宁夏）在默认使用 IAM 身份验证方面存在差异，如此处所述以及其它差异。
- Neptune ML 启动的链接预测推理端点目前只能预测与训练期间图形中存在的节点之间的可能链接。

  例如，假设一个带有 `User` 和 `Movie` 顶点和 `Rated` 边缘的图形。使用相应的 Neptune ML 链接预测推荐模型，您可以向图形中添加新用户并让模型为他们预测电影，但该模型只能推荐在模型训练期间存在的电影。尽管 `User` 节点嵌入是使用其本地子图形和 GNN 模型实时计算的，因此在用户对电影进行评分时可能会随着时间的推移而变化，但还是将其与静态、预先计算的电影嵌入进行比较，以获得最终推荐。
- Neptune ML 支持的 KGE 模型仅适用于链接预测任务，并且表示形式特定于训练期间图形中存在的顶点和边缘类型。这意味着，在训练期间，推理查询中提及的所有顶点和边缘类型都必须存在于图形中。如果不重新训练模型，就无法预测新的边缘类型或顶点。

SageMaker 资源限制

根据您一段时间内的活动和资源使用情况，您可能会遇到错误消息，提示您已超过限额 (ResourceLimitExceeded)，并且需要纵向扩展 SageMaker 资源，请按照本页上申请增加 SageMaker 资源的服务限额过程中的步骤向 AWS Support 申请增加限额。

SageMaker 资源名称对应于 Neptune ML 阶段，如下所示：

- Neptune 数据处理、模型训练和模型转换任务使用 `ProcessingJob` SageMaker。

- Neptune 模型训练任务使用 `HyperParameterTuningJob` SageMaker。
- Neptune 模型训练任务使用 `TrainingJob` SageMaker。
- Neptune 推理端点使用 `Endpoint` SageMaker。

监控 Amazon Neptune 资源

Amazon Neptune 支持多种用于监控数据库性能和使用情况的方法：

- 实例状态 – 检查 Neptune 集群的图形数据库引擎的运行状况，查明安装了引擎的哪个版本，并使用实例状态 API 获取其它实例相关信息。
- 图形摘要 API – 图形摘要 API 可让您快速全面了解图形数据大小和内容。

> **Note**
>
> 由于图形摘要 API 依赖于 DFE 统计数据，因此它仅在启用统计数据后才可用，而 T3 和 T4g 实例类型则不是这样。

- 亚马逊 CloudWatch — Neptune 会自动向警报发送指标 CloudWatch 并支持 CloudWatch 警报。有关更多信息，请参阅 the section called “使用 CloudWatch”。
- 审计日志文件 – 使用 Neptune 控制台来查看、下载或监视数据库日志文件。有关更多信息，请参阅 the section called “使用 Neptune 审计日志”。
- 将日志发布到 Amazon CloudWatch 日志 — 您可以将 Neptune 数据库集群配置为将审核日志数据发布到 Ama CloudWatch zon Logs 中的日志组。借助 CloudWatch 日志，您可以对日志数据进行实时分析，用于 CloudWatch 创建警报和查看指标，并使用 CloudWatch 日志将日志记录存储在高度耐用的存储中。请参阅 Neptun CloudWatch e 日志。
- AWS CloudTrail— Neptune 支持使用 API 日志记录。 CloudTrail有关更多信息，请参阅 the section called “使用记录 Neptune API 调用 AWS CloudTrail”。
- 事件通知订阅 – 订阅 Neptune 事件，随时了解正在发生的事情。有关更多信息，请参阅 the section called “事件通知”。
- 标记 – 使用标签向 Neptune 资源添加元数据并基于标签跟踪使用情况。有关更多信息，请参阅 the section called “给 Neptune 资源加标签”。

主题

- 检查 Neptune 实例的运行状况
- 使用亚马逊监控 Neptune CloudWatch
- 将审计日志用于 Amazon Neptune 集群
- 将 Neptune 日志发布到亚马逊日志 CloudWatch
- 为 Neptune 笔记本启用 Amazon CloudWatch 日志

- 使用 Amazon Neptune 慢速查询日志记录
- 使用记录亚马逊 Neptune API 调用 AWS CloudTrail
- 使用 Neptune 事件通知
- 给 Amazon Neptune 资源加标签

检查 Neptune 实例的运行状况

Amazon Neptune 提供了一种用于检查主机上的图形数据库状态的机制。这也是确认您能够连接到实例的好方法。

使用 `curl` 检查实例的运行状况并获取数据库集群状态：

```
curl -G https://your-neptune-endpoint:port/status
```

或者，从引擎 1.2.1.0.R6 版本开始，您可以改用以下 CLI 命令：

```
aws neptunedata get-engine-status
```

如果实例运行正常，则 `status` 命令会返回具有以下字段的 JSON 对象：

- **`status`** – 如果实例没有遇到问题，则设置为 `"healthy"`。

  如果实例从崩溃中恢复或者进行了重启，并且最近一次服务器关闭时仍有活动的事务在运行，则 `status` 设置为 `"recovery"`。
- **`startTime`** – 设置为当前服务器进程启动的 UTC 时间。
- **`dbEngineVersion`** – 设置为在数据库集群上运行的 Neptune 引擎版本。

  如果此引擎版本发布以来已手动安装补丁程序，则版本号添加前缀 `"Patch-"`。
- **`role`** – 如果实例为只读副本，则设置为 `"reader"`；如果实例为主实例，则设置为 `"writer"`。
- **`dfeQueryEngine`** – 如果 DFE 引擎已完全启用，则设置为 `"enabled"`；或者如果 DFE 引擎仅用于 `useDFE` 查询提示设置为 `true`（`viaQueryHint` 为默认值）的查询，则设置为 `viaQueryHint`。
- **`gremlin`** – 包含有关您的集群上可用的 Gremlin 查询语言的信息。具体而言，它包含一个`version`字段，用于指定引擎正在使用的当前 TinkerPop 版本。
- **`sparql`** – 包含有关您的集群上可用的 SPARQL 查询语言的信息。具体而言，它包含一个 `version` 字段，用于指定引擎正在使用的当前 SPARQL 版本。

- **opencypher** – 包含有关您的集群上可用的 openCypher 查询语言的信息。具体而言，它包含一个 `version` 字段，用于指定引擎正在使用的当前 OperCypher 版本。
- **labMode** – 包含引擎正在使用的 实验室模式 设置。
- **rollingBackTrxCount** – 如果有事务正在回滚，则此字段将设置为此类事务的数量。如果没有，则该字段根本不会出现。
- **rollingBackTrxEarliestStartTime** – 设置为正在回滚的最早事务的开始时间。如果没有回滚任何事务，则该字段根本不会出现。
- **features** – 包含有关在您的数据库集群上启用的特征的状态信息：
  - **lookupCache** – 查找缓存 的当前状态。此字段仅出现在 R5d 实例类型上，因为这些是可能存在查找缓存的仅有实例。该字段是采用以下形式的 JSON 对象：

    ```
    "lookupCache": {
      "status": "current lookup cache status"
    }
    ```

    在 R5d 实例上：
    - 如果启用了查找缓存，则状态将列为 `"Available"`。
    - 如果已禁用查找缓存，则状态将列为 `"Disabled"`。
    - 如果已达到实例上的磁盘限制，则状态将列为 `"Read Only Mode - Storage Limit Reached"`。
  - **ResultCache** – 缓存查询结果 的当前状态。此字段是采用以下形式的 JSON 对象：

    ```
    "ResultCache": {
      "status": "current results cache status"
    }
    ```

    - 如果已启用结果缓存，则状态将列为 `"Available"`。
    - 如果缓存已禁用，则状态将列为 `"Disabled"`。
  - **IAMAuthentication**— 指定是否已在数据库集群上启用 AWS Identity and Access Management (IAM) 身份验证：
    - 如果启用了 IAM 身份验证，则状态将列为 `"enabled"`。
    - 如果禁用 IAM 身份验证，则状态将列为 `"disabled"`。
  - **Streams** – 指定您的数据库集群上是否已启用 Neptune 流：
    - 如果启用了流，则状态将列为 `"enabled"`。

    - 如果流已禁用，则状态将列为 "disabled"。
  - **AuditLog** – 如果启用了审计日志，则等于 enabled，否则为 disabled。
  - **SlowQueryLogs** – 如果启用了慢速查询日志记录，则等于 info 或 debug，否则为 disabled。
  - **QueryTimeout** – 查询超时的值（以毫秒为单位）。
- **settings** – 应用于实例的设置：
  - **clusterQueryTimeoutInMs** – 为整个集群设置的查询超时值（以毫秒为单位）。
  - **SlowQueryLogsThreshold** – 为整个集群设置的查询超时值（以毫秒为单位）。
- **serverlessConfiguration** – 如果集群以无服务器模式运行，则为该集群的无服务器设置：
  - **minCapacity** – 数据库集群中无服务器实例可以缩小到的最小大小，以 Neptune 容量单位 (NCU) 为单位。
  - **maxCapacity** – 数据库集群中无服务器实例可以增长到的最大大小，以 Neptune 容量单位 (NCU) 为单位。

实例状态命令的输出示例

以下是实例状态命令（在本例中，在 R5d 实例上运行）的输出示例：

```
{
  'status': 'healthy',
  'startTime': 'Thu Aug 24 21:47:12 UTC 2023',
  'dbEngineVersion': '1.2.1.0.R4',
  'role': 'writer',
  'dfeQueryEngine': 'viaQueryHint',
  'gremlin': {'version': 'tinkerpop-3.6.2'},
  'sparql': {'version': 'sparql-1.1'},
  'opencypher': {'version': 'Neptune-9.0.20190305-1.0'},
  'labMode': {
    'ObjectIndex': 'disabled',
    'ReadWriteConflictDetection': 'enabled'
  },
  'features': {
    'SlowQueryLogs': 'disabled',
    'ResultCache': {'status': 'disabled'},
    'IAMAuthentication': 'disabled',
    'Streams': 'disabled',
    'AuditLog': 'disabled'
  },
```

```
  'settings': {
    'clusterQueryTimeoutInMs': '120000',
    'SlowQueryLogsThreshold': '5000'
  },
  'serverlessConfiguration': {
    'minCapacity': '1.0',
    'maxCapacity': '128.0'
  }
}
```

如果实例出现问题，此状态命令将返回 HTTP 500 错误代码。如果主机无法访问，此请求将超时。请确保您正在从 Virtual Private Cloud (VPC) 内访问实例并且您的安全组允许对其访问。

使用亚马逊监控 Neptune CloudWatch

Amazon Neptune 和亚马逊 CloudWatch 是集成在一起的，因此您可以收集和分析绩效指标。您可以使用 CloudWatch 控制台、 AWS Command Line Interface (AWS CLI) 或 CloudWatch API 监控这些指标。

CloudWatch 还允许您设置警报，以便在指标值违反您指定的阈值时收到通知。您甚至可以设置 CloudWatch 事件，以便在发生违规行为时采取纠正措施。有关使用 CloudWatch 和警报的更多信息，请参阅CloudWatch 文档。

主题

- 查看 CloudWatch 数据（控制台）
- 查看 CloudWatch 数据 (AWS CLI)
- 查看 CloudWatch 数据 (API)
- CloudWatch 用于监控 Neptune 中的数据库实例性能
- Neptun CloudWatch e 指标
- 海王星尺寸 CloudWatch

查看 CloudWatch 数据（控制台）

查看 Neptune 集群 CloudWatch 的数据（控制台）

1. 登录 AWS Management Console 并打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。

2. 在导航窗格中，选择指标。
3. 在 “所有指标” 窗格中，选择 Neptune，然后选择 DB。ClusterIdentifier
4. 在上方窗格中，向下滚动以查看集群的完整指标列表。查看列表会显示可用的 Neptune 指标选项。

要选择或取消选择单个指标，请在结果窗格中选中资源名称和指标旁边的复选框。控制台底部显示所选项目指标的图形。要了解有关 CloudWatch 图表的更多信息，请参阅 Amazon CloudWatch 用户指南中的图表指标。

查看 CloudWatch 数据 (AWS CLI)

查看 Neptune 集群 CloudWatch 的数据 ()AWS CLI

1. 安装 AWS CLI。有关说明，请参阅 AWS Command Line Interface 用户指南。
2. 使用 AWS CLI 来获取信息。中列出了海王星的相关 CloudWatch 参数。Neptun CloudWatch e 指标

   以下示例检索集 CloudWatch 群每秒 Gremlin 请求数的指标。`gremlin-cluster`

```
aws cloudwatch get-metric-statistics \
    --namespace AWS/Neptune  --metric-name GremlinRequestsPerSec \
    --dimensions Name=DBClusterIdentifier,Value=gremlin-cluster \
    --start-time 2018-03-03T00:00:00Z --end-time 2018-03-04T00:00:00Z \
    --period 60 --statistics=Average
```

查看 CloudWatch 数据 (API)

CloudWatch 还支持一项`Query`操作，以便您可以通过编程方式请求信息。有关更多信息，请参阅CloudWatch 查询 API 文档和 Amazon CloudWatch API 参考。

当 CloudWatch 操作需要特定于 Neptune 监控的参数时（例如）`MetricName`，请使用中列出的值。Neptun CloudWatch e 指标

以下示例显示了使用以下参数的低级 CloudWatch 请求：

- `Statistics.member.1 = Average`
- `Dimensions.member.1 = DBClusterIdentifier=gremlin-cluster`

- Namespace = AWS/Neptune
- StartTime = 2013-11-14T00:00:00Z
- EndTime = 2013-11-16T00:00:00Z
- Period = 60
- MetricName = GremlinRequestsPerSec

以下是 CloudWatch 请求的样子。但是，这只是为了说明该请求的形式；您必须基于自己的指标和时间范围来构造自己的请求。

```
https://monitoring.amazonaws.com/
      ?SignatureVersion=2
      &Action=GremlinRequestsPerSec
      &Version=2010-08-01
      &StartTime=2018-03-03T00:00:00
      &EndTime=2018-03-04T00:00:00
      &Period=60
      &Statistics.member.1=Average
      &Dimensions.member.1=DBClusterIdentifier=gremlin-cluster
      &Namespace=AWS/Neptune
      &MetricName=GremlinRequests
      &Timestamp=2018-03-04T17%3A48%3A21.746Z
      &AWSAccessKeyId=AWS Access Key ID;
      &Signature=signature
```

CloudWatch 用于监控 Neptune 中的数据库实例性能

您可以使用 Neptune 中的 CloudWatch 指标来监控数据库实例上发生的情况，并跟踪数据库观察到的查询队列长度。以下指标特别有用：

- **CPUUtilization** – 显示 CPU 利用率的百分率。
- **VolumeWriteIOPs** – 显示集群卷的磁盘 I/O 写入的平均数量，每隔 5 分钟报告一次。
- **MainRequestQueuePendingRequests** – 显示在输入队列中等待执行的请求数。

还可以使用带有 includeWaiting 参数的 Gremlin 查询状态端点来了解服务器上有多少个待处理的请求。这将为您提供所有正在等待的查询的状态。

以下指标可以帮助您调整 Neptune 预调配和查询策略，以提高效率和性能：

- 一致的延迟、高 CPUUtilization、高 VolumeWriteIOPs 和低 MainRequestQueuePendingRequests 结合在一起表明，服务器正在以可持续的速率积极处理并发写入请求，几乎没有 I/O 等待。
- 一致的延迟、低 CPUUtilization、低 VolumeWriteIOPs 和无 MainRequestQueuePendingRequests 结合在一起表明，主数据库实例上有多余的容量用于处理写入请求。
- 高 CPUUtilization 和高 VolumeWriteIOPs 但延迟和 MainRequestQueuePendingRequests 各不相同结合在一起表明，您发送的工作量超过了服务器在给定间隔内可以处理的工作量。考虑创建批量请求或调整批量请求的大小，以便以更少的事务开销完成相同的工作量和/或扩展主实例的规模，从而增加能够同时处理写入请求的查询线程的数量。
- 低 CPUUtilization 以及高 VolumeWriteIOPs 表明，查询线程正在等待到存储层的 I/O 操作完成。如果您看到延迟可变而 MainRequestQueuePendingRequests 有所增加，请考虑创建批量请求或调整批量请求的大小，以便在减少事务开销的情况下完成相同的工作量。

Neptun CloudWatch e 指标

> **Note**
>
> 只有当指标的值不为零时，Amazon Neptune CloudWatch 才会向其发送这些指标。
> 对于所有 Neptune 指标，聚合粒度为 5 分钟。

主题

Neptun CloudWatch e 指标

下表列出了 Neptune 支持的 CloudWatch 指标。

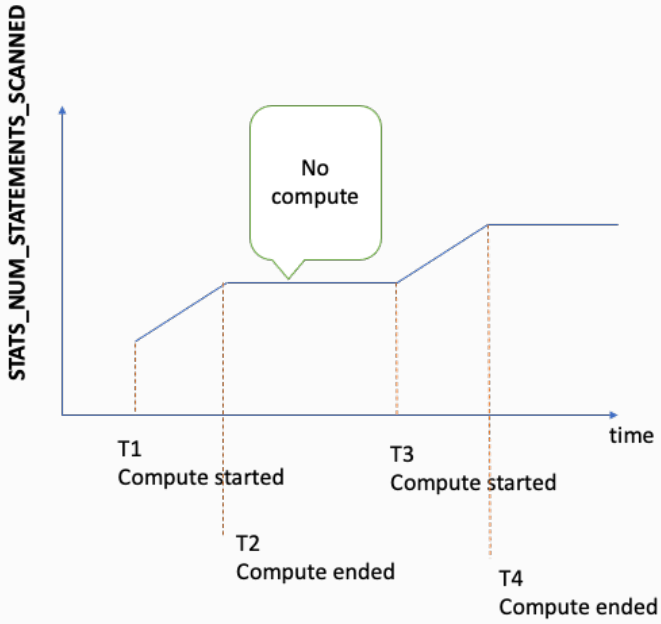
> **Note**
>
> 每当服务器重启时，无论是维护、重启还是从崩溃中恢复，所有累积指标都会重置为零。

Neptun CloudWatch e 指标

指标	描述
BackupRetentionPeriodStorageUsed	为支持 Neptune 数据库集群的备份保留期而使用的备份存储总量（以字节为单位）。包含在 TotalBackupStorageBilled 指标报告的总数中。
BufferCacheHitRatio	缓冲区缓存提供的请求的百分比。此指标可用于诊断查询延迟，因为缓存未命中会导致严重的延迟。如果缓存命中率低于 99.9%，请考虑升级实例类型以在内存中缓存更多数据。
ClusterReplicaLag	对于只读副本，从主实例中复制更新时的滞后总量 (以毫秒为单位)。
ClusterReplicaLagMaximum	数据库集群中主实例和每个 Neptune 数据库实例之间的最大滞后量（以毫秒为单位）。
ClusterReplicaLagMinimum	数据库集群中主实例和每个 Neptune 数据库实例之间的最小滞后量（以毫秒为单位）。
CPUUtilization	CPU 使用百分率。
EngineUptime	实例运行时间长度 (以秒为单位)。
FreeableMemory	随机存取内存的可用量 (以字节为单位)。
GlobalDbDataTransferBytes	Neptune 全局数据库中从主数据库传输 AWS 区域 到辅助 AWS 区域 数据库的重做日志数据的字节数。
GlobalDbReplicatedWriteIO	这是在全球数据库中从主要 AWS 区域 复制到辅助 AWS 区域中的集群卷的写入 I/O 操作数。 Neptune 全球数据库中每个数据库集群的账单计算使用 VolumeWriteIOPS 指标来考虑在该集群中执行的写入。对于主数据库集群，账单计算

指标	描述
	使用 GlobalDbReplicatedWriteIO 考虑向辅助数据库集群的跨区域复制。
GlobalDbProgressLag	对于用户事务和系统事务，辅助集群落后于主集群的毫秒数。
GremlinRequestsPerSec	每秒对 Gremlin 引擎的请求数。
GremlinWebSocketOpenConnections	与 Neptune 的打开 WebSocket 连接数。
LoaderRequestsPerSec	每秒的加载程序请求数。
MainRequestQueuePendingRequests	在输入队列中等待执行的请求数。当请求超过最大队列容量时，Neptune 会开始限制请求。
NCUUtilization	仅适用于 Neptune 无服务器数据库实例或数据库集群。在实例级别，报告按相关实例当前使用的 Neptune 容量单位 (NCU) 数量除以集群的最大 NCU 容量设置计算得出的百分比。NCU 或 Neptune 容量单位由 2GiB 的内存 (RAM) 以及相关的虚拟处理器容量 (vCPU) 和网络组成。 在集群级别，NCUUtilization 报告整个集群使用的容量占最大容量的百分比。
NetworkThroughput	Neptune 数据库集群中每个实例从客户端接收和发送到客户端的网络吞吐量（以每秒字节数为单位）。此吞吐量不 包括数据库集群中的实例与集群卷之间的网络流量。
NetworkTransmitThroughput	Neptune 数据库集群中每个实例发送到客户端的传出网络吞吐量（以每秒字节数为单位）。此吞吐量不 包括数据库集群中的实例与集群卷之间的网络流量。
NumTxCommitted	每秒成功提交的事务数。
NumTxOpened	每秒在服务器上打开的事务数。

指标	描述
NumTxRolledBack	对于写入查询，指的是由于错误每秒在服务器上回滚的事务数。对于只读查询，此指标等于每秒完成的只读事务数。
OpenCypherRequestsPerSec	每秒向 openCypher 引擎发出的请求数（包括 HTTPS 和 Bolt）。
OpenCypherBoltOpenConnections	打开的到 Neptune 的 Bolt 连接数。
ServerlessDatabaseCapacity	作为实例级指标，ServerlessDatabaseCapacity 报告给定 Neptune 无服务器实例的当前实例容量（以 NCU 为单位）。NCU 或 Neptune 容量单位由 2GiB 的内存 (RAM) 以及相关的虚拟处理器容量 (vCPU) 和网络组成。 在集群级，ServerlessDatabaseCapacity 报告集群中数据库实例的所有 ServerlessDatabaseCapacity 值的平均值。
SnapshotStorageUsed	Neptune 数据库集群的所有快照在其备份保留期之外消耗的备份存储总量（以字节为单位）。包含在 TotalBackupStorageBilled 指标报告的总数中。
SparqlRequestsPerSec	每秒对 SPARQL 引擎的请求数。

指标	描述
StatsNumStatementsScanned	自服务器启动以来扫描 DFE 统计数据的语句总数。 每次触发统计数据计算时，这个数字都会增加，但当没有进行计算时，它会保持静态。因此，如果您将它绘制成一段时间内的曲线，您可以分辨出计算何时发生，什么时候没有：  通过查看指标增加期间图形的斜率，您还可以判断计算的速度有多快。 如果没有这样的指标，则意味着您的数据库集群上的统计数据特征已禁用，或者您正在运行的引擎版本没有统计数据特征。如果指标值为零，则表示未进行统计数据计算。
TotalBackupStorageBilled	给定 Neptune 数据库集群的计费备份存储总量（以字节为单位）。包含由 BackupRetentionPeriodStorageUsed 和 SnapshotStorageUsed 指标度量的备份存储。

指标	描述
TotalRequestsPerSec	每秒从所有源到服务器的请求的总数。
TotalClientErrorsPerSec	每秒因客户端问题而导致错误的请求的总数。
TotalServerErrorsPerSec	每秒因内部故障而在服务器端出错的请求的总数。
UndoLogListSize	撤消日志列表中撤消日志的计数。 撤消日志包含已提交事务的记录，当所有活动的事务都比提交时间更晚时，这些已提交事务就会过期。过期的记录会定期清除。删除操作记录的清除时间可能比其它类型事务的记录要长。 清除完全由数据库集群的写入器实例完成，因此清除速率取决于写入器实例类型。如果数据库集群中的 UndoLogListSize 很高且不断增长，请升级写入器实例以提高清除速率。 此外，如果您要从早于 1.2.0.0 的版本升级到引擎版本 1.2.0.0 或更高版本，请先确保 UndoLogListSize 值接近 0。由于引擎版本 1.2.0.0 及更高版本对撤消日志使用不同的格式，因此只有在完全清除之前的撤消日志之后，才能开始升级。请参阅升级到 1.2.0.0 或更高版本了解更多信息。
VolumeBytesUsed	分配给 Neptune 数据库集群的总存储量（以字节为单位）。这是要计费的存储量。它是在数据库集群存在期间的任何时间点分配给该集群的最大存储量，而不是您当前使用的存储量（请参阅Neptune 存储账单）。
VolumeReadIOPs	从集群卷中计费的读取 I/O 操作总数，报告间隔为 5 分钟。计费读取操作数是在集群卷级别计算的，从 Neptune 数据库集群中的所有实例聚合而来，然后每隔 5 分钟报告一次。

指标	描述
VolumeWriteIOPs	向集群卷写入磁盘 I/O 操作的总数，每隔 5 分钟报告一次。

CloudWatch Neptune 中现已弃用的指标

以下 Neptune 指标现在已弃用。它们仍然受支持，但随着新的更好指标出现，将来可能会被淘汰。

指标	描述
GremlinHttp1xx	Gremlin 终端节点每秒的 HTTP 1xx 响应的数量。 我们建议您改用新的 Http1xx 组合指标。
GremlinHttp2xx	Gremlin 终端节点每秒的 HTTP 2xx 响应的数量。 我们建议您改用新的 Http2xx 组合指标。
GremlinHttp4xx	Gremlin 终端节点每秒的 HTTP 4xx 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
GremlinHttp5xx	Gremlin 终端节点每秒的 HTTP 5xx 错误的数量。 我们建议您改用新的 Http5xx 组合指标。
GremlinErrors	Gremlin 遍历中的错误数量。
GremlinRequests	对 Gremlin 引擎的请求数。
GremlinWebSocketSuccess	每秒成功 WebSocket 连接到 Gremlin 端点的次数。

指标	描述
GremlinWebSocketClientErrors	每秒 Gremlin 端点上的 WebSocket 客户端错误数。
GremlinWebSocketServerErrors	每秒 Gremlin 端点上的 WebSocket 服务器错误数。
GremlinWebSocketAvailableConnections	当前可用的潜在 WebSocket 连接数。
Http100	终端节点每秒的 HTTP 100 响应的数量。 我们建议您改用新的 Http1xx 组合指标。
Http101	终端节点每秒的 HTTP 101 响应的数量。 我们建议您改用新的 Http1xx 组合指标。
Http1xx	终端节点每秒的 HTTP 1xx 响应的数量。
Http200	终端节点每秒的 HTTP 200 响应的数量。 我们建议您改用新的 Http2xx 组合指标。
Http2xx	终端节点每秒的 HTTP 2xx 响应的数量。
Http400	终端节点每秒的 HTTP 400 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
Http403	终端节点每秒的 HTTP 403 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
Http405	终端节点每秒的 HTTP 405 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
Http413	终端节点每秒的 HTTP 413 错误的数量。 我们建议您改用新的 Http4xx 组合指标。

指标	描述
Http429	终端节点每秒的 HTTP 429 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
Http4xx	终端节点每秒的 HTTP 4xx 错误的数量。
Http500	终端节点每秒的 HTTP 500 错误的数量。 我们建议您改用新的 Http5xx 组合指标。
Http501	终端节点每秒的 HTTP 501 错误的数量。 我们建议您改用新的 Http5xx 组合指标。
Http5xx	终端节点每秒的 HTTP 5xx 错误的数量。
LoaderErrors	来自加载程序请求的错误数。
LoaderRequests	加载程序请求数。
SparqlHttp1xx	SPARQL 终端节点每秒的 HTTP 1xx 响应的数量。 我们建议您改用新的 Http1xx 组合指标。
SparqlHttp2xx	SPARQL 终端节点每秒的 HTTP 2xx 响应的数量。 我们建议您改用新的 Http2xx 组合指标。
SparqlHttp4xx	SPARQL 终端节点每秒的 HTTP 4xx 错误的数量。 我们建议您改用新的 Http4xx 组合指标。
SparqlHttp5xx	SPARQL 终端节点每秒的 HTTP 5xx 错误的数量。 我们建议您改用新的 Http5xx 组合指标。

指标	描述
SparqlErrors	SPARQL 查询中的错误数。
SparqlRequests	对 SPARQL 引擎的请求数。
StatusErrors	来自状态终端节点的错误数。
StatusRequests	对状态终端节点的请求数。

海王星尺寸 CloudWatch

Amazon Neptune 的指标由账户、图形名称或操作进行限定。您可以使用 Amazon CloudWatch 控制台检索 Neptune 数据以及下表中的任何维度。

维度	描述
DBInstanceIdentifier	筛选您为集群内的特定数据库实例请求的数据。
DBClusterIdentifier	筛选您为特定 Neptune 数据库集群请求的数据。
DBClusterIdentifier ，EngineName	按集群筛选数据。所有 Neptune 实例的引擎名称均为 neptune。
DBClusterIdentifier ，Role	筛选您为特定 Neptune 数据库集群请求的数据，并按实例角色 (WRITER/READER) 聚合指标。例如，您可以聚合属于某个群集的所有 READER 实例的指标。
DBClusterIdentifier ，SourceRegion	按全球数据库主区域中的主集群筛选数据。
DatabaseClass	筛选您为数据库类中的所有实例请求的数据。例如，您可以聚合属于数据库类 db.r4.large 的所有实例的指标
EngineName	所有 Neptune 实例的引擎名称均为 neptune。

维度	描述
`GlobalDbDBClusterIdentifier` ，`SecondaryRegion`	按辅助区域中指定全球数据库的辅助集群筛选数据。

将审计日志用于 Amazon Neptune 集群

要审计 Amazon Neptune 数据库集群活动，请通过设置数据库集群参数来启用审计日志的收集。在启用了审核日志时，您可以用它来记录任意支持事件的组合。您可以查看或者下载审核日志来检查。

启用 Neptune 审计日志

使用 `neptune_enable_audit_log` 参数可启用 (`1`) 或禁用 (`0`) 审核日志。

在由您的数据库集群使用的参数组中设置此参数。您可以使用中编辑数据库集群参数组或数据库参数组显示的过程通过修改参数，也可以使用 m odify-db-cluster-parameter-group 命令或 modifyDB Group API AWS CLI 命令以编程方式修改ClusterParameter参数。 AWS Management Console

修改该参数后，必须重启数据库实例才能应用更改。

使用控制台查看 Neptune 审计日志

您可以使用 AWS Management Console查看并下载审核日志。在实例页面上，选择该数据库实例以显示其详细信息，然后滚动到日志部分。

要下载日志文件，请在日志部分中选择该文件并选择下载。

Neptune 审计日志详细信息

日志文件使用 UTF-8 格式。日志写入到多个文件中，其数量根据实例大小变化。要查看最新事件，您可能需要查看所有审核日志文件。

日志条目不按先后顺序。您可以使用 `timestamp` 值对其进行排序。

日志文件在总和达到 100 MB 时轮换。此限制是无法配置的。

审计日志文件的各行按照以下顺序包含以下逗号分隔的信息：

字段	描述
Timestamp	所记录事件的 Unix 时间戳，精度为微秒。
ClientHost	用户发起连接时所在的主机名或 IP。
ServerHost	记录了其事件的实例的主机名或 IP。
ConnectionType	连接类型。可以是 `Websocket` 、`HTTP_POST` 、`HTTP_GET` 或 `Bolt`。

调用方的 IAM ARN	用于签署请求的 IAM 用户或 IAM 角色的 ARN。如果 IAM 身份验证已禁用，则为空。其格式为：
	`arn:partition :service:region:account:resource`
	例如：
	`arn:aws:iam::123456789012:user/Anna`
	`arn:aws:sts::123456789012:assumed-role/AWSNeptuneNotebookRole/SageMaker`

身份验证上下文	包含身份验证信息的序列化 JSON 对象。如果已对用户进行身份验证，`authenticationSucceeded` 字段将为 `True`。 如果 IAM 身份验证已禁用，则为空。
HttpHeader	HTTP 标头信息。可以包含一个查询。for WebSocket 和 Bolt 连接为空。
有效负载	Gremlin、SPARQL 或 openCypher 查询。

将 Neptune 日志发布到亚马逊日志 CloudWatch

您可以将 Neptune 数据库集群配置为将审核日志数据和/或慢速查询日志数据发布到 Amazon Logs 中的日志组。 CloudWatch 借助 CloudWatch 日志，您可以对日志数据进行实时分析，并 CloudWatch 用于创建警报和查看指标。您可以使用 CloudWatch 日志将日志记录存储在高度耐用的存储中。

要将审核日志发布到 CloudWatch 日志，必须明确启用审核日志（请参阅启用审计日志）。同样，要将慢查询日志发布到日 CloudWatch 志，必须明确启用慢查询日志（请参阅）。使用 Amazon Neptune 慢速查询日志记录

> **Note**
>
> 请注意以下事项：
>
> - 当您将日志发布到时，将收取额外费用 CloudWatch。详情请参阅CloudWatch 定价页面。
> - 您无法将日志发布到中国（北京）或中国（宁夏）区域的 CloudWatch 日志。
> - 如果禁用了导出日志数据，则 Neptune 不会删除现有日志组或日志流。如果禁用导出日志数据，则日志中的 CloudWatch 现有日志数据仍可用，具体取决于日志保留时间，并且您仍会为存储的审核日志数据付费。您可以使用日志控制台、或 Lo CloudWatch gs API 删除日志流和 CloudWatch 日志组。 AWS CLI

使用控制台将 Neptune 日志发布到日志 CloudWatch

从控制台将 Neptune 日志发布到 CloudWatch 日志

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择数据库。
3. 选择要为其发布日志数据的 Neptune 数据库集群。
4. 对于 Actions（操作），选择 Modify（修改）。
5. 在日志导出部分，选择要开始发布到 CloudWatch 日志的日志。
6. 选择继续，然后选择摘要页面上的 修改数据库集群。

使用 CLI 将 Neptune 审核日志发布到日志 CloudWatch

您可以使用带有以下参数的 AWS CLI `create-db-cluster`命令创建新的数据库集群，将审核 CloudWatch 日志发布到日志：

```
aws neptune create-db-cluster \
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --engine neptune \
```

```
    --enable-cloudwatch-logs-exports '["audit"]'
```

您可以使用带有以下参数的 AWS CLI `modify-db-cluster`命令将现有数据库集群配置为将审计 CloudWatch 日志发布到日志：

```
aws neptune modify-db-cluster \
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["audit"]}'
```

使用 CLI 将 Neptune 慢速查询日志发布到日志 CloudWatch

您还可以使用带有以下参数的 AWS CLI `create-db-cluster`命令创建一个新的数据库集群，将慢速查询 CloudWatch 日志发布到日志：

```
aws neptune create-db-cluster \
    --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --engine neptune \
    --enable-cloudwatch-logs-exports '["slowquery"]'
```

同样，您可以使用带有以下参数的 AWS CLI `modify-db-cluster`命令将现有数据库集群配置为将慢查询 CloudWatch 日志发布到 Logs：

```
aws neptune modify-db-cluster --region us-east-1  \
    --db-cluster-identifier my_db_cluster_id \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["slowquery"]}'
```

监控 Amazon 中的 Neptune 日志事件 CloudWatch

启用 Neptune 日志后，您可以在 Ama CloudWatch zon 日志中监控日志事件。将自动使用以下前缀为 Neptune 数据库集群创建新的日志组，其中 *`cluster-name`* 表示数据库集群名称，*`log_type`* 表示日志类型：

```
/aws/neptune/cluster-name/log_type
```

例如，如果您将导出功能配置为包括名为 `mydbcluster` 的数据库集群的审计日志，则日志数据将存储在 `/aws/neptune/mydbcluster/audit` 日志组中。

将使用不同的日志流将来自数据库集群中的所有数据库实例的所有事件推送到一个日志组。

如果存在具有指定名称的日志组，Neptune 将使用该日志组为 Neptune 数据库集群导出日志数据。您可以使用自动配置（例如 AWS CloudFormation）来创建具有预定义日志保留期、指标筛选器和客户访问权限的日志组。否则，将使用日志中的默认日志保留期 “永不过期” 自动创建新的 CloudWatch 日志组。

您可以使用 CloudWatch 日志控制台 AWS CLI、或 Lo CloudWatch gs API 来更改日志保留期。有关在日志中更改日志保留期限的 CloudWatch 更多信息，请参阅更改日志中的 CloudWatch 日志数据保留期。

您可以使用 CloudWatch 日志控制台 AWS CLI、或 L CloudWatch ogs API 在数据库集群的日志事件中搜索信息。有关搜索和筛选日志数据的更多信息，请参阅搜索和筛选日志数据。

为 Neptune 笔记本启用 Amazon CloudWatch 日志

CloudWatch 默认情况下，Neptune 笔记本的日志处于禁用状态。为了调试或其它目的，请按照以下步骤启用它们：

使用启用 Neptune 笔记本的 CloudWatch 日志 AWS Management Console

1. 打开亚马逊 SageMaker 控制台，网址为 https://console.aws.amazon.com/sagemaker/。
2. 在左侧的导航窗格上，选择笔记本，然后选择笔记本实例。查找要为其启用日志的 Neptune 笔记本的名称。
3. 通过选择在上述步骤中提及的笔记本实例的名称，进入详细信息页面。
4. 如果笔记本实例正在运行，请选择笔记本详细信息页面右上角的停止按钮。
5. 在权限和加密下，有一个对应于 IAM 角色 ARN 的字段。选择此字段中的链接可转至此笔记本的 IAM 角色。
6. 创建以下策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogDelivery",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DeleteLogDelivery",
        "logs:Describe*",
```

```
        "logs:GetLogDelivery",
        "logs:GetLogEvents",
        "logs:ListLogDeliveries",
        "logs:PutLogEvents",
        "logs:PutResourcePolicy",
        "logs:UpdateLogDelivery"
      ],
      "Resource": "*"
    }
  ]
}
```

7. 保存此新策略，并将其附加到步骤 4 中的 IAM 角色。
8. 选择 SageMaker笔记本实例详细信息页面右上角的 “启动”。
9. 日志开始流动后，您应该会在详细信息页面笔记本实例设置部分的左下角附近标有生命周期配置的字段下方看到查看日志链接。

如果您的笔记本无法启动，则 SageMaker 主机上的笔记本详细信息页面中将显示一条消息，说明笔记本实例启动时间超过 5 分钟。可以在名称下找到与此问题相关的 CloudWatch 日志:*(your-notebook-name)*/LifecycleConfigOnStart。

如有必要，请参阅 “使用亚马逊 CloudWatch记录亚马逊 SageMaker事件” 了解更多详情。

使用 Amazon Neptune 慢速查询日志记录

识别、调试和优化运行缓慢的查询可能很困难。启用 Neptune 的慢速查询日志记录后，会自动记录所有长时间运行的查询的属性，以简化此过程。

> **Note**
>
> Neptune 引擎版本 1.2.1.0 中引入了慢速查询日志记录。

您可以使用 neptune_enable_slow_query_log 数据库集群参数启用慢速查询日志记录。默认情况下，此参数设置为 `disabled`。将其设置为 `info` 或 `debug` 可启用慢速查询日志记录。`info` 设置记录每个运行缓慢的查询的一些有用属性，而 `debug` 设置则记录所有可用属性。

要为视为运行缓慢的查询设置阈值，请使用 neptune_slow_query_log_threshold 数据库集群参数来指定毫秒数，在此时间之后，正在运行的查询会被视为慢速查询，并在启用慢速查询日志记录时记录下来。默认值为 5000 毫秒（5 秒）。

您可以在中设置这些数据库集群参数 AWS Management Console，也可以使用 modify-db-cluster-parameter -group 命令或 modifyDB 组管理功能设置这些数据库集群参数AWS CLI 。ClusterParameter

> **Note**
>
> 慢速查询日志记录参数是动态的，这意味着更改其值不需要或导致数据库集群重启。

要查看慢查询日志，请访问 AWS Management Console

您可以在中查看和下载慢速查询日志 AWS Management Console，如下所示：

在实例页面上，选择该数据库实例，然后滚动到日志部分。然后，您可以在那里选择一个日志文件，然后选择下载以进行下载。

Neptune 慢速查询日志记录生成的文件

在 Neptune 中通过慢速查询日志记录生成的日志文件具有以下特性：

- 这些文件以 UTF-8 编码。
- 查询及其属性以 JSON 格式记录。
- 除 `queryTime` 数据外，不记录 Null 属性和空属性。
- 日志跨越多个文件，其数量根据实例大小而变化。
- 日志条目不按先后顺序。您可以使用其 `timestamp` 值对它们进行排序。
- 要查看最新事件，您可能必须审核所有慢速查询日志文件。
- 日志文件在总和达到 100MiB 时轮换。此限制是无法配置的。

在 `info` 模式下记录的查询属性

当 `neptune_enable_slow_query_log` 数据库集群参数设置为 `info` 时，会记录慢速查询的以下属性：

组	属性	描述
请求 ResponseMetadata	`requestId`	查询的请求 ID。

组	属性	描述
	requestType	请求类型，比如 HTTP 或 WebSocket。
	responseStatusCode	查询响应状态码，比如 200。
	exceptionClass	查询执行后返回的错误的异常类。
queryStats	query	查询字符串。
	queryFingerprint	查询的指纹。
	queryLanguage	查询语言，如 Gremlin、SPARQL 或 openCypher。
memoryStats	allocatedPermits	分配给查询的许可。
	approximateUsedMemoryBytes	查询在执行期间使用的近似内存量。
queryTime	startTime	查询开始时间 (UTC)。
	overallRunTimeMs	查询总运行时间，以毫秒为单位。
	parsingTimeMs	查询解析时间，以毫秒为单位。
	waitingTimeMs	查询 Gremlin/SPARQL/openCypher 队列等待时间，以毫秒为单位
	executionTimeMs	查询执行时间，以毫秒为单位。
	serializationTimeMs	查询序列化时间，以毫秒为单位。

组	属性	描述
statementCounters	scanned	扫描的语句数量。
	written	写入的语句数量。
	deleted	删除的语句数量。
transactionCounters	committed	已提交的事务数量。
	rolledBack	回滚的事务数量。
vertexCounters	added	添加的顶点数。
	removed	移除的顶点数。
	propertiesAdded	添加的顶点属性的数量。
	propertiesRemoved	移除的顶点属性的数量。
edgeCounters	added	添加的边缘数量。
	removed	移除的边缘数量。
	propertiesAdded	添加的边缘属性的数量。
	propertiesRemoved	移除的边缘属性的数量。
resultCache	hitCount	结果缓存命中计数。
	missCount	结果缓存未命中计数。
	putCount	结果缓存放置计数。
concurrentExecution	acceptedQueryCountAtStart	当前查询执行在开始时接受的并行查询。
	runningQueryCountAtStart	当前查询执行在开始时运行的并行查询。

组	属性	描述
	acceptedQueryCountAtEnd	当前查询执行在结束时接受的并行查询。
	runningQueryCountAtEnd	当前查询执行在结束时运行的并行查询。
queryBatch	queryProcessingBatchSize	查询处理期间的批量大小。
	querySerialisationBatchSize	查询序列化期间的批量大小。

在 **debug** 模式下记录的查询属性

将 neptune_enable_slow_query_log 数据库集群参数设置为 debug 后，除了在 info 模式下记录的属性外，还会记录以下存储计数器属性：

属性	描述
statementsScannedInAllIndexes	在所有索引中扫描的语句。
statementsScannedSPOGIndex	在 SPOG 索引中扫描的语句。
statementsScannedPOGSIndex	在 POGS 索引中扫描的语句。
statementsScannedGPSOIndex	在 GPSO 索引中扫描的语句。
statementsScannedOSGPIndex	在 OSGP 索引中扫描的语句。
statementsScannedInChunk	成块一起扫描的语句。
postFilteredStatementScans	扫描后筛选后留下的语句。
distinctStatementScans	扫描的不同语句。
statementsReadInAllIndexes	在所有索引中进行扫描后筛选后读取的语句。

属性	描述
statementsReadSPOGIndex	在 SPOG 索引中进行扫描后筛选后读取的语句。
statementsReadPOGSIndex	在 POGS 索引中进行扫描后筛选后读取的语句。
statementsReadGPSOIndex	在 GPSO 索引中进行扫描后筛选后读取的语句。
statementsReadOSGPIndex	在 OSGP 索引中进行扫描后筛选后读取的语句。
accessPathSearches	访问路径搜索次数。
fullyBoundedAccessPathSearches	完全限定的密钥访问路径搜索次数。
accessPathSearchedByPrefix	按前缀搜索的访问路径数。
searchesWhereRecordsWereFound	输出 1 条或更多记录的搜索次数。
searchesWhereRecordsWereNotFound	输出中没有记录的搜索次数。
totalRecordsFoundInSearches	从所有搜索中找到的记录总数。
statementsInsertedInAllIndexes	所有索引中插入的语句数。
statementsUpdatedInAllIndexes	所有索引中更新的语句数。
statementsDeletedInAllIndexes	所有索引中删除的语句数。
predicateCount	谓词的数量。
dictionaryReadsFromValueToIdTable	从值到 ID 表的字典读取次数。
dictionaryReadsFromIdToValueTable	从值的 ID 表中读取字典的次数。
dictionaryWritesToValueToIdTable	向值到 ID 表进行字典写入的次数。

属性	描述
dictionaryWritesToIdToValueTable	向 ID 到值表进行字典写入的次数。
rangeCountsInAllIndexes	所有索引中的范围计数数量。
deadlockCount	查询中的死锁数。
singleCardinalityInserts	执行的单基数插入次数。
singleCardinalityInsertDeletions	在单基数插入期间删除的语句数。

慢速查询的调试日志记录示例

以下 Gremlin 查询的运行时间可能比为慢速查询设置的阈值更长：

```
gremlin=g.V().has('code','AUS').repeat(out().simplePath()).until(has('code','AGR')).path().by('
```

然后，如果在调试模式下启用了慢速查询日志记录，则将以这样的形式记录查询的以下属性：

```
{
  "requestResponseMetadata": {
    "requestId": "5311e493-0e98-457e-9131-d250a2ce1e12",
    "requestType": "HTTP_GET",
    "responseStatusCode": 200
  },
  "queryStats": {
    "query":
 "gremlin=g.V().has('code','AUS').repeat(out().simplePath()).until(has('code','AGR')).path().by
    "queryFingerprint":
 "g.V().has(string0,string1).repeat(__.out().simplePath()).until(__.has(string0,string2)).path(
    "queryLanguage": "Gremlin"
  },
  "memoryStats": {
    "allocatedPermits": 20,
    "approximateUsedMemoryBytes": 14838
  },
  "queryTimeStats": {
    "startTime": "23/02/2023 11:42:52.657",
    "overallRunTimeMs": 2249,
    "executionTimeMs": 2229,
```

```
    "serializationTimeMs": 13
  },
  "statementCounters": {
    "read": 69979
  },
  "transactionCounters": {
    "committed": 1
  },
  "concurrentExecutionStats": {
    "acceptedQueryCountAtStart": 1
  },
  "queryBatchStats": {
    "queryProcessingBatchSize": 1000,
    "querySerialisationBatchSize": 1000
  },
  "storageCounters": {
    "statementsScannedInAllIndexes": 69979,
    "statementsScannedSPOGIndex": 44936,
    "statementsScannedPOGSIndex": 4,
    "statementsScannedGPSOIndex": 25039,
    "statementsReadInAllIndexes": 68566,
    "statementsReadSPOGIndex": 43544,
    "statementsReadPOGSIndex": 2,
    "statementsReadGPSOIndex": 25020,
    "accessPathSearches": 27,
    "fullyBoundedAccessPathSearches": 27,
    "dictionaryReadsFromValueToIdTable": 10,
    "dictionaryReadsFromIdToValueTable": 17,
    "rangeCountsInAllIndexes": 4
  }
}
```

使用记录亚马逊 Neptune API 调用 AWS CloudTrail

Amazon Neptune 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在 Neptune 中执行的操作的记录。 CloudTrail 将海王星的 API 调用捕获为事件，包括来自海王星控制台的调用和对海王星 API 的代码调用。

CloudTrail 仅记录 Neptune 管理 API 调用的事件，例如创建实例或集群。如果要审核对图表所做更改，您可以使用审核日志。有关更多信息，请参阅 将审计日志用于 Amazon Neptune 集群。

> **Important**
>
> 亚马逊 Neptune 控制台和 API 调用记录为对亚马逊关系数据库服务 (Amazon RDS) API 的调用。 AWS CLI

如果您创建了跟踪，则可以将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括 Neptune 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Neptune 发出的请求、发出请求的 IP 地址、谁提出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

Neptune 中的信息 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当 Amazon Neptune 中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在自己的 AWS 账户中查看、搜索和下载最近发生的事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录 AWS 账户中的事件，包括 Neptune 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪时，此跟踪应用于所有 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 从多个区域接收 CloudTrail 日志文件和从多个账户接收 CloudTrail 日志文件

如果使用海王星控制台、Neptune 命令行界面或 Neptune SDK API 代表您的 AWS 账户执行操作，则会将该操作 AWS CloudTrail 记录为对 Amazon RDS API 的调用。例如，如果您使用 Neptune 控制台修改数据库实例或调用 m AWS CLIodify-db-instance 命令，则日志会 AWS CloudTrail 显示对 Amazon RDS API modifydb Instance 操作的调用。有关记录的 Neptune API 操作的列表 AWS CloudTrail，请参阅《海王星 API 参考》。

> **Note**
>
> AWS CloudTrail 仅记录 Neptune 管理 API 调用的事件，例如创建实例或集群。如果要审核对图表所做更改，您可以使用审核日志。有关更多信息，请参阅 将审计日志用于 Amazon Neptune 集群。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。

了解 Neptune 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序出现。

以下示例显示了一个用户的 CloudTrail 日志，该用户创建了数据库实例的快照，然后使用 Neptune 控制台删除了该实例。控制台由 `userAgent` 元素标识。控制台 (`CreateDBSnapshot` 和 `DeleteDBInstance`) 请求的 API 调用可在每条记录的 `eventName` 元素中找到。有关用户（`Alice`）的信息可在 `userIdentity` 元素中找到。

```
{
  Records:[
  {
    "awsRegion":"us-west-2",
    "eventName":"CreateDBSnapshot",
    "eventSource":"",
    "eventTime":"2014-01-14T16:23:49Z",
    "eventVersion":"1.0",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS Console, aws-sdk-java\/unknown-version Linux\/2.6.18-
kaos_fleet-1108-prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\/24.45-b08",
    "userIdentity":
    {
```

```
      "accessKeyId":"",
      "accountId":"123456789012",
      "arn":"arn:aws:iam::123456789012:user/Alice",
      "principalId":"AIDAI2JXM4FBZZEXAMPLE",
      "sessionContext":
      {
        "attributes":
        {
          "creationDate":"2014-01-14T15:55:59Z",
          "mfaAuthenticated":false
        }
      },
      "type":"IAMUser",
      "userName":"Alice"
    }
  },
  {
    "awsRegion":"us-west-2",
    "eventName":"DeleteDBInstance",
    "eventSource":"",
    "eventTime":"2014-01-14T16:28:27Z",
    "eventVersion":"1.0",
    "sourceIPAddress":"192.0.2.01",
    "userAgent":"AWS Console, aws-sdk-java\/unknown-version Linux\/2.6.18-
kaos_fleet-1108-prod.2 Java_HotSpot(TM)_64-Bit_Server_VM\/24.45-b08",
    "userIdentity":
    {
      "accessKeyId":"",
      "accountId":"123456789012",
      "arn":"arn:aws:iam::123456789012:user/Alice",
      "principalId":"AIDAI2JXM4FBZZEXAMPLE",
      "sessionContext":
      {
        "attributes":
        {
          "creationDate":"2014-01-14T15:55:59Z",
          "mfaAuthenticated":false
        }
      },
      "type":"IAMUser",
      "userName":"Alice"
    }
  }
  ]
```

```
}
```

使用 Neptune 事件通知

主题

Amazon Neptune 使用 Amazon Simple Notification Service (Amazon SNS) 在发生 Neptune 事件时提供通知。这些通知可以采用某一 AWS 地区的 Amazon SNS 支持的任何形式，例如电子邮件、短信或对 HTTP 终端节点的调用。

Neptune 将这些事件分组为您可以订阅的类别，以便您在该类别事件出现时收取通知。您可以针对数据库实例、数据库集群、数据库快照、数据库集群快照或数据库参数组订阅事件类别。例如，如果您订阅给定数据库实例的 Backup 类别，那么无论何时出现影响该数据库实例的备份相关事件，您都将收到通知。您还将在事件通知订阅更改时收到通知。

事件在数据库集群和数据库实例级别发生，因此，如果您订阅数据库集群或数据库实例，将可收到事件。

事件通知会发送到您在创建订阅时提供的地址。您可能希望创建多个不同的订阅，如使用一个订阅接收所有事件通知，并使用另一个订阅仅接收针对生产数据库实例的关键事件。您无需删除订阅即可轻松关闭通知。为此，请在 Neptune 控制台中将启用单选按钮设置为否。

> **Important**
>
> Amazon Neptune 不保证在事件流中发送的事件的顺序。事件顺序可能会发生变化。

Neptune 使用 Amazon SNS 主题的 Amazon 资源名称 (ARN) 标识每个订阅。Neptune 控制台在您创建订阅时为您创建 ARN。

Neptune 事件通知的计费是通过 Amazon SNS 执行的。Amazon SNS 费用在使用事件通知时适用。有关更多信息，请参阅 Amazon Simple Notification Service 定价。

Amazon Neptune 事件类别和事件消息

Neptune 会以各种类别生成许多事件，您可以使用 Neptune 控制台订阅这些类别。每个类别应用于一种源类型，源类型可以是数据库实例、数据库快照或数据库参数组。

> **Note**
>
> Neptune 使用现有 Amazon RDS 事件定义和 ID。

源自数据库实例的 Neptune 事件

下表显示了当数据库实例为源类型时按事件类别列出的事件列表。

类别	Amazon RDS 事件 ID	描述
可用性	RDS-EVENT-0006	数据库实例已重启。
	RDS-EVENT-0004	数据库实例已关闭。
	RDS-EVENT-0022	重启 Neptune 引擎时出现了错误。
备份	RDS-EVENT-0001	备份数据库实例。
	RDS-EVENT-0002	已完成数据库实例备份。
配置更改	RDS-EVENT-0009	数据库实例已添加到安全组。
	RDS-EVENT-0024	数据库实例正在转换为多可用区数据库实例。
	RDS-EVENT-0030	数据库实例正在转换为单可用区数据库实例。

类别	Amazon RDS 事件 ID	描述
	RDS-EVENT-0012	将修改应用于数据库实例类。
	RDS-EVENT-0018	正在更改此数据库实例的当前存储设置。
	RDS-EVENT-0011	已更改此数据库实例的参数组。
	RDS-EVENT-0092	此数据库实例的参数组已完成更新。
	RDS-EVENT-0028	已禁用此数据库实例的自动备份。
	RDS-EVENT-0032	已启用此数据库实例的自动备份。
	RDS-EVENT-0025	数据库实例已转换为多可用区数据库实例。
	RDS-EVENT-0029	数据库实例已转换为单可用区数据库实例。
	RDS-EVENT-0014	已更改此数据库实例的数据库实例类。
	RDS-EVENT-0017	已更改此数据库实例的存储设置。
	RDS-EVENT-0010	数据库实例已从安全组删除。
创建	RDS-EVENT-0005	数据库实例已创建。
删除	RDS-EVENT-0003	数据库实例已删除。

类别	Amazon RDS 事件 ID	描述
故障转移	RDS-EVENT-0034	Neptune 不会尝试所请求的失效转移，因为数据库实例上最近发生了失效转移。
	RDS-EVENT-0013	已启用可以提升备用实例性能的多可用区故障转移。
	RDS-EVENT-0015	已完成可以提升备用实例性能的多可用区故障转移。可能需要几分钟的时间才能让 DNS 传输到新的主数据库实例。
	RDS-EVENT-0065	实例已从部分故障转移恢复。
	RDS-EVENT-0049	多可用区故障转移已完成。
	RDS-EVENT-0050	多可用区激活已在成功还原实例后开始。
	RDS-EVENT-0051	多可用区激活已完成。现在，应该可以访问您的数据库了。
	RDS-EVENT-0031	由于某个不兼容配置或底层存储问题，数据库实例已失败。从 point-in-time-restore 数据库实例开始。

类别	Amazon RDS 事件 ID	描述
	RDS-EVENT-0036	数据库实例处于不兼容的网络中。有些指定的子网 ID 无效或者不存在。
	RDS-EVENT-0035	数据库实例有无效参数。例如，如果数据库实例因为此实例类的内存相关参数设置过高而无法启动，客户就应该修改内存参数，并重新启动数据库实例。
	RDS-EVENT-0082	Neptune 无法从 Amazon S3 桶复制备份数据。很可能是用来访问 Amazon S3 桶的 Neptune 权限配置不正确。
存储不足	RDS-EVENT-0089	数据库实例已使用其分配的存储空间的 90% 以上。您可以使用可用存储空间指标监控数据库实例的存储空间。
	RDS-EVENT-0007	分配的数据库实例存储空间已用完。要解决此问题，您应该为数据库实例分配额外存储。

类别	Amazon RDS 事件 ID	描述
维护	RDS-EVENT-0026	正在进行数据库实例的脱机维护。数据库实例目前无法使用。
	RDS-EVENT-0027	数据库实例的脱机维护已完成。现在可以使用数据库实例。
	RDS-EVENT-0047	数据库实例的修补已完成。
通知	RDS-EVENT-0044	操作员发出的通知。有关详细信息，请参阅事件消息。
	RDS-EVENT-0048	数据库实例的修补已延迟。
	RDS-EVENT-0087	已停止数据库实例。
	RDS-EVENT-0088	已启动数据库实例。
	RDS-EVENT-0154	数据库实例将由于它超过最大允许停止的时间而正被启动。
	RDS-EVENT-0158	数据库实例处于无法升级的状态。
	RDS-EVENT-0173	数据库实例已修补。
只读副本	RDS-EVENT-0045	在读取复制过程中出错。有关详细信息，请参阅事件消息。

类别	Amazon RDS 事件 ID	描述
	RDS-EVENT-0046	只读副本已恢复复制。此消息会在您首次创建只读副本时出现，或显示为确认复制在正常运行的监控消息。如果此消息在 RDS-EVENT-0045 通知之后出现，则复制已在出现错误之后或是停止复制之后进行了恢复。
	RDS-EVENT-0057	只读副本上的复制已终止。
	RDS-EVENT-0062	只读副本上的复制已手动停止。
	RDS-EVENT-0063	只读副本上的复制已重置。
恢复	RDS-EVENT-0020	已启动数据库实例的还原。恢复时间会随待恢复数据量的变化而变化。
	RDS-EVENT-0021	数据库实例的恢复已完成。
	RDS-EVENT-0023	已请求手动备份，但 Neptune 目前处于创建数据库快照的过程中。请在 Neptune 完成数据库快照后再次提交请求。

类别	Amazon RDS 事件 ID	描述
	RDS-EVENT-0052	已启动多可用区实例的恢复。恢复时间会随待恢复数据量的变化而变化。
	RDS-EVENT-0053	多可用区实例的恢复已完成。
还原	RDS-EVENT-0008	已从数据库快照中恢复数据库实例。
	RDS-EVENT-0019	数据库实例已从 point-in-time备份中恢复。

源自数据库集群的 Neptune 事件

下表显示了当数据库集群为源类型时按事件类别列出的事件列表。

类别	RDS 事件 ID	描述
故障转移	RDS-EVENT-0069	数据库群集的故障转移已失败。
	RDS-EVENT-0070	数据库群集的故障转移已重新启动。
	RDS-EVENT-0071	数据库群集的故障转移已完成。
	RDS-EVENT-0072	数据库群集的故障转移已在同一可用区内开始。
	RDS-EVENT-0073	数据库群集的故障转移已跨可用区开始。

类别	RDS 事件 ID	描述
	RDS-EVENT-0083	Neptune 无法从 Amazon S3 桶复制备份数据。很可能是用来访问 Amazon S3 桶的 Neptune 权限配置不正确。
maintenance	RDS-EVENT-0156	数据库集群具有数据库引擎次要版本升级。
通知	RDS-EVENT-0076	迁移到 Neptune 数据库集群失败。
	RDS-EVENT-0077	在迁移到 Neptune 数据库集群的过程中，尝试将表从源数据库转换为数据库形式失败。
	RDS-EVENT-0150	已停止数据库集群。
	RDS-EVENT-0151	已启动数据库集群。
	RDS-EVENT-0152	数据库集群停止失败。
	RDS-EVENT-0153	数据库集群将由于它超过最大允许停止的时间而正被启动。

源自数据库集群快照的 Neptune 事件

下表显示了 Neptune 数据库集群快照为源类型时的事件类别和事件列表。

类别	RDS 事件 ID	描述
backup	RDS-EVENT-0074	手动数据库群集快照的创建已开始。
backup	RDS-EVENT-0075	已创建手动数据库集群快照。
通知	RDS-EVENT-0162	数据库集群快照导出任务失败。
通知	RDS-EVENT-0163	数据库集群快照导出任务已取消。
通知	RDS-EVENT-0164	数据库集群快照导出任务已完成。
backup	RDS-EVENT-0168	正在创建自动集群快照。
backup	RDS-EVENT-0169	已创建自动集群快照。
创建	RDS-EVENT-0170	已创建数据库集群。
删除	RDS-EVENT-0171	已删除数据库集群。
通知	RDS-EVENT-0172	将数据库集群从 [旧数据库集群名称] 重命名为 [新数据库集群名称]。

源自数据库集群参数组的 Neptune 事件

下表显示的是数据库集群参数组为源类型时的事件类型和事件列表。

类别	RDS 事件 ID	描述
配置更改	RDS-EVENT-0037	参数组已修改。

源于安全组的 Neptune 事件

下表显示了安全组为源类型时的事件类别和事件列表。

类别	RDS 事件 ID	描述
配置更改	RDS-EVENT-0038	安全组已修改。
失败	RDS-EVENT-0039	不存在 [user] 拥有的安全组；已激活安全组的授权。

订阅 Neptune 事件通知

您可以使用 Neptune 控制台订阅事件通知，如下所示：

订阅 Neptune 事件通知

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择事件订阅。
3. 在事件订阅窗格中，选择创建事件订阅。
4. 在创建事件订阅对话框中，请执行以下操作：
    a. 对于名称，输入事件通知订阅的名称。
    b. 对于发送通知到，选择 Amazon SNS 主题的现有 Amazon SNS ARN，或者选择创建主题来输入主题的名称和收件人列表。
    c. 对于源类型，请选择一种源类型。
    d. 选择是以启用订阅。如果要创建订阅，但尚未发送通知，请选择否。
    e. 根据选定源类型的情况，选择您希望从中接收事件通知的事件类别和源。
    f. 选择创建。

管理 Neptune 事件通知订阅

如果您在 Neptune 控制台的导航窗格中选择事件订阅，则可以查看订阅类别和当前订阅列表。

您也可以修改或删除特定的订阅。

修改 Neptune 事件通知订阅

列出当前的 Neptune 事件通知订阅

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择事件订阅。事件订阅窗格中会显示您的所有事件通知订阅。
3. 在事件订阅窗格中，选择您要修改的订阅，然后选择编辑。
4. 在目标或来源部分中对订阅进行更改。您可以通过在源部分选择或取消选择源标识符来添加或删除它们。
5. 选择编辑。Neptune 控制台会表明正在修改订阅。

删除 Neptune 事件通知订阅

当您不再需要时，可以删除订阅。该主题的所有用户都将再也不会收到订阅指定的事件通知。

删除 Neptune 事件通知订阅

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择事件订阅。
3. 在 事件订阅窗格中，选择您希望删除的订阅。
4. 选择删除。
5. Neptune 控制台会表明正在删除订阅。

给 Amazon Neptune 资源加标签

可以使用 Neptune 标签向 Neptune 资源添加元数据。此外，您可以使用带有 AWS Identity and Access Management (IAM) 策略的标签来管理对 Neptune 资源的访问权限并控制可以对这些资源应用哪些操作。最后，您可以将具有类似标签的资源的费用分组在一起，使用标签来跟踪成本。

所有 Neptune 管理资源都可加标签，其中包括：

- 数据库实例
- 数据库集群
- 只读副本
- 数据库快照
- 数据库集群快照
- 事件订阅
- 数据库参数组
- 数据库集群参数组
- 数据库子网组

Neptune 资源标签概述

Amazon Neptune 标签是由您定义的名称/值对，与某种 Neptune 资源关联。此名称也叫键。为键提供值为可选操作。可使用标签向 Neptune 资源分配任意信息。例如，您可以使用标签键定义一个类别，而标签值可以是该类别中的一个项目。例如，您可能定义“project”标签键和“Salix”标签值，以指示将 Neptune 资源分配给 Salix 项目。您也可以使用标签通过 `environment=test` 或 `environment=production` 等键指定 Neptune 资源用于测试或生产。我们建议使用一组具有一致性的标签键，以使跟踪与 Neptune 资源关联的元数据变得更轻松。

使用标签来整理 AWS 账单，以反映您自己的成本结构。为此，请注册以获取包含标签键值的 AWS 账户 账单。然后，如需查看组合资源的成本，请按有同样标签键值的资源组织您的账单信息。例如，您可以将特定的应用程序名称用作几个资源的标签，然后组织账单信息，以查看在数个服务中的使用该应用程序的总成本。有关更多信息，请参阅 AWS Billing 用户指南 中的使用成本分配标签。

每个 Neptune 资源都有一个标签集，其中包含分配给该 Neptune 资源的所有标签。一个标签集可以包含多达 10 个标签，也可以为空。如果向 Neptune 资源添加一个标签，而该标签的键与资源上某个现有的标签相同，则新值将覆盖旧值。

AWS 不对您的标签应用任何语义含义；标签严格解释为字符串。Neptune 可以在数据库实例或其它 Neptune 资源上设置标签，这取决于您创建资源时使用的设置。例如，Neptune 可能添加一个标签来指示数据库实例用于生产或测试。

- 标签键是标签的名称，属于必填内容。该字符串值的长度可以在 1 到 128 个 Unicode 字符之间，并且不能带有前缀“`aws:`”或“`rds:`”。字符串只能包含 Unicode 字母、数字、空

格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。

- 标签值是标签的可选字符串值。该字符串值的长度可以在 1 到 256 个 Unicode 字符之间，并且不能带有前缀“aws:”。字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。

  在标签集中，值不必具有唯一性，且可为空值。例如，在 `project/Trinity` 和 `cost-center/Trinity` 的标签集中，可以存在键-值对。

> **Note**
>
> 您可以向快照添加标签。但是，您的账单不会反映此分组。

您可以使用 AWS Management Console AWS CLI、或 Neptune API 在海王星资源上添加、列出和删除标签。使用 AWS CLI 或 Neptune API 时，必须为要使用的海王星资源提供亚马逊资源名称 (ARN)。有关构造 ARN 的详细信息，请参阅为 Neptune 构建 ARN。

对标签进行缓存以用于授权。因此，可能先用几分钟添加和更新 Neptune 资源上的标签，然后标签才可用。

在 Neptune 中复制标签

在创建或还原数据库实例时，您可以指定将数据库实例中的标签复制到数据库实例的快照。复制标签将确保数据库快照的元数据与源数据库实例的元数据匹配，并且数据库快照的任何访问策略与源数据库实例的任何访问策略匹配。默认情况下不复制标签。

您可以为以下操作指定将标签复制到数据库快照：

- 创建数据库实例。
- 还原数据库实例。
- 创建只读副本。
- 复制数据库快照。

> **Note**
>
> 如果您在 create-db-cluster- AWS CLI snapshot 命令的--tag-key参数中包含一个值（或者为 API CreateDBClusterSnapshot 操作提供至少一个标签），那么 Neptune 不会将标签从源数据库实例复制到新的数据库快照。即使源数据库实例已启用 --copy-tags-to-snapshot(CopyTagsToSnapshot) 选项也是如此。
> 这意味着，您可以从数据库快照创建数据库实例的副本，而无需添加不适用于新数据库实例的标签。使用 AWS CLI create-db-cluster-snapshot命令（或 Nep CreateDBClusterSnapshot tune API 操作）创建数据库快照后，您可以按照本主题后面的说明添加标签。

在 Neptune 中使用标记 AWS Management Console

为 Amazon Neptune 资源加标签的过程对于所有资源均类似。以下过程展示如何为 Neptune 数据库实例添加标签。

要向数据库实例添加标签，请执行以下操作：

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择实例。

   > **Note**
   >
   > 要在实例窗格中筛选数据库实例列表，请在筛选实例框中键入文本字符串。只会显示包含该字符串的数据库实例。

3. 选择您要为其添加标签的数据库实例。
4. 选择实例操作，然后选择查看详细信息。
5. 在详细信息部分中，向下滚动到标签部分。
6. 选择 Add。将显示添加标签窗口。
7. 为标签键和值键入一个值。
8. 要添加其他标签，您可以选择添加其他标签，并为其标签键和值键入一个值。

   将该步骤重复执行所需的次数。

9. 选择 Add。

删除数据库实例的标签

1. 登录 AWS 管理控制台，打开亚马逊 Neptune 控制台，网址为 https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择实例。

> **Note**
>
> 要在实例窗格中筛选数据库实例列表，请在筛选实例框中键入文本字符串。只会显示包含该字符串的数据库实例。

3. 选择您要为其添加标签的数据库实例。
4. 选择实例操作，然后选择查看详细信息。
5. 在详细信息部分中，向下滚动到标签部分。
6. 选择要删除的标签。
7. 选择删除，然后在删除标签窗口中选择删除。

在 Neptune 中使用标记 AWS CLI

可以使用 AWS CLI为 Neptune 中的数据库实例添加、列出或删除标签。

- 要向 Neptune 资源添加一个或多个标签，请使用命令。 AWS CLI `add-tags-to-resource`
- 要列出 Neptune 资源上的标签，请使用命令。 AWS CLI `list-tags-for-resource`
- 要从 Neptune 资源中移除一个或多个标签，请使用命令。 AWS CLI `remove-tags-from-resource`

要了解有关如何构建所需 Amazon 资源名称 (ARN) 的更多信息，请参阅为 Neptune 构建 ARN。

使用 API 在 Neptune 中加标签

您可使用 Neptune API 为数据库实例添加、列出或删除标签。

- 要向 Neptune 资源添加标签，请使用 `AddTagsToResource` 操作。

- 要列出分配给 Neptune 资源的标签，请使用 ListTagsForResource。
- 要从 Neptune 资源删除标签，请使用 RemoveTagsFromResource 操作。

要了解有关如何构建所需 ARN 的更多信息，请参阅为 Neptune 构建 ARN。

在通过 Neptune API 使用 XML 时，标签会使用如下架构：

```
<Tagging>
    <TagSet>
        <Tag>
            <Key>Project</Key>
            <Value>Trinity</Value>
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

下表提供了允许使用的 XML 标签及其特征的列表。Key 和 Value 的值都区分大小写。例如，project=Trinity 和 PROJECT=Trinity 是两个不同的标签。

标签元素	描述
TagSet	标签集是分配给 Neptune 资源的所有标签的容器。每个资源只能有一个标签集。您只可以通过 Neptune API 使用 TagSet。
Tag	标签是用户定义的键值对。一个标签集中可以有 1 到 50 个标签。
Key	键是标签必需的名称。该字符串值的长度可以在 1 到 128 个 Unicode 字符之间，并且不能带有前缀“rds:”或“aws:”。字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$ ”）。 密钥在标签集中必须具有唯一性。例如，标签集中不能有键相同但值不同的键-值对，如 project/Trinity 和 project/Xanadu 。
值	值是标签的可选内容。该字符串值的长度可以在 1 到 256 个 Unicode 字符之间，并且不能带有前缀“rds:”或“aws:”。字符串只能包含 Unicode

标签元素	描述
	字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$ ”）。 在标签集中，值不必具有唯一性，且可为空值。例如，在 project/Trinity 和 cost-center/Trinity 的标签集中，可以存在键-值对。

在 Amazon Neptune 中使用管理 ARN

在 Amazon Web Services 中创建的资源分别使用 Amazon 资源名称 (ARN) 进行唯一标识。对于某些 Amazon Neptune 操作，您必须通过指定 Neptune 资源的 ARN 来唯一标识该资源。

> **Important**
>
> Amazon Neptune 共享 Amazon RDS ARN 的格式，用于使用 管理 API 参考 的管理操作。Neptune 管理 ARN 包含 rds，但不包含 neptune-db。有关标识 Neptune 数据资源的数据面板 ARN，请参阅指定数据资源。

主题

- 为 Neptune 构建 ARN
- 在 Amazon Neptune 中获取现有 ARN

为 Neptune 构建 ARN

您可以使用以下语法为 Amazon Neptune 资源构造 ARN。请注意，Neptune 共享 Amazon RDS ARN 的格式。

arn:aws:rds:*<region>*:*<account number>*:*<resourcetype>*:*<name>*

下表显示在构造特定 Neptune 管理资源类型的 ARN 时应使用的格式。

资源类型	ARN 格式
数据库实例	arn:aws:rds:*<region>*:*<account>* :db:*<name>*

资源类型	ARN 格式
	例如： `arn:aws:rds: us-east-2 :123456789012 :db:my-instance-1`
数据库群集	arn:aws:rds:*<region>*:*<account>* :cluster: *<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster: my-cluster-1`
事件订阅	arn:aws:rds:*<region>*:*<account>* :es:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :es:my-subscription`
数据库参数组	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :pg:my-param-enable-logs`
数据库集群参数组	arn:aws:rds:*<region>*:*<account>* :cluster-pg: *<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-pg: my-cluster-param-timezone`

资源类型	ARN 格式
数据库集群快照	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot: *<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-snap-20160809`
数据库子网组	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* 例如： `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

在 Amazon Neptune 中获取现有 ARN

你可以使用 AWS Management Console、 AWS Command Line Interface ()AWS CLI或 Neptune API 获取 Neptune 资源的 ARN。

使用获取现有 ARN AWS Management Console

要使用控制台获取 ARN，请导航到要获取其 ARN 的资源，然后查看该资源的详细信息。例如，要获取数据库实例的 ARN，请在导航面板中选择实例，然后从列表中选择所需的实例。此 ARN 位于实例详细信息部分。

使用获取现有 ARN AWS CLI

要使用获取特定 Neptune 资源的 ARN，请使用该资源的`describe`命令。 AWS CLI 下表显示了每条 AWS CLI 命令以及用于获取 ARN 的命令的 ARN 属性。

AWS CLI 命令	ARN 属性
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn

AWS CLI 命令	ARN 属性
describe-db-parameter-groups	DB ParameterGroup Arn
describe-db-cluster-parameter-groups	数据库 ClusterParameter GroupArn
describe-db-instances	数据库 InstanceArn
describe-events	SourceArn
describe-db-subnet-groups	DB SubnetGroup Arn
describe-db-clusters	数据库 ClusterArn
describe-db-cluster-snapshots	DB ClusterSnapshot Arn

例如，以下 AWS CLI 命令获取数据库实例的 ARN。

Example

对于 Linux、OS X 或 Unix：

```
aws neptune describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2
```

对于 Windows：

```
aws neptune describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2
```

使用 API 获取现有 ARN

要获取特定 Neptune 资源的 ARN，请调用以下 API 操作并使用如下所示的 ARN 属性。

Neptune API 操作	ARN 属性
DescribeEvent订阅	EventSubscriptionArn

Neptune API 操作	ARN 属性
DescribeCertificates	CertificateArn
describedB ParameterGroups	DB ParameterGroup Arn
describedB 群组 ClusterParameter	数据库 ClusterParameter GroupArn
DescribeDBInstances	数据库 InstanceArn
DescribeEvents	SourceArn
describedB SubnetGroups	DB SubnetGroup Arn
DescribeDBClusters	数据库 ClusterArn
describedB ClusterSnapshots	DB ClusterSnapshot Arn

备份和还原 Amazon Neptune 数据库集群

本节介绍如何备份和还原 Amazon Neptune 数据库集群。

主题

备份和还原 Neptune 数据库集群的概述

本节提供有关在 Amazon Neptune 中备份和还原数据的概括信息。

主题

Neptune 数据库集群的容错能力

Neptune 数据库集群设计为具有容错能力。群集卷跨一个 AWS 区域中的多个可用区，每个可用区均包含一个群集卷数据副本。该功能意味着您的数据库集群可容忍可用区的故障，而不发生任何数据丢失，只是会短暂中断服务。

如果数据库集群中的主实例失败，Neptune 将通过两种方式之一来自动失效转移到新的主实例：

- 将现有的 Neptune 副本提升为新的主实例
- 创建新的主实例

如果数据库集群具有一个或多个 Neptune 副本，则 Neptune 副本将在故障事件期间提升为主实例。故障事件将导致短暂中断，其间的读取和写入操作将失败并引发异常。不过，服务通常会在 120 秒内 (经常在 60 秒内) 还原。要提高数据库集群的可用性，建议您在两个或更多的不同可用区中创建至少一个或多个 Neptune 副本。

您可以通过为每个副本分配一个优先级来自定义发生故障后将 Neptune 副本提升为主实例的顺序。优先级介于 0（最高优先级）和 15（最低优先级）之间。如果主实例失败，则 Neptune 会将具有最高优先级的 Neptune 副本提升为新的主实例。您可以随时修改 Neptune 副本的优先级。修改优先级不会触发故障转移。

您可以使用 AWS CLI 来设置数据库实例的失效转移优先级，如下所示：

```
aws neptune modify-db-instance --db-instance-identifier (the instance ID) --promotion-
tier (the failover priority value)
```

多个 Neptune 副本可同属一个优先级，这会产生提升层问题。如果两个或更多 Neptune 副本同属一个优先级，则 Neptune 将提升最大的副本。如果两个或多个 Neptune 副本的优先级和大小均相同，则 Neptune 将提升同一提升层中的任意副本。

如果数据库集群不包含任何 Neptune 副本，则将在故障事件期间重新创建主实例。故障事件将导致中断，其间的读取和写入操作将失败并引发异常。创建新的主实例时将还原服务，该操作所需的时间通常在 10 分钟内。将 Neptune 副本提升为主实例要比创建新的主实例快得多。

Neptune 备份

Neptune 自动备份您的集群卷并将还原数据保留备份保留期的时长。Neptune 备份是连续且递增的，因此，您可以快速还原到备份保留期内的任何时间点。在写入备份数据时，不会发生任何性能影响或数据库服务中断。在创建或修改数据库集群时，可指定备份保留期 (1 天到 35 天)。

要控制备份存储使用量，可以缩短备份保留间隔和/或删除不再需要的旧的手动快照。要帮助控制成本，您可以监控持续备份和存在时间超出保留期的手动快照所消耗的存储量。您可以缩短备份保留间隔，并在不再需要手动快照时删除它们。

如果希望备份的保留期超出备份保留期，还可为集群卷中的数据创建快照。存储快照会产生 Neptune 的标准存储费用。有关 Neptune 存储定价的更多信息，请参阅 Amazon Neptune 定价。

Neptune 在整个备份保留期间保留增量还原数据。因此，您只需为在备份保留期之外所要保留的数据构建快照。您可以从该快照创建新的数据库集群。

> **Important**
>
> 如果您删除数据库集群，则会同时删除其所有自动备份，并且无法恢复。这意味着，除非您选择手动创建最终数据库快照，否则您以后无法将数据库实例还原到其最终状态。在删除集群时，不会删除手动快照。

> **Note**
>
> - 对于 Amazon Neptune 数据库集群，默认备份保留期为 1 天，不管创建数据库集群的方式如何。
> - 您无法在 Neptune 上禁用自动备份。Neptune 的备份保留期是由数据库集群管理的。

用于管理 Neptune 备份存储的 CloudWatch 指标

您可以使用 Amazon CloudWatch 指标 `TotalBackupStorageBilled`、`SnapshotStorageUsed` 和 `BackupRetentionPeriodStorageUsed` 来查看和监控 Neptune 备份使用的存储量，如下所示：

- `BackupRetentionPeriodStorageUsed` 表示目前用于存储连续备份的备份存储量（以字节为单位）。此值取决于集群卷的大小和您在保留期内所做的更改的数量。但是，出于计费目的，它不会超过保留期内的集群卷的累计大小。例如，如果您的集群 `VolumeBytesUsed` 大小为 107,374,182,400 字节 (100 GiB) 且保留期为两天，则 `BackupRetentionPeriodStorageUsed` 的最大值为 214,748,364,800 字节 (100 GiB + 100 GiB)。
- `SnapshotStorageUsed` 表示用于存储超出备份保留期的手动快照的备份存储量（以字节为单位）。当手动快照的创建时间戳在保留期内时，手动快照不会计入快照备份存储。所有自动快照也不会计入快照备份存储。每个快照的大小是您拍摄快照时的集群卷的大小。`SnapshotStorageUsed` 值取决于您保留的快照数和每个快照的大小。例如，假设您有一个保留期外的手动快照，并且在拍摄快照时集群的 `VolumeBytesUsed` 大小为 100 GiB。SnapshotStorageUsed 量为 107,374,182,400 字节 (100 GiB)。
- `TotalBackupStorageBilled` 表示 `BackupRetentionPeriodStorageUsed` 与 `SnapshotStorageUsed` 之和（以字节为单位）再减去免费备份存储量（等于一天的集群卷的大小）。免费备份存储等于最新的卷大小。例如，如果您的集群 `VolumeBytesUsed` 大小为 100 GiB，保留期为两天，并且您有一个保留期外的手动快照，则 `TotalBackupStorageBilled` 为 214,748,364,800 字节 (200 GiB + 100 GiB - 100 GiB)。

您可以通过 CloudWatch 控制台来使用 CloudWatch 指标监控 Neptune 集群并生成报告。有关如何使用 CloudWatch 指标的更多信息，请参阅监控 Neptune以及Neptun CloudWatch e 指标中的指标表。

从 Neptune 备份还原数据

您可以从 Neptune 保留的备份数据或您保存的数据库集群快照创建新的 Neptune 数据库集群以恢复数据。您可以将从备份数据创建的新数据库集群副本快速还原到备份保留期内的任何时间点。备份保留期内的 Neptune 备份的持续和增量性质意味着您无需频繁创建数据快照来缩短还原时间。

要确定数据库实例的最近或最早的可还原时间，请在 Neptune 控制台上查找 `Latest Restorable Time` 或 `Earliest Restorable Time` 值。数据库集群的最近可还原时间是您可还原数据库集群的最近时间点，通常为当前时间的 5 分钟内。最早可还原时间指定可将集群卷还原到的备份保留期内的时间点。

您可以通过检查 `Latest Restorable Time` 和 `Earliest Restorable Time` 值来确定数据库集群还原完成的时间。在还原操作完成之前，`Latest Restorable Time` 和 `Earliest Restorable Time` 值将返回 NULL。如果 `Latest Restorable Time` 或 `Earliest Restorable Time` 返回 NULL，则无法请求备份或还原操作。

使用 AWS Management Console 将数据库实例还原到指定时间

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择实例。选择要还原的数据库集群的主实例。
3. 选择实例操作，然后选择还原到时间点。

   在启动数据库实例窗口中，在还原时间下选择自定义。
4. 在 Custom (自定义) 下方指定要还原到的日期和时间。
5. 在设置下为数据库实例标识符键入已还原的新数据库实例的名称。
6. 选择 Launch DB Instance (启动数据库实例) 以启动还原后数据库实例。

   此时将使用您指定的名称创建新的数据库实例，并创建新的数据库集群。数据库集群名称是在新的数据库实例名称后面加上 `-cluster`。例如，如果新数据库实例名称为 `myrestoreddb`，则新数据库集群名称为 `myrestoreddb-cluster`。

Neptune 中的备份时段

自动备份在每天的首选备份时段中进行。如果备份所需的时间超过了分配到备份时段的时间，则备份将在该时段结束后继续，直至完成。备份时段不能与数据库实例的每周维护时段重叠。

在自动备份时段期间，启动备份进程时可能会短时间暂停存储 I/O (通常不到几秒)。在备份多可用区部署时，可能需要等待几分钟。

备份时段通常由 Neptune 底层的 Amazon RDS 控制面板从每个区域的八小时时间期间中随机选择。《Amazon RDS 用户指南》的备份时段部分记录了从中分配默认备份时段的每个区域的时间期间。

在 Neptune 中创建数据库集群快照

Neptune 创建数据库集群的存储卷快照，并备份整个数据库集群而不仅仅是各个数据库。当您创建数据库集群快照时，需要确定所要备份的数据库集群。然后，为数据库集群快照命名，以便以后通过它进行还原。创建数据库集群快照所用时间因数据库大小而异。快照包含整个存储卷。因此，文件（如临时文件）的大小也会影响创建快照所需时间。

您可以使用 AWS Management Console、AWS CLI 或 Neptune API 创建数据库集群快照。

使用控制台创建数据库集群快照

创建数据库集群快照

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择 Databases (数据库)。
3. 在数据库实例列表中，选择数据库集群的主实例。
4. 选择实例操作，然后选择拍摄快照。

   将显示拍摄数据库快照窗口。
5. 在快照名称框中，输入数据库集群快照的名称。
6. 选择拍摄快照。

从数据库集群快照还原

在创建数据库集群的 Amazon Neptune 快照时，Neptune 创建集群的存储卷快照，同时备份集群的所有数据，而不仅仅是各个实例。您随后可通过从该数据库集群快照还原来创建新的数据库集群。还原数据库集群时，您需要提供用于还原的数据库集群快照的名称，然后提供还原所创建的新数据库集群的名称。

目录

从快照还原 Neptune 数据库集群时需要注意的事项

您无法还原到现有的数据库集群

还原过程始终会创建新的数据库集群，因此您无法还原到已存在的数据库集群。

不还原任何实例

通过还原创建的新数据库集群没有与之关联的实例。

一旦还原完成且您的新数据库集群变为可用状态后，就显式创建将需要的实例。您可以在 Neptune 控制台上或使用 CreateDBInstance API 执行此操作。

不还原任何自定义参数组

通过还原创建的新数据库集群会自动具有与其关联的默认数据库参数组。

一旦还原完成并且新的数据库集群变为可用状态，就请关联您从中还原的实例所用的任何自定义数据库参数组。为此，请使用 Neptune 控制台上的修改命令或 ModifyDBInstance API。

> **Important**
>
> 我们建议您保存您正在创建其快照的数据库集群中使用的自定义参数组。然后，当您从该快照还原时，可以轻松地将正确的参数组与还原的数据库集群关联。

不还原任何自定义安全组

通过还原创建的新数据库集群会自动具有与其关联的默认安全组。

一旦还原完成并且新的数据库集群变为可用状态，就请关联您从中还原的实例所用的任何自定义安全组。为此，请使用 Neptune 控制台上的修改命令或 ModifyDBInstance API。

您无法从共享的加密快照中还原

您无法从共享并且加密的数据库集群快照中还原数据库集群。

而应创建快照的非共享副本，并从该副本还原。

还原后的数据库集群使用的存储空间与以前一样多

在从数据库集群快照还原数据库集群时，分配给新集群的存储量与分配给从中创建快照的数据库集群的存储量相同，不管实际使用了多少分配的存储量。

换句话说，计费的“高水位”不会发生变化。重置高水位需要从图表中导出数据，然后将这些数据重新加载到新的数据库集群中（请参阅Neptune 存储账单）。

如何从快照还原

您可以使用 AWS Management Console、AWS CLI 或 Neptune API 从数据库集群快照还原数据库集群。

使用控制台从快照中还原

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择快照。
3. 选择要从其还原的数据库集群快照。
4. 依次选择 Actions (操作) 和 Restore Snapshot (还原快照)。

5. 在 Restore DB Instance (还原数据库实例) 页面上的 DB Instance Identifier (数据库实例标识符) 字段中，输入还原的数据库集群的名称。
6. 选择 Restore DB Instance。
7. 如果要还原创建快照的数据库集群包含的数据库集群功能，您必须修改数据库集群以使用安全组。后续步骤假定您的数据库集群在 Virtual Private Cloud (VPC) 中。如果您的数据库集群不在 VPC 中，请使用 Amazon EC2 控制台找到该数据库集群所需的安全组。

   a. 通过以下网址打开 Amazon VPC 控制台：https://console.aws.amazon.com/vpc/。
   b. 在导航窗格中，选择 Security Groups（安全组）。
   c. 选择要用于数据库集群的安全组。如有必要，请添加规则，将该安全组链接到 EC2 实例的安全组。

复制数据库集群快照

使用 Neptune，您可以复制自动或手动数据库集群快照。在复制快照后，该副本为手动快照。

您可以在同一 AWS 区域中和跨 AWS 区域复制快照。

将自动快照复制到另一个 AWS 账户的过程包含两个步骤：首先从自动快照创建手动快照，然后将手动快照复制到另一个账户。

作为对复制的替代，您也可与其他 AWS 账户共享手动快照。有关更多信息，请参阅共享数据库集群快照。

主题

有关复制快照的限制

复制快照时，存在以下一些限制：

- 您可以在中国（北京）与中国（宁夏）之间复制快照，但不能在这些中国区域与其它 AWS 区域之间复制快照。
- 您可以在 AWS GovCloud（美国东部）与 AWS GovCloud（美国西部）之间复制快照，但不能在这些 AWS GovCloud (US) 区域与其它 AWS 区域之间复制快照。
- 如果您在目标快照可用之前删除了源快照，则快照复制会失败。在删除源快照之前，请确保目标快照的状态为 `AVAILABLE`。
- 每个账户最多可以同时进行指向单一区域的五个快照复制请求。
- 根据所涉及的区域和要复制的数据量，可能需要数小时才能完成跨区域快照复制。

  如果有大量跨区域快照复制请求来自给定的源 AWS 区域，则 Neptune 可能将来自该源 AWS 区域的新跨区域复制请求排入队列，直到完成某些正在进行的复制。当复制请求在该队列中时，不显示有关这些复制请求的进度信息。只有在复制开始后才会显示进度信息。

数据库集群快照副本的保留期

Neptune 在以下情况下会删除自动快照：

- 在保留期结束时。
- 当您对数据库集群禁用自动快照时。
- 当您删除数据库集群时。

如果要长期保留自动快照，则可复制它以创建一个手动数据库快照，之后该快照在您删除之前将会一直保留。如果手动快照超出了默认存储空间，则可能会产生 Neptune 存储成本。

有关备份存储成本的更多信息，请参阅 Neptune 定价。

复制快照时处理加密

您可以复制已使用 AWS KMS 加密密钥加密的快照。如果您复制加密的快照，则此快照的副本也必须加密。您可使用加密原始快照的 AWS KMS 加密密钥来加密副本，也可指定不同的 AWS KMS 加密密钥。

您无法在复制时加密未加密的数据库集群快照。

对于 Amazon Neptune 数据库集群快照，您也可以选择不加密数据库集群快照，而是在还原时指定 AWS KMS 加密密钥。还原的数据库集群使用指定的密钥加密。

跨 AWS 区域复制快照

> **Note**
>
> 此特征从 Neptune 引擎版本 1.0.2.1 开始推出。

在向与源快照的 AWS 区域不同的 AWS 区域复制快照时，即使复制增量快照，第一个副本也是完整快照副本。完整快照副本包含还原数据库实例需要的所有数据和元数据。在第一个快照副本后，可将同一数据库实例的增量快照复制到同一个 AWS 账户内的相同目标区域。

增量快照仅包含在同一数据库实例的最近快照后发生更改的数据。与完整快照复制相比，增量快照复制速度更快，产生的存储成本更低。已加密和未加密快照均支持跨 AWS 区域的增量快照复制。

> **Important**
>
> 对于共享快照，不支持复制增量快照。对于共享快照，所有副本都是完整的快照，即使在相同区域中也是如此。

根据所涉及的 AWS 区域和要复制的数据量，可能需要数小时才能完成跨区域快照复制。

使用控制台复制数据库集群快照

如果源数据库引擎为 Neptune，则您的快照是数据库集群快照。对于每个 AWS 账户，每个 AWS 区域一次最多可以复制五个数据库集群快照。支持复制加密和未加密的数据库集群快照。

有关数据传输定价的更多信息，请参阅 Neptune 定价。

要在正在进行复制时取消操作，请在数据库集群快照处于 copying (正在复制) 状态时删除目标数据库集群快照。

以下过程适用于复制加密和未加密的数据库集群快照：

复制数据库集群快照

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择快照。
3. 选中要复制的数据库集群快照的复选框。
4. 选择操作，然后选择复制快照。此时显示 Make Copy of DB Snapshot (建立数据库快照副本) 页面。
5. 在 New DB Snapshot Identifier (新数据库实例标识符) 中输入数据库集群快照副本的名称。
6. 要将标签和值从快照复制到快照的副本，请选择 Copy Tags (复制标签)。
7. 对于 Enable Encryption，请选择下列选项之一：
   - 如果数据库集群快照未加密，且不需要加密该副本，请选择启用加密。
   - 如果数据库集群快照未加密，但需要加密该副本，请选择禁用加密。在这种情况下，对于主密钥，请指定用于加密数据库集群快照副本的 AWS KMS 密钥标识符。
   - 如果数据库集群快照已加密，请选择启用加密。在这种情况下，必须加密该副本，因此，是为已选中状态。对于主密钥，指定用于加密数据库集群快照副本的 AWS KMS 密钥标识符。

8. 选择 Copy Snapshot (复制快照)。

使用 AWS CLI 复制数据库集群快照

您可以使用 copy-db-cluster-snapshot AWS CLI 命令复制数据库快照。

如果您将快照复制到新 AWS 区域，请在该新区域中运行命令。

使用以下参数描述和示例来确定使用 AWS CLI 复制快照时使用的参数。

- `--source-db-cluster-snapshot-identifier` – 源数据库快照的标识符。
  - 如果源快照与副本位于相同 AWS 区域中，则指定有效的数据库快照标识符，如 `neptune:instance1-snapshot-20130805`。
  - 如果源快照与副本位于不同 AWS 区域，则指定有效的数据库快照 ARN，如 `arn:aws:neptune:us-west-2:123456789012:snapshot:instance1-snapshot-20130805`。
  - 如果从共享的手动数据库快照进行复制，则该参数必须为共享的数据库快照的 Amazon Resource Name (ARN)。
  - 如果您复制加密快照，则此参数对于源 AWS 区域必须为 ARN 格式，并且必须匹配 `PreSignedUrl` 参数中的 `SourceDBSnapshotIdentifier`。
- `--target-db-cluster-snapshot-identifier` – 加密数据库快照的新副本的标识符。
- `--kms-key-id` – 加密数据库快照的 AWS KMS 密钥 ID。AWS KMS 密钥 ID 是 Amazon 资源名称 (ARN)、AWS KMS 密钥标识符或 AWS KMS 加密密钥的 AWS KMS 密钥别名。
  - 如果您从 AWS 账户复制加密的数据库快照，则可以为该参数指定值以使用新的 AWS KMS 加密密钥来加密副本。如果您不为该参数指定值，则使用与源数据库快照相同的 AWS KMS 密钥来加密数据库快照的副本。
  - 您无法使用此参数创建未加密快照的加密副本。尝试这样做会生成错误。
  - 如果将加密快照复制到不同 AWS 区域，必须为目标 AWS 区域指定 AWS KMS 密钥。AWS KMS 加密密钥特定于在其中创建它们的 AWS 区域，您无法将一个 AWS 区域中的加密密钥用于另一个 AWS 区域中。
- `--source-region` – 源数据库快照所在 AWS 区域的 ID。如果将加密快照复制到不同 AWS 区域，则必须指定该选项。
- `--region` – 正在将快照复制到的 AWS 区域的 ID。如果将加密快照复制到不同 AWS 区域，则必须指定该选项。

Example 从未加密快照，到相同区域

以下代码使用新名称 mydbsnapshotcopy，创建从 us-east-1 AWS 区域到 us-west-2 区域的快照副本。

针对 Linux、OS X 或 Unix：

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

对于 Windows：

```
aws neptune copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier instance1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy
```

Example 从未加密快照，跨区域

以下代码使用新名称 mydbsnapshotcopy，创建从 us-east-1 AWS 区域到 us-west-2 区域的快照副本。在 us-west-2 区域中运行命令。

针对 Linux、OS X 或 Unix：

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
east-1:123456789012:snapshot:instance1-snapshot-20130805 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \
  --source-region us-east-1 \
  --region us-west-2
```

对于 Windows：

```
aws neptune copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
east-1:123456789012:snapshot:instance1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^
  --source-region us-east-1 ^
  --region us-west-2
```

Example 从加密快照，跨区域

以下代码示例将加密的数据库快照从 us-east-1 AWS 区域复制到 us-west-2 区域。在 us-west-2 区域中运行命令。

针对 Linux、OS X 或 Unix：

```
aws neptune copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
west-2:123456789012:snapshot:instance1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy \
  --source-region us-east-1 \
  --region us-west-2
  --kms-key-id my_us_west_2_key
```

对于 Windows：

```
aws neptune copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:neptune:us-
west-2:123456789012:snapshot:instance1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier mydbsnapshotcopy ^
  --source-region us-east-1 ^
  --region us-west-2
  --kms-key-id my-us-west-2-key
```

共享数据库集群快照

使用 Neptune，您可以按以下方式共享手动数据库集群快照：

- 共享手动数据库集群快照（无论是否加密）可允许经授权的 AWS 账户复制快照。
- 共享手动数据库集群快照（无论是否加密）可允许经授权的 AWS 账户直接从快照还原数据库集群，无需复制数据库集群再从中进行还原。

> **Note**
>
> 要共享自动数据库集群快照，请通过复制自动快照来创建手动数据库集群快照，然后共享该副本。

有关从数据库集群快照还原数据库集群的更多信息，请参阅如何从快照还原。

您可以与最多 20 个其他 AWS 账户共享手动快照。您也可以将未加密的手动快照作为公有快照进行共享，这样所有 AWS 账户均可使用此快照。以公有快照形式共享快照时应谨慎，不要将您的私有信息包含在任何公有快照之中。

> **Note**
>
> 当使用 AWS Command Line Interface (AWS CLI) 或 Neptune API 从共享的快照还原数据库集群时，您必须指定共享快照的 Amazon 资源名称 (ARN) 作为快照标识符。

主题

共享加密的数据库集群快照

您可共享使用 AES-256 加密算法“静态”加密的数据库集群快照。有关更多信息，请参阅静态加密 Neptune 资源。为此，您必须执行以下步骤：

1. 与您希望允许其访问快照的任何账户共享用于加密快照的 AWS Key Management Service (AWS KMS) 加密密钥。

可将另一 AWS 账户添加到 KMS 密钥策略，来与该账户共享 AWS KMS 加密密钥。有关更新密钥策略的详细信息，请参阅《AWS KMS 开发人员指南》中的密钥策略。有关创建密钥策略的示例，请参阅本主题下文中的 创建 IAM 策略来启用加密快照的复制功能。

2. 使用 AWS Management Console、AWS CLI 或 Neptune API 与其它账户共享加密的快照。

这些限制适用于共享加密快照：

- 您无法公开共享加密的快照。
- 如果某个快照已使用共享该快照的 AWS 账户的默认 AWS KMS 加密密钥进行加密，则您无法共享该快照。

允许访问 AWS KMS 加密密钥

要让另一 AWS 账户复制通过您的账户共享的加密数据库集群快照，您与之共享快照的账户必须有权访问加密快照的 KMS 密钥。要允许另一 AWS 账户访问 AWS KMS 密钥，请使用您要与之共享的 AWS 账户的 ARN，将 KMS 密钥的密钥策略更新为 KMS 密钥策略中的 `Principal`。然后允许 `kms:CreateGrant` 操作。有关一般说明，请参阅《AWS Key Management Service 开发人员指南》中的允许其它账户中的用户使用 KMS 密钥。

在您已向某个 AWS 账户提供对 KMS 加密密钥的访问权限后，要复制您的加密快照，该 AWS 账户必须创建 IAM 用户（如果还没有此用户的话）。KMS 安全限制不允许为此使用根 AWS 账户身份。AWS 账户还必须将 IAM policy 附加到该 IAM 用户，以允许此 IAM 用户使用您的 KMS 密钥复制加密的数据库集群快照。

在下面的密钥策略示例中，用户 `111122223333` 是 KMS 加密密钥的所有者，而用户 `444455556666` 是要与之共享密钥的账户。通过包含用户 `444455556666` 的根 AWS 账户身份的 ARN 作为策略的 `Principal`，以及通过允许 `kms:CreateGrant` 操作，此更新的密钥策略为 AWS 账户提供了访问 KMS 密钥的权限。

```
{
  "Id=": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid=": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
```

```
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid=": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KeyUser",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

创建 IAM 策略来启用加密快照的复制功能

在外部 AWS 账户有权访问您的 KMS 密钥之后，账户的所有者可创建一个策略来允许为该账户创建的 IAM 用户复制使用 KMS 密钥加密的快照。

以下示例显示了一个可附加到 AWS 账户 444455556666 的 IAM 用户的策略。它使 IAM 用户能够从 AWS 账户 111122223333 复制已使用 us-west-2 区域中 KMS 密钥 c989c1dd-a3f2-4a5d-8d96-e793d082ab26 加密的共享快照。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid=": "AllowUseOfTheKey",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKey*",
                "kms:DescribeKey",
                "kms:CreateGrant",
                "kms:RetireGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"]
        },
        {
            "Sid=": "AllowAttachmentOfPersistentResources",
            "Effect": "Allow",
            "Action": [
                "kms:CreateGrant",
                "kms:ListGrants",
                "kms:RevokeGrant"
            ],
            "Resource": ["arn:aws:kms:us-west-2:111122223333:key/c989c1dd-
a3f2-4a5d-8d96-e793d082ab26"],
            "Condition": {
                "Bool": {
                    "kms:GrantIsForAWSResource": true
                }
            }
        }
    ]
}
```

有关更新密钥策略的详细信息，请参阅《AWS Key Management Service 开发人员指南》中的密钥策略。

共享数据库集群快照

您可以使用 AWS Management Console、AWS CLI 或 Neptune API 共享数据库集群快照。

使用控制台来共享数据库集群快照

使用 Neptune 控制台，可以与最多 20 个 AWS 账户共享手动数据库集群快照。您也可以停止与一个或多个账户共享手动快照。

共享手动数据库集群快照

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择快照。
3. 选择要共享的手动快照。
4. 依次选择 Actions (操作) 和 Share Snapshot (共享快照)。
5. 为 DB snapshot visibility (数据库快照可见性) 选择以下一个选项。
   - 如果源未加密，选择公有可允许所有 AWS 账户从您的手动数据库集群快照还原数据库集群。或者选择私有，以仅允许您指定的 AWS 账户从您的手动数据库集群快照还原数据库集群。

     > **Warning**
     >
     > 如果将 DB snapshot visibility（数据库快照可见性）设置为 Public（公有），则所有 AWS 账户均可从您的手动数据库集群快照还原数据库集群，并且可访问您的数据。请勿将包含私密信息的任何手动数据库集群快照以公开形式共享。

   - 如果源已加密，由于已加密的快照无法公开共享，DB snapshot visibility (数据库快照可见性) 将设为 Private (私密)。
6. 对于 AWS 账户 ID，输入您想要允许从您的手动快照还原数据库集群的账户的 AWS 账户标识符。然后，选择 Add (添加)。重复操作以加入其他 AWS 账户标识符，最多可包含 20 个 AWS 账户。

   如果您在许可账户列表中错加了某个 AWS 账户标识符，可以选择错误 AWS 账户标识符右侧的 Delete (删除) 将其从列表中删除。
7. 为您想要允许还原手动快照的所有 AWS 账户添加标识符以后，选择保存。

停止与某个 AWS 账户共享手动数据库集群快照

1. 通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择快照。
3. 选择要停止共享的手动快照。

4. 选择 Actions (操作)，然后 Share Snapshot (共享快照)。
5. 要取消某 AWS 账户的权限，请从授权账户列表中选择该账户的 AWS 账户标识符所对应的 Delete (删除)。
6. 选择 Save（保存）。

删除 Neptune 快照

您可以使用 AWS Management Console、AWS CLI 或 Neptune 管理 API 删除数据库快照：

使用控制台删除

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon Neptune 控制台：https://console.aws.amazon.com/neptune/home。
2. 在导航窗格中，选择快照。
3. 选择要删除的数据库快照。
4. 对于 Actions (操作)，选择 Delete Snapshot (删除快照)。
5. 在确认页面上选择 Delete (删除)。

使用 AWS CLI 删除

您还可以使用 AWS CLI delete_db_cluster_snapshot 命令删除数据库快照，使用 `--db-snapshot-identifier` 参数标识要删除的快照：

针对 Linux、OS X 或 Unix：

```
aws neptune delete-db-cluster-snapshot \
    --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

对于 Windows：

```
aws neptune delete-db-cluster-snapshot ^
    --db-snapshot-identifier <name-of-the-snapshot-to-delete>
```

使用 Neptune 管理 API 执行删除

您可以使用开发工具包之一，通过调用 DeleteDBClusterSnapshot API 来删除数据库快照，使用 `DBSnapshotIdentifier` 参数标识要删除的数据库快照。

最佳实践：充分利用 Neptune

下面是有关使用 Amazon Neptune 的一些一般建议。使用此信息作为参考可以快速找到使用 Amazon Neptune 和最大程度地提高性能的建议。

目录

- 在失效转移后创建新连接
- 将 maxInProcessPerConnection 和 maxSimultaneousUsagePerConnection 设置为相同值
- 将查询以字节码而不是字符串的格式发送到服务器
- 始终完全消耗查询返回的 ResultSet 或迭代器
- 按批次批量添加顶点和边缘
- 禁用 Java 虚拟机中的 DNS 缓存
- （可选）在每个查询级别设置超时
- 故障排除 java.util.concurrent.TimeoutException

- 使用 openCypher 和 Bolt 的 Neptune 最佳实践
  - 在查询中首选定向边缘而非双向边缘
  - Neptune 不支持在一个事务中进行多个并发查询
  - 在失效转移后创建新连接
  - 长寿命应用程序的连接处理
  - 的连接处理 AWS Lambda
  - 完成后关闭驱动程序对象
  - 使用显式事务模式进行读写
    - 只读事务
    - 只读事务
  - 异常的重试逻辑
  - 使用单个 SET 子句一次设置多个属性
  - 使用 SET 子句一次删除多个属性
  - 使用参数化查询
  - 在 UNWIND 子句中使用扁平化地图而不是嵌套地图
  - 在可变长度路径 (VLP) 表达式中将限制性更强的节点放在左侧
  - 使用精细的关系名称避免冗余的节点标签检查
  - 尽可能指定边缘标签
  - 尽可能避免使用 WITH 子句
  - 尽早在查询中放置限制性筛选器
  - 明确检查属性是否存在
  - 不要使用命名路径（除非是必需的）

Amazon Neptune 基本操作指导

以下是使用 Neptune 时您应遵循的基本操作指导。

- 了解 Neptune 数据库实例，以便您可以根据自己的性能和用例要求适当调整其大小。请参阅Amazon Neptune 数据库集群和实例。
- 监控您的 CPU 和内存使用情况。这有助于您了解何时迁移到具有更大 CPU 或内存容量的数据库实例类，以实现您所需的查询性能。可将 Amazon CloudWatch 设置为在使用模式发生更改或接近部署容量时向您发送通知。这样做可以帮助您维护系统性能和可用性。有关详细信息，请参阅 监控实例 和 监控 Neptune。

  由于 Neptune 具有自己的内存管理器，即使 CPU 使用率较高，相对较低的内存使用率也是正常的。执行查询时遇到内存不足异常的情况表明您最好增加可用内存。
- 启用自动备份并设置备份时段，使备份在便利的时段进行。
- 测试您的数据库实例的故障转移，以了解对于您的使用案例而言，该过程需要多长时间。它还有助于确保访问您数据库实例的应用程序在故障转移之后可以自动连接到新数据库实例。
- 如果可能，请在同一区域和 VPC 中运行您的客户端和 Neptune 集群，因为使用 VPC 对等连接的跨区域连接可能会导致查询响应时间的延迟。对于几个毫秒的查询响应，需要将客户端和 Neptune 集群保留在同一个区域和 VPC 中。
- 在创建只读副本实例时，该实例至少应与主写入器实例一样大。这有助于阻止复制滞后，并避免副本重新启动。请参阅避免在集群中使用不同的实例类。
- 在升级到新的主要引擎版本之前，请务必在升级之前在其上测试您的应用程序。为此，您可以克隆数据库集群，使克隆集群运行新的引擎版本，然后在克隆上测试您的应用程序。

- 为了便于进行故障转移，理想情况下，所有实例的大小应相同。

主题

Amazon Neptune 安全最佳实践

使用 AWS Identity and Access Management (IAM) 账户控制对 Neptune API 操作的访问权限。控制创建、修改或删除 Neptune 资源（如数据库实例、安全组、选项组或参数组）的操作，以及执行常见管理操作（如备份和还原数据库实例）的操作。

- 尽可能使用临时凭证而不是永久凭证。
- 为每个管理 Amazon Relational Database Service (Amazon RDS) 资源的用户分配一个 IAM 账户。切勿使用 AWS 账户根用户来管理 Neptune 资源。为每个人（包括您自己）创建一个 IAM 用户。
- 授予每位用户执行其职责所需的最小权限集。
- 使用 IAM 组有效地管理适用于多个用户的权限。
- 定期交替 IAM 凭证。

有关使用 IAM 访问 Neptune 资源的更多信息，请参阅Amazon Neptune 中的安全性。有关使用 IAM 的一般信息，请参阅《IAM 用户指南》中的 AWS Identity and Access Management 和 IAM 最佳实践。

避免在集群中使用不同的实例类

当您的数据库集群包含不同类的实例时，可能会随着时间推移而出现问题。最常见的问题是，由于复制滞后，小的读取器实例可能会进入重复重启循环。如果读取器节点的数据库实例类配置比写入器数据库实例的配置弱，则更改量可能太大，读取器无法跟上。

> **Important**
>
> 为避免复制滞后导致重复重启，请配置您的数据库集群，使所有实例具有相同的实例类（大小）。

您可以使用 Amazon CloudWatch 中的 `ClusterReplicaLag` 指标，查看数据库集群中的写入器实例（主实例）和读取器之间的滞后。`VolumeWriteIOPs` 指标还允许您检测集群中可能造成复制滞后的写入活动突发情况。

避免在批量加载期间重复重启

如果您在批量加载期间由于复制滞后而经历了重复重启只读副本的循环，则您的副本可能无法跟上数据库集群中写入器的速度。

要么将读取器扩展到比写入器大，要么在批量加载期间暂时将它们移除，然后在完成后重新创建。

如果您的谓词数量很多，则启用 OSGP 索引

如果您的数据模型包含大量不同的谓词（大多数情况下超过 1000），则可能会面临性能降低和更高的运营成本。

在这种情况下，您可以通过启用 OSGP 索引来提高性能。请参阅OSGP 索引。

尽可能避免长时间运行的事务

长时间运行的事务（无论是只读事务还是读写事务）可能导致以下类型的意外问题：

读取器实例或具有并发写入的写入器实例上长时间运行的事务可能会导致大量累积不同版本的数据。这可能会给筛选掉大部分结果的读取查询带来更高的延迟。

在某些情况下，数小时内累积的版本可能会导致新的写入受到节流。

如果实例重启，长时间运行且写入次数较多的读写事务也可能导致问题。如果实例因维护事件或崩溃而重启，则将回滚所有未提交的写入操作。此类撤消操作通常在后台运行，不会阻止实例恢复正常，但是任何与正在回滚的操作冲突的新写入操作都会失败。

例如，如果在上一次运行中断开连接后重试同一查询，则实例重启时查询可能会失败。

撤消操作所需的时间与所涉及更改的大小成正比。

使用 Neptune 指标的最佳实践

要确定因资源不足和其它常见瓶颈导致的性能问题，您可以监控可用于 Neptune 数据库集群的指标。

定期监控性能指标以收集有关各种时间范围内的平均值、最大值和最小值数据。这可帮助确定性能下降的时间。使用此数据，您可以针对特定指标阈值设置 Amazon CloudWatch 警报，以便在达到这些阈值时向您发出警报。

设置新的数据库集群并让它在典型工作负载下运行时，尝试按一些不同的间隔（例如，一小时、二十四小时、一周、两周）来捕获所有性能指标的平均值、最大值和最小值。这将使您能够了解运行状况。这有助于将操作的峰值时间与非峰值时间进行比较。您随后可以利用这些信息确定性能何时降到标准水平以下，然后相应设置警报。

有关如何查看 Neptune 指标的信息，请参阅 使用亚马逊监控 Neptune CloudWatch。

以下是首先要查看的最重要指标：

- BufferCacheHitRatio – 缓冲区缓存满足的请求的百分比。缓存未命中会给查询执行带来显著的延迟。如果缓存命中率低于 99.9%，并且您的应用程序存在延迟问题，请考虑升级实例类型以在内存中缓存更多数据。
- CPU 利用率 – 使用的计算机处理容量的百分比。根据您的查询性能目标，高 CPU 消耗值可能是正常情况。
- 可释放内存 – 数据库实例上可用的 RAM 量（以 MB 为单位）。Neptune 有自己的内存管理器，因此该指标可能低于您的预期。可以很好地指示您应考虑将实例类升级到具有更多 RAM 的实例类的迹象是，查询经常引发内存不足异常。

对于 CPU 和内存指标，监控选项卡中的红线指标标记为 75%。如果实例内存消耗频繁越过红线，请检查您的工作负载并考虑升级实例以改进查询性能。

优化 Neptune 查询的最佳实践

提高 Neptune 性能的最佳方式之一是优化最常使用和占用资源最多的查询，以降低其运行成本。

有关如何调整 Gremlin 查询的信息，请参阅Gremlin 查询提示和调整 Gremlin 查询。有关如何调整 SPARQL 查询的信息，请参阅 SPARQL 查询提示。

跨只读副本的负载均衡

读取器终端节点轮询路由的运行方式是更改 DNS 条目指向的主机。客户端必须创建一个新的连接并解析 DNS 记录，才能获取与新只读副本的连接，因为 WebSocket 连接通常会长时间保持活动状态。

要为连续请求获取不同的只读副本，请确保您的客户端在每次连接时都会解析 DNS 条目。这可能需要关闭连接并重新连接到读取器终端节点。

您还可以通过显式连接到实例终端节点来跨只读副本对请求进行负载均衡。

使用较大的临时实例加快加载速度

实例的大小越大，您的加载性能越高。如果您没有使用较大的实例类型，但希望提高加载速度，则可以先使用较大的实例进行加载然后删除它。

> **Note**
>
> 以下过程是针对新集群的。如果您目前已经有一个集群，则可以添加一个新的较大实例，然后将其提升为新的主数据库实例。

使用较大的实例大小加载数据

1. 使用单个 `r5.12xlarge` 实例创建集群。此实例是主数据库实例。
2. 创建大小相同 (`r5.12xlarge`) 的一个或多个只读副本。

   您可以创建大小较小的只读副本，但如果它们的大小不足以跟上主实例的写入速度，则可能必须频繁地重启。由此产生的停机时间会大大降低性能。
3. 在批量加载程序命令中，加入 “`parallelism` ： “`OVERSUBSCRIBE`” 以告知 Neptune 使用所有可用的 CPU 资源进行加载（请参阅Neptune 加载程序请求参数）。然后，加载操作将在 I/O 允许的范围内以最快的速度执行，这通常需要 60-70% 的 CPU 资源。
4. 使用 Neptune 加载程序加载您的数据。加载任务在主数据库实例上运行。
5. 数据完成加载后，请务必将集群中的所有实例缩减为相同的实例类型，以避免额外费用和重复重启问题（请参阅避免使用不同的实例大小）。

通过失效转移到只读副本来调整写入器实例的大小

调整数据库集群中实例（包括写入器实例）大小的最佳方法是创建或修改只读副本实例，使其具有所需的大小，然后故意失效转移到该只读副本。您的应用程序看到的停机时间只是更改写入器的 IP 地址所需的时间，大约为 3 到 5 秒。

您用来故意将当前写入器实例失效转移到只读副本实例的 Neptune 管理 API 是 FailoverDBCluster。如果您使用的是 Gremlin Java 客户端，则可能需要在失效转移后创建一个新的客户端对象来获取新的 IP 地址，如此处所述。

请务必将所有实例更改为相同的大小，这样可以避免重复重启的循环，如下所述。

数据预提取任务中断错误后重试上传

当您使用批量加载程序将数据加载到 Neptune 时，有时可能会出现 `LOAD_FAILED` 状态，在对某个请求的响应中报告 `PARSING_ERROR` 和 `Data prefetch task interrupted` 消息，以要求提供详细信息，如下所示：

```
"errorLogs" : [
  {
    "errorCode" : "PARSING_ERROR",
    "errorMessage" : "Data prefetch task interrupted: Data prefetch task for 11467
 failed",
    "fileName" : "s3://some-source-bucket/some-source-file",
    "recordNum" : 0
  }
]
```

如果遇到此错误，只需重试批量上传请求。

在发生临时中断时会出现此错误，这种中断一般不是由您的请求或数据导致的，并且通常可以通过再次运行批量上传请求加以解决。

如果您使用的是默认设置，即 `"mode":"AUTO"` 和 `"failOnError":"TRUE"`，则在发生中断时，加载程序会跳过已经成功加载的文件，并继续加载尚未加载的文件。

将 Gremlin 与 Neptune 结合使用的一般最佳实践

在将 Gremlin 图形遍历语言与 Neptune 一起使用时，请遵循以下建议。有关将 Gremlin 与 Neptune 一起使用的信息，请参阅the section called “Gremlin”。

Important

TinkerPop 版本 3.4.11 中进行了一项更改，可提高查询处理方式的正确性，但目前有时会严重影响查询性能。
例如，这种查询的运行速度可能会慢得多：

```
g.V().hasLabel('airport').
  order().
    by(out().count(),desc).
  limit(10).
  out()
```

由于 TinkerPop 3.4.11 的这一更改，限制步骤之后的顶点现在以非最佳方式提取。为避免这种情况，您可以通过在 `order().by()` 之后的任何点添加 barrier() 步骤来修改查询。例如：

```
g.V().hasLabel('airport').
  order().
    by(out().count(),desc).
  limit(10).
  barrier().
  out()
```

TinkerPop 3.4.11 已在 Neptune 引擎版本 1.0.5.0 中启用。

主题

在要部署 Gremlin 代码的上下文中对其进行测试

在 Gremlin 中，客户端可以通过多种方式向服务器提交查询：使用 WebSocket 或 Bytecode GLV，或者通过 Gremlin 控制台使用基于字符串的脚本提交。

务必认识到，根据您提交查询的方式，Gremlin 查询执行可能会有所不同。如果在字节码模式下提交，返回空结果的查询可能会视为成功，但如果在脚本模式下提交，则视为失败。例如，如果您在脚本模式查询中包含 `next()`，则 `next()` 会发送到服务器，但是使用字节码时，客户端通常会自行处理 `next()`。在第一种情况下，如果未找到任何结果，查询就会失败；但在第二种情况下，无论结果集是否为空，查询都会成功。

如果您在一个上下文中开发和测试代码（例如，通常以文本形式提交查询的 Gremlin 控制台），但随后在不同的上下文中部署代码（例如，通过使用字节码的 Java 驱动程序），则可能会遇到问题，即代码在生产环境中的行为与在开发环境中的行为不同。

> **Important**
>
> 请务必在将要部署 Gremlin 代码的 GLV 环境中测试该代码，以免出现意外结果。

构建更新插入查询以利用 DFE 引擎

通过尽可能充分地利用 Neptune DFE 引擎，可以显著提高更新插入查询的性能。

使用 Gremlin `mergeV()` 和 `mergeE()` 步骤进行高效的更新插入解释了如何构建更新插入查询以尽可能有效地使用 DFE 引擎。

创建高效的多线程 Gremlin 写入

使用 Gremlin 将数据通过多线程加载到 Neptune 时，有一些指导方针可供遵循。

如果有可能，向每个线程提供一组可以插入或修改而不造成冲突的顶点或边缘。例如，线程 1 地址 ID 范围是 1–50000，线程 2 地址 ID 范围是 50001–100000，以此类推。这会减少造成 `ConcurrentModificationException` 的可能性。为安全起见，围绕所有写入放置一个 `try/catch` 块。如果出现任何失败，您可在短暂延迟后重试它们。

以介于 50 到 100（顶点或边缘）之间的批大小进行的批量写入通常可以很好地工作。如果您要为每个顶点添加大量属性，其数量接近 50，那么 100 是比较好的选择。进行一些试验是值得的。因此，对于批量写入，您可以使用类似于下文的方法：

```
g.addV('test').property(id,'1').as('a').
  addV('test').property(id,'2').
  addE('friend').to('a').
```

然后，在每个批量操作中重复此方法。

相比在 Gremlin 与服务器的每次往返通信中添加一个顶点或边缘，使用批处理可大幅提升效率。

如果您使用 Gremlin 语言变体 (GLV) 客户端，则可以先创建遍历以便用编程方式创建批处理。然后在其中进行添加，最后对其进行迭代，例如：

```
t.addV('test').property(id,'1').as('a')
t.addV('test').property(id,'2')
t.addE('friend').to('a')
t.iterate()
```

如果可能，最好是使用 Gremlin 语言变体客户端。不过您可以使用其他客户端执行类似的操作，通过将字符串连接起来构建一个批次，以文本字符串的格式提交查询。

如果您使用的是 Gremlin 客户端库之一而不是基本 HTTP 进行查询，线程应该全部共享同一个客户端、集群或连接池。您可能需要调整设置来实现尽可能最佳的吞吐量，诸如 Gremlin 客户端使用的连接池大小和工作线程数等设置。

使用创建时间属性修剪记录

您可以通过将创建时间作为属性存储在顶点上并定期删除它们来修剪过时的记录。

如果需要将数据存储特定的生命周期，然后从图形中删除它（顶点生存时间），则可以在创建顶点时存储时间戳属性。您随后可以定期发出对在特定时间之前创建的所有顶点的 `drop()` 查询；例如：

```
g.V().has("timestamp", lt(datetime('2018-10-11')))
```

将 `datetime( )` 方法用于 Groovy 时间数据

Neptune 提供了 `datetime` 方法，用于为 Gremlin Groovy 变体中发送的查询指定日期和时间。这包括 Gremlin 控制台、使用 HTTP REST API 的文本字符串以及使用 Groovy 的任何其他序列化。

> **Important**
>
> 这仅 适用于您将 Gremlin 查询作为文本字符串 发送的方法。如果您使用的是 Gremlin 语言变体，则必须使用该语言的本机日期类和函数。有关更多信息，请参阅下一部分：the section called "本机日期和时间"。

从 TinkerPop 3.5.2（在 Neptune 引擎版本 1.1.1.0 中引入）开始，datetime 是 TinkerPop 不可或缺的一部分。

您可以使用 datetime 方法来存储和比较日期：

```
g.V('3').property('date',datetime('2001-02-08'))
```

```
g.V().has('date',gt(datetime('2000-01-01')))
```

将本机日期和时间用于 GLV 时间数据

如果您使用的是 Gremlin 语言变体 (GLV)，则必须对 Gremlin 时间数据使用由编程语言提供的原生日期和时间类以及函数。

官方 TinkerPop Java、Node.js (JavaScript)、Python 或 .NET 库是 Gremlin 语言变体库。

Important

这仅 适用于 Gremlin 语言变体 (GLV) 库。如果您使用了将 Gremlin 查询作为文本字符串发送的方法，则必须使用 Neptune 提供的 datetime() 方法。这包括 Gremlin 控制台、使用 HTTP REST API 的文本字符串以及使用 Groovy 的任何其他序列化。有关更多信息，请参阅上部分the section called “datetime( )”。

Python

下面是采用 Python 的示例的一部分，该示例为 ID 是“3”的顶点创建名为“date”的单个属性。它将值设置为使用 Python datetime.now() 方法生成的日期。

```
import datetime

g.V('3').property('date',datetime.datetime.now()).next()
```

有关使用 Python 连接到 Neptune 的完整示例，请参阅使用 Python 连接到 Neptune 数据库实例

Node.js (JavaScript)

下面是采用 JavaScript 的示例的一部分，该示例为 ID 是“3”的顶点创建名为“date”的单个属性。它将值设置为使用 Node.js `Date()` 构造函数生成的日期。

```
g.V('3').property('date', new Date()).next()
```

有关使用 Node.js 连接到 Neptune 的完整示例，请参阅使用 Node.js 连接到 Neptune 数据库实例

Java

下面是采用 Java 的示例的一部分，该示例为 ID 为“3”的顶点创建名为“date”的单个属性。它将值设置为使用 Java `Date()` 构造函数生成的日期。

```
import java.util.date

g.V('3').property('date', new Date()).next();
```

有关使用 Java 连接到 Neptune 的完整示例，请参阅使用 Java 客户端连接到 Neptune 数据库实例

.NET (C#)

下面是采用 C# 的示例的一部分，该示例为 ID 为“3”的顶点创建名为“date”的单个属性。它将值设置为使用 .NET *`DateTime.UtcNow`* 属性生成的日期。

```
Using System;

g.V('3').property('date', DateTime.UtcNow).next()
```

有关使用 C# 连接到 Neptune 的完整示例，请参阅使用 .NET 连接到 Neptune 数据库实例

将 Gremlin Java 客户端与 Neptune 结合使用的最佳实践

使用最新兼容版本的 Apache TinkerPop Java 客户端

如果可以，请始终使用你正在使用的引擎版本所支持的最新版本的 Apache TinkerPop Gremlin Java 客户端。更新的版本包含大量错误修复，可以提升客户端的稳定性、性能和可用性。

有关与各种 Neptune 引擎版本兼容的客户端版本的列表，请参阅Apache TinkerPop Java Gremlin 客户端。

跨多个线程重用客户端对象

在多个线程之间重用同一个客户端（或 GraphTraversalSource）对象 即，在应用程序中创建 org.apache.tinkerpop.gremlin.driver.Client 类的共享实例，而不是在每个线程中这样做。Client 对象是线程安全的，并且将其初始化的开销非常大。

在内部创建 Client 对象的 GraphTraversalSource 也是这种情况。例如，下面的代码会导致实例化新的 Client 对象：

```
import static
 org.apache.tinkerpop.gremlin.process.traversal.AnonymousTraversalSource.traversal;

  /////

GraphTraversalSource traversal = traversal()
                                    .withRemote(DriverRemoteConnection.using(cluster));
```

为读取和写入端点创建单独的 Gremlin Java 客户端对象

您可以通过仅在写入器终端节点上执行写入并从一个或多个只读终端节点进行读取来提高性能。

```
Client readerClient = Cluster.build("https://reader-endpoint")
          ...
          .connect()

Client writerClient = Cluster.build("https://writer-endpoint")
          ...
          .connect()
```

将多个只读副本端点添加到 Gremlin Java 连接池

在创建 Gremlin Java Cluster 对象时，您可以使用 .addContactPoint() 方法将多个只读副本实例添加到连接池的联系点。

```
Cluster.Builder readerBuilder = Cluster.build()
          .port(8182)
          .minConnectionPoolSize(…)
          .maxConnectionPoolSize(…)
          ……
          .addContactPoint("reader-endpoint-1")
```

```
            .addContactPoint("reader-endpoint-2")
```

关闭客户端以避免连接限制

使用完客户端后，请务必关闭客户端，以确保服务器关闭 WebSocket 连接并释放与连接关联的所有资源。如果您使用 `Cluster.close( )` 关闭集群，则客户端会自动关闭，因为随后在内部调用了 `client.close( )`。

如果客户端未正确关闭，Neptune 会在 20 到 25 分钟后终止所有空闲 WebSocket连接。但是，如果您在使用完连接后没有明确关闭 WebSocket连接，并且实时连接的数量达到WebSocket 并发连接限制，则会拒绝其他连接，并显示 HTTP 429 错误代码。此时，您必须重启 Neptune 实例来关闭连接。

调用 `cluster.close()` 的建议不适用于 Java AWS Lambda 函数。有关详细信息，请参阅 在 AWS Lambda 函数中管理 Gremlin WebSocket 连接。

在失效转移后创建新连接

在故障转移的情况下，Gremlin 驱动程序可能会继续连接到旧写入器，因为集群 DNS 名称解析为 IP 地址。如果出现这种情况，您可以在故障转移后创建新的 `Client` 对象。

将 `maxInProcessPerConnection` 和 `maxSimultaneousUsagePerConnection` 设置为相同值

`maxInProcessPerConnection`和`maxSimultaneousUsagePerConnection`参数都与您在单个WebSocket连接上可以同时提交的最大查询数有关。在内部，这些参数相互关联，如果修改其中之一而不修改其他参数，可能会导致客户端在尝试从客户端连接池提取连接时收到超时错误。

我们建议您保留默认的最小处理中值和同时使用值，并将 `maxInProcessPerConnection` 和 `maxSimultaneousUsagePerConnection` 设置为相同值。

为这些参数设置的值取决于查询复杂性和数据模型。在查询返回大量数据的使用案例中，每个查询会需要更多带宽，因此应使用较低的参数值和较高的 `maxConnectionPoolSize` 值。

相比之下，在查询返回数据量较少的情况下，`maxInProcessPerConnection` 和 `maxSimultaneousUsagePerConnection` 应设置为比 `maxConnectionPoolSize` 更高的值。

将查询以字节码而不是字符串的格式发送到服务器

在提交查询时，使用字节码而不是字符串具有以下好处：

- 及早捕获无效查询语法：使用字节码变量让您可以在编译阶段检测到无效的查询语法。如果您使用基于字符串的变量，则在将查询提交到服务器并返回错误之前，您无法发现无效语法。
- 避免基于字符串的性能损失：任何基于字符串的查询提交，无论是使用 WebSockets 还是 HTTP，都会导致顶点分离，这意味着 Vertex 对象由 ID、Label 和与顶点关联的所有属性组成（请参阅元素的属性）。

  这会导致当无需属性时在服务器上产生不必要的计算。例如，在客户希望使用查询 `g.V("hakuna#1")` 获得 ID 为“hakuna#1”的顶点时。如果查询是以基于字符串的提交形式发送，服务器将花费时间在 ID、标签以及此顶点的所有属性中检索。如果查询是以字节码提交的形式发送，服务器仅花时间检索顶点的 ID 和标签。

换而言之，不要提交类似于下文的查询：

```
 final Cluster cluster = Cluster.build("localhost")
                               .port(8182)
                               .maxInProcessPerConnection(32)
                               .maxSimultaneousUsagePerConnection(32)
                               .serializer(Serializers.GRAPHBINARY_V1D0)
                               .create();

 try {
     final Client client = cluster.connect();
     List<Result> results =
client.submit("g.V().has('name','pumba').out('friendOf').id()").all().get();
     System.out.println(verticesWithNamePumba);
 } finally {
     cluster.close();
 }
```

而是使用字节码提交查询，如下所示：

```
 final Cluster cluster = Cluster.build("localhost")
                               .port(8182)
                               .maxInProcessPerConnection(32)
                               .maxSimultaneousUsagePerConnection(32)
                               .serializer(Serializers.GRAPHBINARY_V1D0)
                               .create();

 try {
     final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
```

```
      List<Object> verticesWithNamePumba = g.V().has("name",
 "pumba").out("friendOf").id().toList();
      System.out.println(verticesWithNamePumba);
  } finally {
      cluster.close();
  }
```

始终完全消耗查询返回的 ResultSet 或迭代器

客户端对象应始终完全使用 `ResultSet`（在基于字符串的提交时）或 `GraphTraversal` 返回的迭代器。如果未完全使用查询结果，服务器会挂起，等待客户端使用完它们。

如果您的应用程序只需一部分结果，您可以在查询中使用 `limit(X)` 步骤来限制服务器生成的结果数。

按批次批量添加顶点和边缘

对 Neptune 数据库的每个查询都在单个事务的范围内运行，除非您使用会话。这意味着，如果您需要使用 gremlin 查询插入大量数据，请将它们以 50-100 个的批大小打包到一起，通过减少为每个负载创建的事务数来改善性能。

例如，如下所示添加 5 个顶点到数据库：

```
// Create a GraphTraversalSource for the remote connection
final GraphTraversalSource g =
 traversal().withRemote(DriverRemoteConnection.using(cluster));
// Add 5 vertices in a single query
g.addV("Person").property(T.id, "P1")
 .addV("Person").property(T.id, "P2")
 .addV("Person").property(T.id, "P3")
 .addV("Person").property(T.id, "P4")
 .addV("Person").property(T.id, "P5").iterate();
```

禁用 Java 虚拟机中的 DNS 缓存

在一个环境中，如果您希望跨多个线程只读副本实现请求的负载均衡，则需要禁用 Java 虚拟机 (JVM) 中的 DNS 缓存并在创建集群时提供 Neptune 的读取终端节点。禁用 JVM DNS 缓存可以确保为每个新连接重新解析 DNS，以在所有只读副本之间分配连接请求。可以在应用程序的初始化代码中使用以下行执行此操作：

```
java.security.Security.setProperty("networkaddress.cache.ttl", "0");
```

但是，上面的 A mazon Gremlin Java 客户端代码提供了更完整、更强大的负载平衡解决方案。GitHubAmazon Java Gremlin 客户端知道您的集群拓扑，并在您的 Neptune 集群中的一组实例之间公平地分配连接和请求。有关使用该客户端的 Java Lambda 函数示例，请参阅此博客文章。

（可选）在每个查询级别设置超时

Neptune 向您提供了使用参数组选项 neptune_query_timeout 为查询设置超时的功能（请参阅参数）。但是，从 Java 客户端版本 3.3.7 开始，您也可以使用如下所示代码覆盖全局超时：

```
 final Cluster cluster = Cluster.build("localhost")
                                .port(8182)
                                .maxInProcessPerConnection(32)
                                .maxSimultaneousUsagePerConnection(32)
                                .serializer(Serializers.GRAPHBINARY_V1D0)
                                .create();

 try {
     final GraphTraversalSource g =
traversal().withRemote(DriverRemoteConnection.using(cluster));
     List<Object> verticesWithNamePumba = g.with(ARGS_EVAL_TIMEOUT,
500L).V().has("name", "pumba").out("friendOf").id().toList();
     System.out.println(verticesWithNamePumba);
 } finally {
     cluster.close();
 }
```

或者，对于基于字符串的查询提交，代码如下所示：

```
 RequestOptions options = RequestOptions.build().timeout(500).create();
 List<Result> result = client.submit("g.V()", options).all().get();
```

> **Note**
>
> 如果您将查询超时值设置得过高，可能会产生意想不到的成本，尤其是在无服务器实例上。如果没有合理的超时设置，查询保持运行的时间可能会比预期长得多，从而产生您意想不到的成本。在运行查询时可以纵向扩展到昂贵的大型实例类型的无服务器实例上尤其如此。
>
> 您可以通过使用查询超时值来避免此类意外开支，该值可以适应您预期的运行时间，并且只会导致异常长的运行超时。

故障排除 java.util.concurrent.TimeoutException

当 Gremlin 请求在等待其中一个java.util.concurrent.TimeoutException连接中的插槽可用时，客户端本身超时，Gremlin Java 客户端会抛出。 WebSocket 此超时持续时间由 maxWaitForConnection 客户端可配置参数控制。

> **Note**
>
> 由于在客户端上超时的请求从不会发送到服务器，因此它们不会反映在服务器上捕获的任何指标中，例如 GremlinRequestsPerSec。

这种超时通常通过以下两种方式之一引起：

- 服务器实际上已达到最大容量。如果是这样的话，服务器上的队列就会被填满，你可以通过监视“MainRequestQueuePending请求” CloudWatch 指标来检测到这一点。服务器可以处理的并行查询数量取决于其实例大小。

  如果 MainRequestQueuePendingRequests 指标未显示服务器上积累了待处理的请求，则服务器可以处理更多请求，而超时是由客户端节流造成的。
- 客户端节流请求。通常可以通过更改客户端配置设置来解决这个问题。

  可通过以下方式粗略估计客户端可以发送的最大并行请求数：

  ```
  maxParallelQueries = maxConnectionPoolSize * Max( maxSimultaneousUsagePerConnection,
   maxInProcessPerConnection )
  ```

  向客户端发送的查询超过 maxParallelQueries 会导致 java.util.concurrent.TimeoutException 异常。您可以通过多种方式解决这个问题：
  - 增加连接超时时间。如果延迟对您的应用程序不重要，请增加客户端的 maxWaitForConnection 设置。然后，客户端会等待更长的时间才会超时，这反过来又会增加延迟。
  - 增加每个连接的最大请求数。这允许使用同一个 WebSocket 连接发送更多请求。通过增加客户端的 maxSimultaneousUsagePerConnection 和 maxInProcessPerConnection 设置来实现此目的。这些设置通常应具有相同的值。
  - 增加连接池中的连接数。通过增加客户端的 maxConnectionPoolSize 设置来实现此目的。代价是资源消耗增加，因为每个连接都使用内存和操作系统文件描述符，并且在初始化期间需要 SSL 和 WebSocket握手。

使用 openCypher 和 Bolt 的 Neptune 最佳实践

将 openCypher 查询语言和 Bolt 协议与 Neptune 结合使用时，请遵循以下最佳实践。有关在 Neptune 中使用 openCypher 的信息，请参阅使用 openCypher 访问 Neptune 图形。

在查询中首选定向边缘而非双向边缘

当 Neptune 执行查询优化时，双向边缘会使创建最佳查询计划变得困难。次优计划要求引擎执行不必要的工作，从而导致性能降低。

因此，请尽可能使用定向边缘而不是双向边缘。例如，使用：

```
MATCH p=(:airport {code: 'ANC'})-[:route]->(d) RETURN p)
```

而不是：

```
MATCH p=(:airport {code: 'ANC'})-[:route]-(d) RETURN p)
```

大多数数据模型实际上不需要在两个方向上遍历边缘，因此，通过切换到使用定向边缘，查询可以显著提高性能。

如果您的数据模型确实需要遍历双向边缘，请将 MATCH 模式中的第一个节点（左侧）设置为筛选限制最严的节点。

例如，“为我找到往返 ANC 机场的所有 routes”。编写这个查询以从 ANC 机场开始，如下所示：

```
MATCH p=(src:airport {code: 'ANC'})-[:route]-(d) RETURN p
```

引擎可以执行最少的工作量来满足查询，因为受限制最严的节点放置为模式中的第一个节点（左侧）。然后，引擎可以优化查询。

这比在模式末尾筛选 ANC 机场要好得多，如下所示：

```
MATCH p=(d)-[:route]-(src:airport {code: 'ANC'}) RETURN p
```

当受限制最严的节点没有放在模式中的首位时，引擎必须执行额外的工作，因为它无法优化查询，必须执行额外的查找才能得出结果。

Neptune 不支持在一个事务中进行多个并发查询

尽管 Bolt 驱动程序本身允许在事务中进行并发查询，但 Neptune 不支持在一个事务中并发运行多个查询。相反，Neptune 要求一个事务中的多个查询按顺序运行，并且在启动下一个查询之前完全消耗掉每个查询的结果。

以下示例显示了如何使用 Bolt 在一个事务中按顺序运行多个查询，以便在下一个查询开始之前完全消耗掉每个查询的结果：

```
final String query = "MATCH (n) RETURN n";

try (Driver driver = getDriver(HOST_BOLT, getDefaultConfig())) {
  try (Session session = driver.session(readSessionConfig)) {
    try (Transaction trx = session.beginTransaction()) {
      final Result res_1 = trx.run(query);
      Assert.assertEquals(10000, res_1.list().size());
      final Result res_2 = trx.run(query);
      Assert.assertEquals(10000, res_2.list().size());
    }
  }
}
```

在失效转移后创建新连接

在失效转移的情况下，Bolt 驱动程序可以继续连接到旧的写入器实例，而不是新的活动写入器实例，因为 DNS 名称已解析为特定的 IP 地址。

为防止出现这种情况，请在进行任何失效转移后关闭 `Driver` 对象，然后重新连接该对象。

长寿命应用程序的连接处理

在构建长寿命的应用程序（例如，在容器内或 Amazon EC2 实例上运行的应用程序）时，只需实例化 `Driver` 对象一次，然后在应用程序的生命周期内重用该对象。`Driver` 对象是线程安全的，并且将其初始化的开销非常大。

的连接处理 AWS Lambda

不建议在 AWS Lambda 功能中使用螺栓驱动器，因为它们具有连接开销和管理要求。请改用 HTTPS 端点。

完成后关闭驱动程序对象

在完成对客户端的操作后，务必关闭客户端，以便服务器关闭 Bolt 连接并释放与连接关联的所有资源。如果您使用 `driver.close()` 关闭驱动程序，则会自动发生这种情况。

如果驱动程序未正确关闭，Neptune 会在 20 分钟后终止所有空闲的 Bolt 连接，或者，如果您使用的是 IAM 身份验证，则会在 10 天后终止所有空闲的 Bolt 连接。

Neptune 支持的并发 Bolt 连接不超过 1000 个。如果您在使用完连接后没有显式关闭连接，并且实时连接的数量达到了 1000 的限制，则任何新的连接尝试都会失败。

使用显式事务模式进行读写

在将事务与 Neptune 和 Bolt 驱动程序结合使用时，最好将读取和写入事务的访问模式显式设置为正确的设置。

只读事务

对于只读事务，如果您在构建会话时没有传入适当的访问模式配置，则使用默认的隔离级别，即突变查询隔离。因此，对于只读事务来说，将访问模式显式设置为 `read` 非常重要。

自动提交读取事务示例：

```
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.READ)
  .build();
Session session = driver.session(sessionConfig);
try {
  (Add your application code here)
} catch (final Exception e) {
  throw e;
} finally {
  driver.close()
}
```

读取事务示例：

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
```

```
  .withDefaultAccessMode(AccessMode.READ)
  .build();
driver.session(sessionConfig).readTransaction(
  new TransactionWork<List<String>>() {
    @Override
    public List<String> execute(org.neo4j.driver.Transaction tx) {
      (Add your application code here)
    }
  }
);
```

在这两种情况下，都使用 Neptune 只读事务语义实现 SNAPSHOT 隔离。

由于只读副本仅接受只读查询，因此提交到只读副本的任何查询都在 SNAPSHOT 隔离语义下运行。

只读事务没有脏读或不可重复读取。

只读事务

对于突变查询，有三种不同的机制可以创建写入事务，每种机制如下所示：

隐式写入事务示例：

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withDefaultAccessMode(AccessMode.WRITE)
  .build();
driver.session(sessionConfig).writeTransaction(
  new TransactionWork<List<String>>() {
    @Override
    public List<String> execute(org.neo4j.driver.Transaction tx) {
      (Add your application code here)
    }
  }
);
```

自动提交写入事务示例：

```
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.Write)
```

```
  .build();
Session session = driver.session(sessionConfig);
try {
  (Add your application code here)
} catch (final Exception e) {
    throw e;
} finally {
    driver.close()
}
```

显式写入事务示例：

```
Driver driver = GraphDatabase.driver(url, auth, config);
SessionConfig sessionConfig = SessionConfig
  .builder()
  .withFetchSize(1000)
  .withDefaultAccessMode(AccessMode.WRITE)
  .build();
Transaction beginWriteTransaction = driver.session(sessionConfig).beginTransaction();
  (Add your application code here)
beginWriteTransaction.commit();
driver.close();
```

写入事务的隔离级别

- 作为突变查询的一部分进行的读取是在 READ COMMITTED 事务隔离下运行的。
- 对于作为突变查询一部分进行的读取，没有脏读。
- 在突变查询中读取时，记录和记录范围会被锁定。
- 当突变事务已读取索引范围时，可以强力保证在读取结束之前，任何并发事务都不会修改该范围。

突变查询不是线程安全的。

有关冲突，请参阅使用锁定等待超时解决冲突。

突变查询失败时不会自动重试。

异常的重试逻辑

对于所有允许重试的异常，通常最好使用指数回退和重试策略，在两次重试之间提供逐渐延长的等待时间，以便更好地处理诸如 ConcurrentModificationException 错误等临时问题。下面显示指数回退和重试模式的示例：

```
public static void main() {
  try (Driver driver = getDriver(HOST_BOLT, getDefaultConfig())) {
    retriableOperation(driver, "CREATE (n {prop:'1'})")
        .withRetries(5)
        .withExponentialBackoff(true)
        .maxWaitTimeInMilliSec(500)
        .call();
  }
}

protected RetryableWrapper retriableOperation(final Driver driver, final String query){
  return new RetryableWrapper<Void>() {
    @Override
    public Void submit() {
      log.info("Performing graph Operation in a retry manner......");
      try (Session session = driver.session(writeSessionConfig)) {
        try (Transaction trx =  session.beginTransaction()) {
            trx.run(query).consume();
            trx.commit();
        }
      }
      return null;
    }

    @Override
    public boolean isRetryable(Exception e) {
      if (isCME(e)) {
        log.debug("Retrying on exception.... {}", e);
        return true;
      }
      return false;
    }

    private boolean isCME(Exception ex) {
      return ex.getMessage().contains("Operation failed due to conflicting concurrent
 operations");
    }
  };
}



/**
```

```
 * Wrapper which can retry on certain condition. Client can retry operation using this
 class.
 */
@Log4j2
@Getter
public abstract class RetryableWrapper<T> {

  private long retries = 5;
  private long maxWaitTimeInSec = 1;
  private boolean exponentialBackoff = true;

  /**
   * Override the method with custom implementation, which will be called in retryable
 block.
   */
  public abstract T submit() throws Exception;

  /**
   * Override with custom logic, on which exception to retry with.
   */
  public abstract boolean isRetryable(final Exception e);

  /**
   * Define the number of retries.
   *
   * @param retries -no of retries.
   */
  public RetryableWrapper<T> withRetries(final long retries) {
    this.retries = retries;
    return this;
  }

  /**
   * Max wait time before making the next call.
   *
   * @param time - max polling interval.
   */
  public RetryableWrapper<T> maxWaitTimeInMilliSec(final long time) {
    this.maxWaitTimeInSec = time;
    return this;
  }

  /**
   * ExponentialBackoff coefficient.
```

```
   */
  public RetryableWrapper<T> withExponentialBackoff(final boolean expo) {
    this.exponentialBackoff = expo;
    return this;
  }

  /**
   * Call client method which is wrapped in submit method.
   */
  public T call() throws Exception {
    int count = 0;
    Exception exceptionForMitigationPurpose = null;
    do {
      final long waitTime = exponentialBackoff ? Math.min(getWaitTimeExp(retries),
maxWaitTimeInSec) : 0;
      try {
          return submit();
      } catch (Exception e) {
        exceptionForMitigationPurpose = e;
        if (isRetryable(e) && count < retries) {
          Thread.sleep(waitTime);
          log.debug("Retrying on exception attempt - {} on exception cause - {}",
count, e.getMessage());
        } else if (!isRetryable(e)) {
          log.error(e.getMessage());
          throw new RuntimeException(e);
        }
      }
    } while (++count < retries);

    throw new IOException(String.format(
          "Retry was unsuccessful.... attempts %d. Hence throwing exception " + "back
to the caller...", count),
          exceptionForMitigationPurpose);
  }

  /*
   * Returns the next wait interval, in milliseconds, using an exponential backoff
   * algorithm.
   */
  private long getWaitTimeExp(final long retryCount) {
    if (0 == retryCount) {
      return 0;
    }
```

```
    return ((long) Math.pow(2, retryCount) * 100L);
  }
}
```

使用单个 SET 子句一次设置多个属性

与其使用多个 SET 子句来设置单个属性，不如使用映射一次为一个实体设置多个属性。

您可以使用：

```
MATCH (n:SomeLabel {`~id`: 'id1'})
SET n += {property1 : 'value1',
property2 : 'value2',
property3 = 'value3'}
```

而不是：

```
MATCH (n:SomeLabel {`~id`: 'id1'})
SET n.property1 = 'value1'
SET n.property2 = 'value2'
SET n.property3 = 'value3'
```

SET 子句接受单个属性或映射。如果在单个实体上更新多个属性，则使用带有映射的单个 SET 子句允许在单个操作中执行更新，而不是在多个操作中执行更新，这样可以更有效地执行多个操作。

使用 SET 子句一次删除多个属性

使用 OpenCypher 语言时，REMOVE 用于从实体中移除属性。在 Neptune 中，要删除的每个属性都需要单独的操作，这增加了查询延迟。相反，您可以将 SET 与地图一起使用，将所有属性值设置为`null`，在 Neptune 中，这等同于删除属性。当需要移除单个实体上的多个属性时，Neptune 的性能将得到提高。

使用:

```
WITH {prop1: null, prop2: null, prop3: null} as propertiesToRemove
MATCH (n)
SET n += propertiesToRemove
```

而不是：

```
MATCH (n)
```

```
REMOVE n.prop1, n.prop2, n.prop3
```

使用参数化查询

建议在使用 OpenCypher 进行查询时始终使用参数化查询。查询引擎可以利用重复的参数化查询来实现查询计划缓存等功能，在这些功能中，重复调用具有不同参数的相同参数化结构可以利用缓存的计划。为参数化查询生成的查询计划只有在 100 毫秒内完成且参数类型为 NUMBER、BOOLEAN 或 STRING 时才会被缓存和重复使用。

使用:

```
MATCH (n:foo) WHERE id(n) = $id RETURN n
```

带参数：

```
parameters={"id": "first"}
parameters={"id": "second"}
parameters={"id": "third"}
```

而不是：

```
MATCH (n:foo) WHERE id(n) = "first" RETURN n
MATCH (n:foo) WHERE id(n) = "second" RETURN n
MATCH (n:foo) WHERE id(n) = "third" RETURN n
```

在 UNWIND 子句中使用扁平化地图而不是嵌套地图

深层嵌套结构可能会限制查询引擎生成最佳查询计划的能力。为了部分缓解此问题，以下定义的模式将为以下情况创建最佳计划：

- 场景 1：使用包含数字、字符串和布尔值的密码字面值列表进行解压。
- 场景 2：使用扁平化地图列表进行展开，其中仅包含密码文字（数字、字符串、布尔值）作为值。

在编写包含 UNWIND 子句的查询时，请使用上述建议来提高性能。

场景 1 示例：

```
UNWIND $ids as x
MATCH(t:ticket {`~id`: x})
```

带参数：

```
parameters={
  "ids": [1, 2, 3]
}
```

场景 2 的一个示例是生成要创建或合并的节点列表。与其发出多个语句，不如使用以下模式将属性定义为一组扁平化地图：

```
UNWIND $props as p
CREATE(t:ticket {title: p.title, severity:p.severity})
```

带参数：

```
parameters={
  "props": [
    {"title": "food poisoning", "severity": "2"},
    {"title": "Simone is in office", "severity": "3"}
  ]
}
```

而不是嵌套的节点对象，比如：

```
UNWIND $nodes as n
CREATE(t:ticket n.properties)
```

带参数：

```
parameters={
  "nodes": [
    {"id": "ticket1", "properties": {"title": "food poisoning", "severity": "2"}},
    {"id": "ticket2", "properties": {"title": "Simone is in office", "severity": "3"}}
  ]
}
```

在可变长度路径 (VLP) 表达式中将限制性更强的节点放在左侧

在可变长度路径 (VLP) 查询中，查询引擎通过选择在表达式的左侧或右侧开始遍历来优化评估。该决定基于左侧和右侧模式的基数。基数是与指定模式匹配的节点数。

- 如果右侧模式的基数为 1，则右侧将是起点。

- 如果左侧和右侧的基数均为 1，则检查两侧的扩展并从较小的扩展侧开始。扩展是 VLP 表达式中左侧节点和右侧节点的传出或传入边的数量。仅当 VLP 关系为单向关系且提供了关系类型时，才使用优化的这一部分。
- 否则，左侧将是起点。

对于 VLP 表达式链，此优化只能应用于第一个表达式。其他 VLP 从左侧开始评估。例如，假设 (a)、(b) 的基数为一，(c) 的基数大于一。

- `(a)-[*1..]->(c)`:评价以 (a) 开头。
- `(c)-[*1..]->(a)`:评价以 (a) 开头。
- `(a)-[*1..]-(c)`:评价以 (a) 开头。
- `(c)-[*1..]-(a)`:评价以 (a) 开头。

现在让 (a) 的传入边为二，(a) 的出边为三，(b) 的传入边为四，(b) 的出边为五。

- `(a)-[*1..]->(b)`: 评估从 (a) 开始，因为 (a) 的传出边缘小于 (b) 的传入边缘。
- `(a)<-[*1..]-(b)`: 评估从 (a) 开始，因为 (a) 的传入边缘小于 (b) 的传出边缘。

通常，将限制性更强的模式放在 VLP 表达式的左侧。

使用精细的关系名称避免冗余的节点标签检查

在优化性能时，使用节点模式专有的关系标签可以删除节点上的标签筛选。考虑一个图表模型，其中关系`likes`仅用于定义两个`person`节点之间的关系。我们可以编写以下查询来找到这种模式：

```
MATCH (n:person)-[:likes]->(m:person)
RETURN n, m
```

n 和 m 上的`person`标签检查是多余的，因为我们将关系定义为只有当两者都属于该类型时才会出现`person`。为了优化性能，我们可以按如下方式编写查询：

```
MATCH (n)-[:likes]->(m)
RETURN n, m
```

当属性仅限于单个节点标签时，也可以应用此模式。假设只有`person`节点具有该属性`email`，因此验证节点标签是否匹配`person`是多余的。将此查询写成：

```
MATCH (n:person)
WHERE n.email = 'xxx@gmail.com'
RETURN n
```

比将此查询写成以下方式效率要低：

```
MATCH (n)
WHERE n.email = 'xxx@gmail.com'
RETURN n
```

只有在性能很重要并且在建模过程中要进行检查以确保这些边缘标签不会被重复用于涉及其他节点标签的模式时，才应采用这种模式。如果您稍后在另一个节点标签上引入一个email属性（例如）company，则这两个版本的查询结果将有所不同。

尽可能指定边缘标签

在图案中指定边缘时，建议尽可能提供边缘标签。以以下示例查询为例，该查询用于将居住在城市中的所有人与访问过该城市的所有人联系起来。

```
MATCH (person)-->(city {country: "US"})-->(anotherPerson)
RETURN person, anotherPerson
```

如果您的图表模型使用多个边缘标签将人们链接到城市以外的节点，则由于不指定结束标签，Neptune将需要评估其他路径，这些路径稍后将被丢弃。在上面的查询中，由于没有给出边缘标签，因此引擎会先做更多工作，然后筛选出值以获得正确的结果。上述查询的更好版本可能是：

```
MATCH (person)-[:livesIn]->(city {country: "US"})-[:visitedBy]->(anotherPerson)
RETURN person, anotherPerson
```

这不仅有助于评估，而且使查询计划器能够创建更好的计划。您甚至可以将此最佳实践与冗余节点标签检查相结合，以删除城市标签检查并将查询写成：

```
MATCH (person)-[:livesIn]->({country: "US"})-[:visitedBy]->(anotherPerson)
RETURN person, anotherPerson
```

尽可能避免使用 WITH 子句

OpenCypher 中的 WITH 子句充当边界，在此之前的所有内容都执行完毕，然后将结果值传递给查询的其余部分。当你需要临时聚合或想要限制结果数量时，需要使用 WITH 子句，但除此之外，你应

该尽量避免使用 WITH 子句。一般指导是删除这些简单的 WITH 子句（不包括聚合、排序依据或限制），以使查询计划器能够处理整个查询，从而创建全局最优计划。举个例子，假设你写了一个查询来返回居住在India以下地区的所有人：

```
MATCH (person)-[:lives_in]->(city)
WITH person, city
MATCH (city)-[:part_of]->(country {name: 'India'})
RETURN collect(person) AS result
```

在上述版本中，WITH 子句限制了之前(person)-[:lives_in]->(city)模式的位置(city)-[:part_of]->(country {name: 'India'})（限制性更强）。这使得该计划不太理想。对此查询的优化是删除 WITH 子句，让计划者计算出最佳计划。

```
MATCH (person)-[:lives_in]->(city)
MATCH (city)-[:part_of]->(country {name: 'India'})
RETURN collect(person) AS result
```

尽早在查询中放置限制性筛选器

在所有情况下，在查询中尽早放置筛选器有助于减少查询计划必须考虑的中间解决方案。这意味着执行查询所需的内存和计算资源更少。

以下示例可帮助您了解这些影响。假设你写了一个查询来返回所有居住在里面的人India。查询的一个版本可能是：

```
MATCH (n)-[:lives_in]->(city)-[:part_of]->(country)
WITH country, collect(n.firstName + " "  + n.lastName) AS result
WHERE country.name = 'India'
RETURN result
```

上述版本的查询并不是实现此用例的最佳方式。筛选器稍后country.name = 'India'会出现在查询模式中。它将首先收集所有人员及其居住地，然后按国家对他们进行分组，然后仅筛选该群组country.name = India。仅查询居住在里面的人India然后执行收集聚合的最佳方式。

```
MATCH (n)-[:lives_in]->(city)-[:part_of]->(country)
WHERE country.name = 'India'
RETURN collect(n.firstName + " "  + n.lastName) AS result
```

一般规则是在引入变量后尽快放置过滤器。

明确检查属性是否存在

根据 OpenCypher 语义，当访问属性时，它等同于可选联接，即使该属性不存在，也必须保留所有行。如果您根据图表架构知道该实体将始终存在特定属性，则显式检查该属性的存在可以让查询引擎创建最佳计划并提高性能。

考虑一个图形模型，其中类型的节点`person`始终具有属性`name`。与其这样做，不如这样做：

```
MATCH (n:person)
RETURN n.name
```

使用 IS NOT NOT NULL 检查明确验证查询中是否存在属性：

```
MATCH (n:person)
WHERE n.name IS NOT NULL
RETURN n.name
```

不要使用命名路径（除非是必需的）

查询中的命名路径总是会产生额外的成本，这可能会增加延迟和内存使用量的损失。请考虑以下查询：

```
MATCH p = (n)-[:commentedOn]->(m)
WITH p, m, n, n.score + m.score as total
WHERE total > 100
MATCH (m)-[:commentedON]->(o)
WITH p, m, n, distinct(o) as o1
RETURN p, m.name, n.name, o1.name
```

在上面的查询中，假设我们只想知道节点的属性，那么就没有必要使用路径 “p”。通过将命名路径指定为变量，使用 DISTINCT 的聚合操作在时间和内存使用方面都将变得昂贵。上述查询的更优化的版本可能是：

```
MATCH (n)-[:commentedOn]->(m)
WITH m, n, n.score + m.score as total
WHERE total > 100
MATCH (m)-[:commentedON]->(o)
WITH m, n, distinct(o) as o1
RETURN m.name, n.name, o1.name
```

避免收集 (不同 ())

每当要形成包含不同值的列表时，都会使用 COLLECT (DISTINCT ())。COLLECT 是一个聚合函数，分组是根据同一语句中投影的其他键来完成的。当使用 distinct 时，输入会被分成多个块，其中每个区块表示一个要减少的组。随着小组数量的增加，性能将受到影响。在 Neptune 中，在实际收集/形成列表之前执行 DISTINCT 要有效得多。这允许直接在整个区块的分组键上进行分组。

请考虑以下查询：

```
MATCH (n:Person)-[:commented_on]->(p:Post)
WITH n, collect(distinct(p.post_id)) as post_list
RETURN n, post_list
```

编写此查询的更优方法是：

```
MATCH (n:Person)-[:commented_on]->(p:Post)
WITH DISTINCT n, p.post_id as postId
WITH n, collect(postId) as post_list
RETURN n, post_list
```

检索所有属性值时，最好使用属性函数而不是单个属性查找

该`properties()`函数用于返回包含实体所有属性的地图，并且比单独返回属性要有效得多。

假设您的`Person`节点包含 5 个属性`firstName`、`lastName`、`age`、`dept`、`company`、和，则首选以下查询：

```
MATCH (n:Person)
WHERE n.dept = 'AWS'
RETURN properties(n) as personDetails
```

而不是使用：

```
MATCH (n:Person)
WHERE n.dept = 'AWS'
RETURN n.firstName, n.lastName, n.age, n.dept, n.company

=== OR ===

MATCH (n:Person)
```

```
WHERE n.dept = 'AWS'
RETURN {firstName: n.firstName, lastName: n.lastName, age: n.age,
department: n.dept, company: n.company} as personDetails
```

在查询之外执行静态计算

建议在客户端解析静态计算（简单的数学/字符串运算）。以这个例子为例，你想查找所有比作者大一岁或小于一岁的人：

```
MATCH (m:Message)-[:HAS_CREATOR]->(p:person)
WHERE p.age <= ($age + 1)
RETURN m
```

在这里$age，通过参数注入到查询中，然后将其添加到固定值中。然后将该值与进行比较p.age。相反，更好的方法是在客户端进行加法，并将计算出的值作为参数 $ageplusone 传递。这有助于查询引擎创建优化的计划，并避免对每个传入的行进行静态计算。遵循这些指导原则，更有效的查询版本是：

```
MATCH (m:Message)-[:HAS_CREATOR]->(p:person)
WHERE p.age <= $ageplusone
RETURN m
```

使用 UNWIND 而不是单个语句进行批量输入

每当需要对不同的输入执行相同的查询时，与其为每个输入执行一个查询，不如为一批输入运行一个查询，性能会更高。

如果你想在一组节点上合并，一个选择是为每个输入运行一个合并查询：

```
MERGE (n:Person {`~id`: $id})
SET n.name = $name, n.age = $age, n.employer = $employer
```

带参数：

```
params = {id: '1', name: 'john', age: 25, employer: 'Amazon'}
```

需要对每个输入执行上述查询。虽然这种方法行得通，但可能需要为大量输入执行许多查询。在这种情况下，批处理可能有助于减少在服务器上执行的查询数量，并提高整体吞吐量。

请使用以下模式：

```
UNWIND $persons as person
MERGE (n:Person {`~id`: person.id})
SET n += person
```

带参数：

```
params = {persons: [{id: '1', name: 'john', age: 25, employer: 'Amazon'},
{id: '2', name: 'jack', age: 28, employer: 'Amazon'},
{id: '3', name: 'alice', age: 24, employer: 'Amazon'}...]}
```

建议尝试不同的批次大小，以确定哪种批次最适合您的工作负载。

最好为节点/关系使用自定义 ID

Neptune 允许用户在节点和关系上显式分配 ID。ID 在数据集中必须是全局唯一且具有确定性才有用。确定性 ID 可以像属性一样用作查找或筛选机制；但是，从查询执行的角度来看，使用 ID 比使用属性要优化得多。使用自定义 ID 有几个好处-

- 现有实体的属性可以为空，但是 ID 必须存在。这允许查询引擎在执行期间使用优化的联接。
- 当执行并发突变查询时，当使用 ID 访问节点时，出现并发修改异常 (CME) 的可能性会大大降低，因为由于其强制唯一性，对于 ID 的锁比对属性的锁要少。
- 使用 ID 可以避免创建重复数据的机会，因为 Neptune 强制执行 ID 的唯一性，这与属性不同。

以下查询示例使用自定义 ID：

> **Note**
>
> 该属性`~id`用于指定 ID，而`id`仅作为任何其他属性存储。

```
CREATE (n:Person {`~id`: '1', name: 'alice'})
```

不使用自定义 ID：

```
CREATE (n:Person {id: '1', name: 'alice'})
```

如果使用后一种机制，则不会强制执行唯一性，您可以稍后执行查询：

```
CREATE (n:Person {id: '1', name: 'john'})
```

这将创建第二个id=1名为 named 的节点john。在这种情况下，你现在将有两个节点id=1，每个节点都有不同的名称-（alice 和 john）。

避免在查询中进行~id计算

在查询中使用自定义 ID 时，请务必在查询之外执行静态计算，并在参数中提供这些值。当提供静态值时，引擎能够更好地优化查找并避免扫描和筛选这些值。

如果要在数据库中存在的节点之间创建边，可以选择以下方法：

```
UNWIND $sections as section
MATCH (s:Section {`~id`: 'Sec-' + section.id})
MERGE (s)-[:IS_PART_OF]->(g:Group {`~id`: 'g1'})
```

带参数：

```
parameters={sections: [{id: '1'}, {id: '2'}]}
```

在上面的查询中id，正在查询中计算该节的。由于计算是动态的，因此引擎无法静态内联 ID，最终会扫描所有节点。然后，引擎对所需的节点执行后筛选。如果数据库中有许多节点，这可能会很昂贵。

实现这一目标的更好方法是在传递到数据库的 ID 中Sec-加上前缀：

```
UNWIND $sections as section
MATCH (s:Section {`~id`: section.id})
MERGE (s)-[:IS_PART_OF]->(g:Group {`~id`: 'g1'})
```

带参数：

```
parameters={sections: [{id: 'Sec-1'}, {id: 'Sec-2'}]}
```

使用 SPARQL 的 Neptune 最佳实践

当您将 SPARQL 查询语言与 Neptune 一起使用时，请遵循以下最佳实践。有关在 Neptune 中使用 SPARQL 的信息，请参阅使用 SPARQL 访问 Neptune 图形。

默认查询所有命名图形

Amazon Neptune 将每个三元组与一个命名图形相关联。默认图形定义为所有命名图形的并集。

如果提交 SPARQL 查询而未通过 GRAPH 关键字或 FROM NAMED 之类的构造明确指定图形，Neptune 将始终考虑数据库实例中的所有三元组。例如，以下查询从 Neptune SPARQL 端点返回所有三元组：

```
SELECT * WHERE { ?s ?p ?o }
```

显示在多个图形中的三元组将仅返回一次。

有关默认图形规范的信息，请参阅 SPARQL 1.1 查询语言规范的 RDF 数据集一节。

为加载指定命名图形

Amazon Neptune 将每个三元组与一个命名图形相关联。如果加载、插入或更新三元组时未指定命名图形，Neptune 使用 URI `http://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph` 定义的回退命名图形。

如果您使用的是 Neptune 批量加载程序，则可以使用 `parserConfiguration: namedGraphUri` 参数指定要用于所有三元组（或第四个位置空白的四元组）的命名图形。有关 Neptune 加载程序 Load 命令语法的信息，请参阅the section called “加载程序命令”。

在查询的 FILTER、FILTER...IN 和 VALUES 之间进行选择

有三种方法在 SPARQL 查询中注入值：FILTER、FILTER...IN 和 VALUES。

例如，假设您要在单个查询内查找多个人员的好友。您可以使用 FILTER 构建查询，如下所示：

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. FILTER (?s = ex:person1 || ?s = ex:person2)}
```

这将返回图形中 ?s 绑定到 ex:person1 或 ex:person2 并且传出边缘标记为 foaf:knows 的所有三元组。

您也可以创建一个使用 FILTER...IN 的查询，该查询返回等同的结果：

```
PREFIX ex: <https://www.example.com/>
```

```
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. FILTER (?s IN (ex:person1, ex:person2))}
```

您也可以使用 VALUES 创建查询，在此例中也会返回等同结果：

```
PREFIX ex: <https://www.example.com/>
PREFIX foaf : <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o
WHERE {?s foaf:knows ?o. VALUES ?s {ex:person1 ex:person2}}
```

尽管在许多情况下，这些查询在语义上是等效的，但在某些情况下，两个 FILTER 变体不同于 VALUES 变体：

- 第一种情况是在您注入重复值时，例如将同一个人注入两次。在这种情况下，VALUES 查询会在结果中包含重复项。您可通过将 DISTINCT 添加到 SELECT 子句来显式消除此类重复。但是，在某些情况下，您可能希望在查询结果中包含重复项用于冗余值注入。

  而 FILTER 和 FILTER...IN 在相同值多次显示时仅提取该值一次。
- 第二种情况涉及 VALUES 始终执行完全匹配，而 FILTER 在某些情况下可能会应用类型提升并执行模糊匹配。

  例如，当您在 values 子句中包含类似于 "2.0"^^xsd:float 的文本时，VALUES 查询与此文本完全匹配，包括文本值和数据类型。

  相比之下，FILTER 会为这些数字文本生成模糊匹配。该匹配会包括值相同但数字数据类型（例如 xsd:double）不同的文本。

> **Note**
>
> 枚举字符串文本或 URI 时，FILTER 与 VALUES 的行为没有区别。

FILTER 与 VALUES 之间的区别会影响优化以及所造成的查询计算策略。除非您的用例需要模糊匹配，否则我们建议您使用 VALUES，因为它会避免查找与类型转换相关的特殊情况。因此，VALUES 通常会生成更高效的查询，运行速度更快，而且成本更低。

Amazon Neptune 限制

区域

亚马逊 Neptune 在以下 AWS 地区上市：

- 美国东部（弗吉尼亚州北部）：us-east-1
- 美国东部（俄亥俄州）：us-east-2
- 美国西部（北加利福尼亚）：us-west-1
- 美国西部（俄勒冈州）：us-west-2
- 加拿大（中部）：ca-central-1
- 南美洲（圣保罗）：sa-east-1
- 欧洲地区（斯德哥尔摩）：eu-north-1
- 欧洲地区（爱尔兰）：eu-west-1
- 欧洲地区（伦敦）：eu-west-2
- 欧洲地区（巴黎）：eu-west-3
- 欧洲地区（法兰克福）：eu-central-1
- 中东（巴林）：me-south-1
- 中东（阿联酋）：me-central-1
- 以色列（特拉维夫）：il-central-1
- 非洲（开普敦）：af-south-1
- 亚太地区（香港）：ap-east-1
- 亚太地区（东京）：ap-northeast-1
- 亚太地区（首尔）：ap-northeast-2
- 亚太地区（大阪）：ap-northeast-3
- 亚太地区（新加坡）：ap-southeast-1
- 亚太地区（悉尼）：ap-southeast-2
- 亚太地区（孟买）：ap-south-1
- 中国（北京）：cn-north-1
- 中国（宁夏）：cn-northwest-1
- AWS GovCloud （美国西部）：us-gov-west-1

- AWS GovCloud（美国东部）：`us-gov-east-1`

中国区域的差异

与许多 AWS 服务一样，Amazon Neptune 在中国（北京）和中国（宁夏）的运营方式与其他 AWS 地区略有不同。

例如，当 Neptune ML 使用 Amazon API Gateway 创建其导出服务时，IAM 身份验证默认处于启用状态。在中国区域，更改该选项的过程与其它区域略有不同。

此处解释了这些差异和其它差异。

存储集群卷的最大大小

在除中国以外的所有受支持区域，Neptune 集群的最大容量可以增长到 128 TiB (TiB)，中国除外，其上限为 6 GovCloud 4 TiB。对于从版本：1.0.2.2（2020 年 3 月 9 日）开始的所有引擎版本都是如此。请参阅 Amazon Neptune 存储、可靠性和可用性。

支持的数据库实例大小

Neptune 支持不同区域中的不同 AWS 数据库实例类。如需了解指定区域中支持哪些类，请转到 Amazon Neptune 定价并选择您感兴趣的区域。

每个 AWS 账户的限额

对于某些管理特征，Amazon Neptune 使用与 Amazon Relational Database Service (Amazon RDS) 共享的操作技术。

每个 AWS 账户对每个区域可以创建的 Amazon Neptune 和 Amazon RDS 资源的数量都有限制。这些资源包括数据库实例和数据库集群。

在您达到某一资源的限制时，再进行创建该资源的调用就会失败并引发异常。

有关 Amazon Neptune 和 Amazon RDS 之间共享的限制列表，请参阅《Amazon RDS 用户指南》中的 Amazon RDS 中的限制。

连接到 Neptune 需要 VPC

Amazon Neptune 是一项仅针对虚拟私有云 (VPC) 的服务。

此外，不允许从 VPC 外部访问实例。

Neptune 需要 SSL

从引擎版本 `1.0.4.0` 开始，Amazon Neptune 仅允许使用安全套接字层 (SSL) 连接通过 HTTPS 连接到任何实例或集群端点。

Neptune 需要 TLS 版本 1.2，并使用以下强密码套件：

- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`

可用区和数据库子网组

Amazon Neptune 要求每个集群都有一个数据库子网组，该组至少在两个受支持的可用区 (AZ) 中有子网。

我们建议在不同的可用区中使用三个或更多子网。

HTTP 请求负载最大值 (150MB)

Gremlin 和 SPARQL HTTP 请求的总大小必须小于 150 MB。如果请求超过此大小，Neptune 返回 `HTTP 400: BadRequestException`。

此限制不适用于 Gremlin 连接 WebSockets 。

Gremlin 实施的不同之处

Amazon Neptune Gremlin 实施具有特定的实施细节，可能会不同于其它 Gremlin 实施。

有关更多信息，请参阅 Amazon Neptune 中的 Gremlin 标准合规性。

Neptune 不支持字符串数据中的 null 字符

Neptune 不支持字符串中的 null 字符。Gremlin 和 openCypher 的属性图数据以及 RDF/SPARQL 数据都是如此。

URI 中的 SPARQL UPDATE LOAD

URI 中的 `SPARQL UPDATE LOAD` 仅适用于位于同一 VPC 中的资源。

这包括创建了 Amazon S3 VPC 端点的集群所在区域中的 Amazon S3 URL。

Amazon S3 URL 必须是 HTTPS，并且任何身份验证都必须包含在 URL 中。有关更多信息，请参阅《Amazon Simple Storage Service API 参考》中的验证请求：使用查询参数。

有关创建 VPC 端点的信息，请参阅 创建 Amazon S3 VPC 端点。

如果您需要从文件加载数据，我们建议使用 Amazon Neptune 加载程序 API。有关更多信息，请参阅使用 Amazon Neptune 批量加载程序收集数据。

> **Note**
>
> Amazon Neptune 加载程序 API 是非 ACID 的。

IAM 身份验证和访问控制

在版本 1.2.0.0 之前的 Neptune 引擎版本中，仅在数据库集群级别支持 IAM 身份验证和访问控制。但是，从版本 `1.2.0.0` 开始，您可以使用 IAM policy 中的条件键在更精细的级别上控制基于查询的访问权限。有关更多信息，请参阅在 Neptune 数据访问策略语句中使用查询操作和亚马逊 Neptune 中的 AWS Identity and Access Management (IAM) 概述

亚马逊 Neptune 控制台需要NeptuneReadOnlyAccess权限。您可以通过撤消此访问权限来限制对 IAM 用户的访问。有关更多信息，请参阅AWS 亚马逊 Neptune 的托管（预定义）策略

Amazon Neptune 不支持基于用户名/密码的访问控制。

WebSocket 并发连接和最长连接时间

每个 Neptune 数据库实例的并发 WebSocket 连接数有限制。当达到该限制时，Neptune 会限制任何打开新 WebSocket 连接的请求，以防止耗尽所有分配的堆内存。

对于 Neptune 支持的所有大型实例类型和所有无服务器实例，最大并发 WebSocket 连接数为 32K (32,768)。

下表列出了较小实例类型的最大并发 WebSocket 连接数：

实例类型	最大并发 WebSocket 连接数
db.t3.medium	512
db.t4g.medium	512
db.r5.large	2,048
db.r5d.large	2,048
db.r5.xlarge	4,096
db.r5.2xlarge	8192
db.r5d.2xlarge	8192
db.r5.4xlarge	16,384
db.r5d.4xlarge	16,384
db.r6g.large	2,048
db.r6gd.large	2,048
db.r6g.xlarge	4,096
db.r6gd.xlarge	4,096
db.r6g.2xlarge	8192
db.r6gd.2xlarge	8192
db.r6g.4xlarge	16,384
db.r6gd.4xlarge	16,384
db.x2g.large	2,048

实例类型	最大并发 WebSocket 连接数
db.x2gd.large	2,048
db.x2g.xlarge	4,096
db.x2gd.xlarge	4,096
db.x2iedn.xlarge	4,096
db.x2g.2xlarge	8192
db.x2gd.2xlarge	8192
db.x2g.4xlarge	16,384
db.x2gd.4xlarge	16,384
db.x2iedn.2xlarge	16,384
db.x2iezn.2xlarge	16,384
无服务器	32,768
（其它大型实例类型）	32,768

> **Note**
>
> 从 Neptune 引擎版本 1.1.0.0 开始，Neptune 不再支持 R4 实例类型。

当客户端正常关闭连接时，该关闭将立即反映在打开的连接计数中。

如果客户端没有关闭连接，则连接可能会在 20 到 25 分钟的空闲超时后自动关闭（空闲超时是指从客户端收到最后一条消息以来经过的时间）。但是，只要未达到空闲超时，Neptune 就会无限期地将连接保持为打开状态。

启用 IAM 身份验证后，如果 WebSocket 连接尚未关闭，则连接总是在建立 10 天后的几分钟内断开连接。

对于属性和标签的限制

对图形中可以具有的顶点和边数或 RDF 四边形的数量没有限制。

对任何一个顶点或边可以具有的属性或标签的数量也没有限制。

对单个属性或标签的大小的限制为 55 MB。在 RDF 术语中，这意味着 RDF 四边形的任何列（S、P、O 或 G）中的值不能超过 55 MB。

如果需要将较大的对象（例如图像）与图形中的顶点或节点相关联，则可以将其存储为 Amazon S3 中的文件，并将 Amazon S3 路径用作属性或标签。

影响 Neptune 批量加载程序的限制

一次排队的 Neptune 批量加载任务数不能超过 64 个。

Neptune 仅跟踪最近的 1024 个批量加载任务。

Neptune 仅存储每个任务的后 10000 个错误详细信息。

使用其他AWS服务

您可以将 Amazon Neptune 与许多其它 AWS 服务结合使用：

Neptune 与其它服务集成

- AWS Glue – AWS Glue 是一项无服务器数据集成服务，可以帮助您对数据执行提取、转换、加载 (ETL) 任务。

  Neptune 提供了一个开源库 neptune-python-utilities，可简化在 Glue 任务中使用 Python 和 Gremlin。还支持使用 Neo4j Spark 连接器运行 Scala 和 openCypher Glue 任务。

- Amazon SageMaker – Amazon SageMaker 是一个功能齐全的机器学习平台，用于构建、训练和部署高质量的机器学习模型。

  Neptune 通过两种主要方式与 SageMaker 集成：

  - Neptune 为 Jupyter 笔记本提供一个开源 Python 软件包，可以在 GitHub 上的 Neptune 图形笔记本项目中找到该软件包。该软件包包含一组 Jupyter 魔术命令、教程笔记本和代码示例，这些示例在交互式编码环境中提供，您可以在其中学习图形技术和 Neptune。Neptune 为由 SageMaker 托管的 Jupyter 笔记本提供了一个完全托管式环境，并自动链接到开源 Neptune 图形笔记本项目中的笔记本。
  - Neptune ML 特征可以在几小时而不是几周内在大型图形上构建和训练有用的机器学习模型。为了实现这一目标，Neptune ML 使用由 Amazon SageMaker 和深度图表库 (DGL) 提供支持的图形神经网络 (GNN) 技术。

- AWS Lambda – AWS Lambda 函数在 Neptune 应用程序中有许多用途。

  有关如何将 Lambda 函数与任何流行的 Gremlin 驱动程序和语言变体一起使用的信息，以及用 Java、JavaScript 和 Python 编写的 Lambda 函数的具体示例，请参阅在 Amazon Neptune 中使用 AWS Lambda 函数。

- Amazon Athena – Amazon Athena 是一种交互式查询服务，让您能够轻松使用标准 SQL 分析 Amazon Simple Storage Service 和其它联合数据来源中的数据。

  Neptune 提供了一个连接到 Athena 的连接器，使 Athena 能够与存储在 Neptune 中的数据进行通信。

- AWS Database Migration Service (AWS DMS) – AWS Database Migration Service 是一项 AWS Web 服务，可用于将数据从一个数据库迁移到另一个数据库。

AWS DMS 可以快速安全地将支持的源数据库中的数据加载到 Neptune。源数据库在迁移过程中可保持完全正常运行，从而最大程度减少依赖于该数据库的应用程序停机时间。

- AWS Backup – AWS Backup 是一项完全托管式备份服务，可在云中以及本地方便地集中管理和自动执行跨 AWS 服务的数据备份。

  AWS Backup 允许您跨支持的 AWS 服务针对数据库、存储和计算使用集中式数据保护策略，以创建 Neptune 集群的自动定期快照。

- AWS SDK for pandas – AWS SDK for pandas（以前称为 AWS Data Wrangler 或 `awswrangler`）是一项 AWS 专业服务开源 python 计划，它将 `pandas` Python 数据分析库的功能扩展到 AWS，同时连接 `DataFrames` 和 30 多个 AWS 数据相关服务，包括 Neptune。

  除了 SDK 之外，还有一个关于如何在 Neptune 上使用它的教程，以及几个示例 Neptune 笔记本，即欺诈环检测、合成身份检测和物流分析。

- JDBC 驱动程序 – Neptune JDBC 驱动程序支持 openCypher、Gremlin、SQL-Gremlin 和 SPARQL 查询。

  JDBC 连接使您可以使用 Tableau 等商业智能 (BI) 工具轻松连接到 Neptune。

Neptune 工具和实用程序

Amazon Neptune 提供了许多工具和实用程序，可以简化和自动执行处理图形的过程。其中包括：

Amazon Neptune 工具

- 适用于 GraphQL 的 Amazon Neptune 实用程序 – 适用于 GraphQL 的 Amazon Neptune 实用程序是一款开源 Node.js 命令行工具，可以帮助您为 Neptune 属性图形数据库创建和维护 GraphQL API。对于具有可变数量的输入参数并返回可变数量的嵌套字段的 GraphQL 查询，这是一种创建 GraphQL 解析器的无代码方法。

适用于 GraphQL 的 Amazon Neptune 实用程序

适用于 GraphQL 的 Amazon Neptune 实用程序是一款开源 Node.js 命令行工具，可以帮助您为 Neptune 属性图形数据库创建和维护 GraphQL API（它尚未处理 RDF 数据）。对于具有可变数量的输入参数并返回可变数量的嵌套字段的 GraphQL 查询，这是一种创建 GraphQL 解析器的无代码方法。

它已作为开源项目发布，网址为 https://github.com/aws/ amazon-neptune-for-graphql。

您可以像这样使用 NPM 来安装该实用程序（有关详细信息，请参阅安装和设置）：

```
npm i @aws/neptune-for-graphql -g
```

该实用程序可以发现现有 Neptune 属性图的图形架构，包括节点、边缘、属性和边缘基数。然后，它生成一个 GraphQL 架构，其中包含将 GraphQL 类型映射到节点和数据库边缘所需的指令，并自动生成解析器代码。解析器代码旨在通过仅返回由 GraphQL 查询请求的数据来最大限度地减少延迟。

也可以从现有的 GraphQL 架构和空的 Neptune 数据库开始，然后让该实用程序推理所需的指令，以便将该 GraphQL 架构映射到要加载到数据库中的数据的节点和边缘。或者，您可以从已经创建或修改的 GraphQL 架构和指令开始。

该实用程序能够创建其管道所需的所有AWS资源，包括 AWS AppSync API、IAM 角色、数据源、架构和解析器，以及查询 Neptune 的 Lamb AWS da 函数。

> **Note**
>
> 此处的命令行示例假设使用 Linux 控制台。如果使用的是 Windows，请将行末尾的反斜杠 ('\') 替换为插入符号 ('^')。

主题

安装和设置适用于 GraphQL 的 Amazon Neptune 实用程序

如果要将该实用程序与现有 Neptune 数据库一起使用，则需要它才能连接到数据库端点。默认情况下，只能从 Neptune 数据库所在的 VPC 内访问该数据库。

由于该实用程序是 Node.js 命令行工具，因此必须安装 Node.js（版本 18 或更高版本）才能运行该实用程序。要在与 Neptune 数据库相同的 EC2 实例上安装 Node.js，请按照此处的说明操作。运行该实用程序的最小大小实例为 t2.micro。在创建实例期间，从常见安全组下拉菜单中选择 Neptune 数据库 VPC。

但是，从引擎版本 1.2.0.0 开始，您可以为 Neptune 数据库创建一个可在 VPC 外部访问的公有端点。如果您创建了公有端点，则可以在本地计算机上安装 Node.js 和该实用程序。要在 macOS 或 Windows 上安装 Node.js，请访问 Node.js 网站以下载安装程序。

要在 EC2 实例或本地计算机上安装实用程序本身，请使用 NPM：

```
npm install aws-neptune-for-graphql -g
```

然后，您可以运行该实用程序的 help 命令来检查它安装是否正确：

```
neptune-for-graphql --help
```

您可能还需要安装 AWS CLI 以管理 AWS 资源。

扫描现有 Neptune 数据库中的数据

无论您是否熟悉 GraphQL，下面的命令都是创建 GraphQL API 的最快方法。这假设您已按照安装部分所述安装并配置了适用于 GraphQL 的 Neptune 实用程序，以便它连接到您的 Neptune 数据库的端点。

```
neptune-for-graphql \
```

```
  --input-graphdb-schema-neptune-endpoint (your neptune database endpoint):(port
 number) \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (your new GraphQL API name) \
  --output-resolver-query-https
```

该实用程序分析数据库以发现其中的节点、边缘和属性的架构。根据该架构，它推理带有关联查询和突变的 GraphQL 架构。然后，它会创建一个 AppSync GraphQL API 以及使用它所需的AWS资源。这些资源包括一对 IAM 角色和一个包含 GraphQL 解析器代码的 Lambda 函数。

该实用程序完成后，您将在 AppSync控制台中找到一个新的 GraphQL API，名称与您在命令中分配的名称相同。要对其进行测试，请使用菜单上的 “ AppSync 查询” 选项。

如果您在向数据库添加更多数据后再次运行相同的命令，则会相应地更新 AppSync API 和 Lambda 代码。

要释放与该命令关联的所有资源，请运行：

```
neptune-for-graphql \
  --remove-aws-pipeline-name (your new GraphQL API name from above)
```

从没有指令的 GraphQL 架构开始

您可以从空的 Neptune 数据库开始，使用没有指令的 GraphQL 架构来创建数据并进行查询。以下命令会自动创建 AWS 资源来执行此操作：

```
neptune-for-graphql \
  --input-schema-file (your GraphQL schema file)
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (name for your new GraphQL API) \
  --create-update-aws-pipeline-neptune-endpoint (your Neptune database endpoint):(port
 number) \
  --output-resolver-query-https
```

GraphQL 架构文件必须包含 GraphQL 架构类型，如下面的 TODO 示例所示。该实用程序会分析您的架构，并根据您的类型创建扩展版本。它为存储在图形数据库中的节点添加查询和突变，如果您的架构具有嵌套类型，则它会添加作为边缘存储在数据库中的类型之间的关系。

该实用程序创建了一个 AppSync GraphQL API 以及所需的所有AWS资源。其中包括一对 IAM 角色和一个包含 GraphQL 解析器代码的 Lambda 函数。命令完成后，您可以使用您在控制台中指定的名称找到一个新的 GraphQL API。 AppSync 要对其进行测试，请在 AppSync 菜单中使用查询。

以下示例说明了其工作原理：

Todo 示例，从没有指令的 GraphQL 架构开始

在这个例子中，我们从一个没有指令的 Todo GraphQL 架构开始（可以在 *???samples???* 目录中找到该架构）。它包括以下两种类型：

```
type Todo {
  name: String
  description: String
  priority: Int
  status: String
  comments: [Comment]
}

type Comment {
  content: String
}
```

此命令处理待办事项架构和空的 Neptune 数据库的端点，以便在以下位置创建 GraphQL API：AWS AppSync

```
neptune-for-graphql /
  --input-schema-file ./samples/todo.schema.graphql \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name TodoExample \
  --create-update-aws-pipeline-neptune-endpoint (empty Neptune database endpoint):(port
 number) \
  --output-resolver-query-https
```

该实用程序在名为的输出文件夹中创建一个新文件`TodoExample.source.graphql`，并在中创建 GraphQL API。 AppSync该实用程序推理以下内容：

- 在待办事项类型中，它`@relationship`为一种新 CommentEdge 类型添加了。这指示解析器使用名为的图形数据库边缘将 Todo 连接到 Comment。 CommentEdge
- 它添加了一个名 TodoInput 为帮助查询和变更的新输入。
- 它为每种类型（Todo、Comment）添加了两个查询：一个查询用于使用输入中列出的一个 `id` 或任何一个类型字段检索单个类型，另一个查询用于检索使用该类型的输入进行筛选的多个值。
- 它为每种类型添加了三个突变：创建、更新和删除。要删除的类型是使用 `id` 或该类型的输入来指定的。这些突变会影响存储在 Neptune 数据库中的数据。

- 它为连接添加了两个突变：连接和删除。它们将 Neptune 使用的起始顶点和结束顶点的节点 ID 作为输入，连接是数据库中的边缘。

解析器通过名称识别查询和突变，但您可以按如下所示对其进行自定义。

以下是 TodoExample.source.graphql 文件的内容：

```
type Todo {
  _id: ID! @id
  name: String
  description: String
  priority: Int
  status: String
  comments(filter: CommentInput, options: Options): [Comment] @relationship(type:
 "CommentEdge", direction: OUT)
  bestComment: Comment @relationship(type: "CommentEdge", direction: OUT)
  commentEdge: CommentEdge
}

type Comment {
  _id: ID! @id
  content: String
}

input Options {
  limit: Int
}

input TodoInput {
  _id: ID @id
  name: String
  description: String
  priority: Int
  status: String
}

type CommentEdge {
  _id: ID! @id
}

input CommentInput {
  _id: ID @id
  content: String
```

```
}

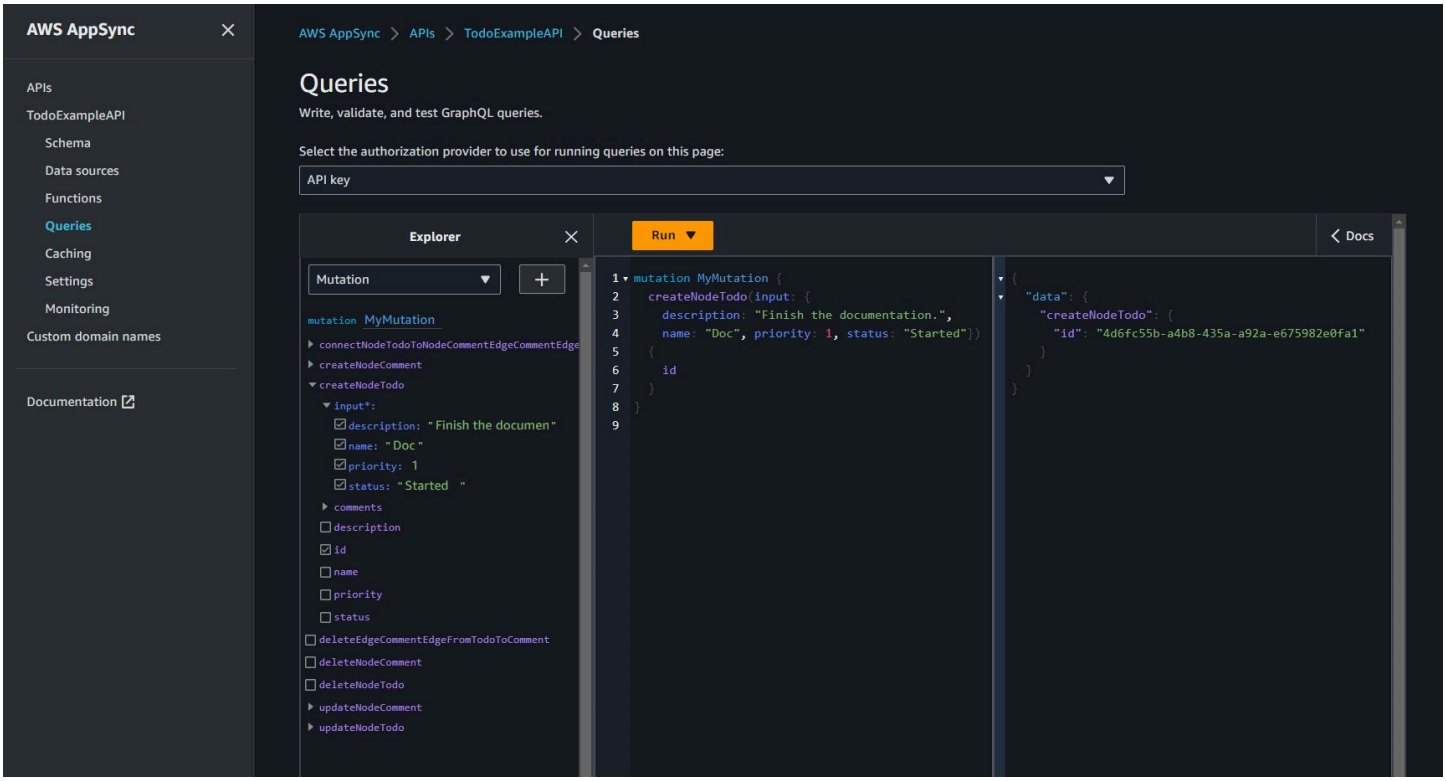
input Options {
  limit: Int
}

type Query {
  getNodeTodo(filter: TodoInput, options: Options): Todo
  getNodeTodos(filter: TodoInput): [Todo]
  getNodeComment(filter: CommentInput, options: Options): Comment
  getNodeComments(filter: CommentInput): [Comment]
}

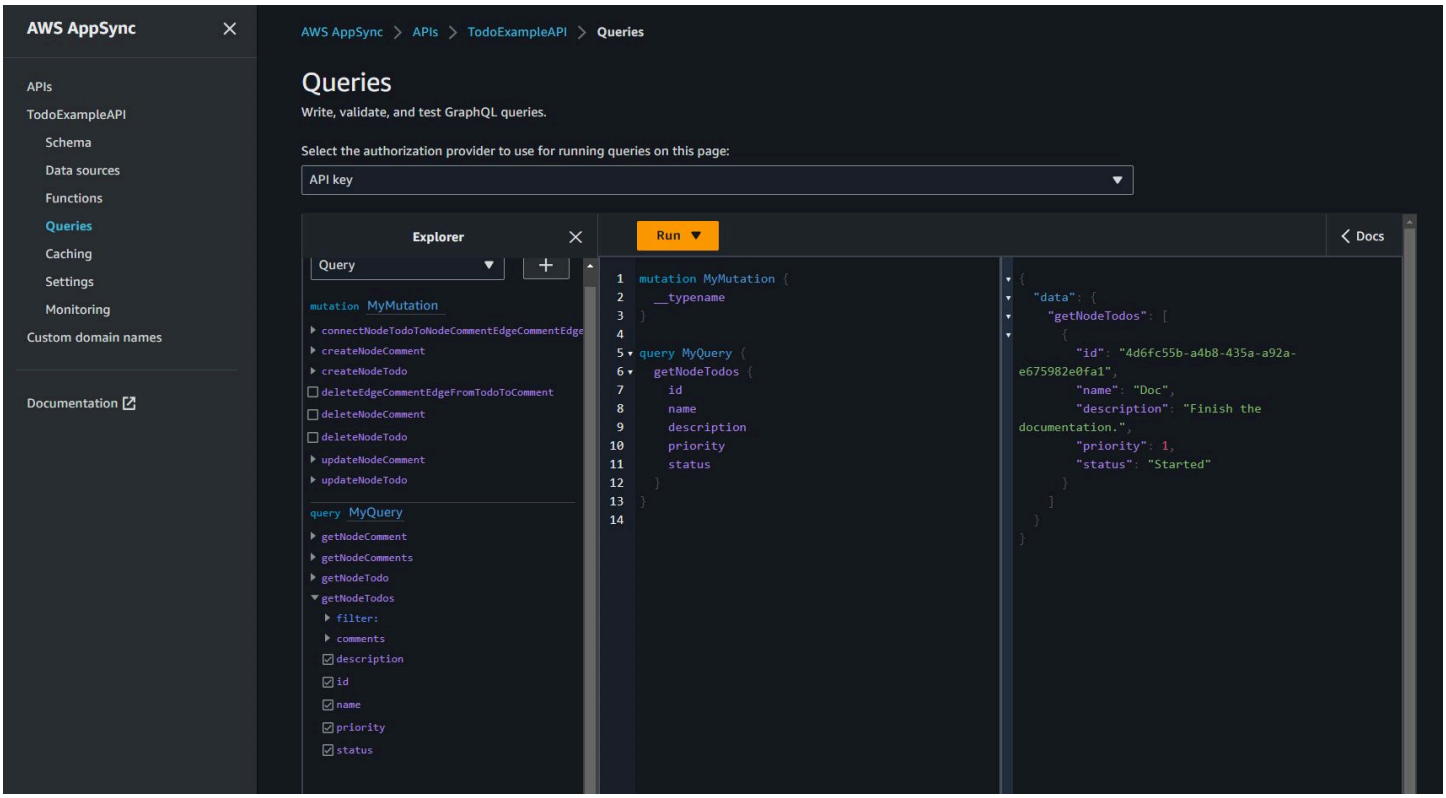
type Mutation {
  createNodeTodo(input: TodoInput!): Todo
  updateNodeTodo(input: TodoInput!): Todo
  deleteNodeTodo(_id: ID!): Boolean
  connectNodeTodoToNodeCommentEdgeCommentEdge(from_id: ID!, to_id: ID!): CommentEdge
  deleteEdgeCommentEdgeFromTodoToComment(from_id: ID!, to_id: ID!): Boolean
  createNodeComment(input: CommentInput!): Comment
  updateNodeComment(input: CommentInput!): Comment
  deleteNodeComment(_id: ID!): Boolean
}

schema {
  query: Query
  mutation: Mutation
}
```

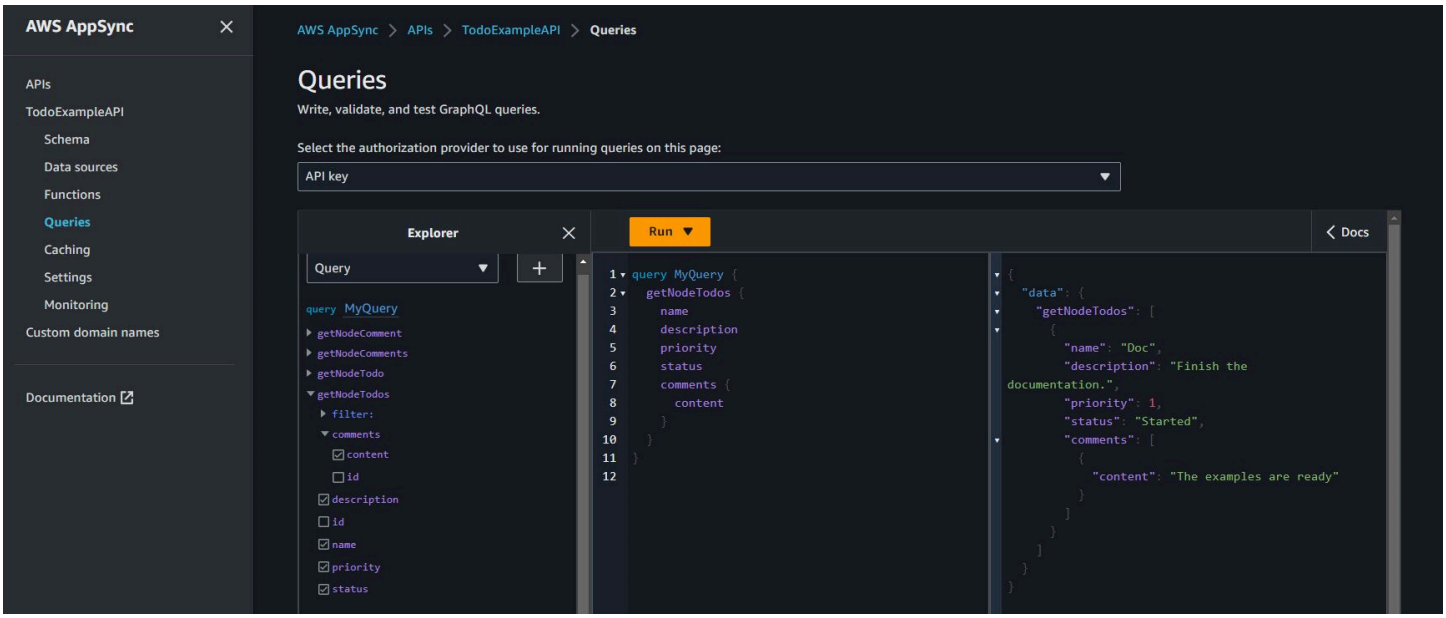
现在，您可以创建和查询数据。以下是用于测试新 GraphQL API 的 Queries 控制台的快照，在本例**TodoExampleAPI**中命名。 AppSync在中间的窗口中，Explorer 显示查询和突变的列表，您可以从中挑选查询、输入参数和返回字段。此屏幕截图显示了使用 `createNodeTodo` 突变创建 Todo 节点类型的过程：



此屏幕截图显示使用 `getNodeTodos` 查询来查询所有 Todo 节点：



使用 `createNodeComment` 创建 Comment（注释）后，您可以使用 `connectNodeTodoToNodeCommentEdgeCommentEdge` 突变通过指定其 ID 来连接它们。以下是检索 Todo 及其所附注释的嵌套查询：



如果要按照使用指令中所述对 `TodoExample.source.graphql` 文件进行更改，则可以使用编辑后的架构作为输入并再次运行该实用程序。然后，该实用程序将相应地修改 GraphQL API。

使用 GraphQL 架构的指令

您可以从已经有指令的 GraphQL 架构开始，使用如下命令：

```
neptune-for-graphql \
  --input-schema-file (your GraphQL schema file with directives) \
  --create-update-aws-pipeline \
  --create-update-aws-pipeline-name (name for your new GraphQL API) \
  --create-update-aws-pipeline-neptune-endpoint  (empty Neptune database
 endpoint):(port number) \
  --output-resolver-query-https
```

您可以修改该实用程序创建的指令，也可以将自己的指令添加到 GraphQL 架构中。以下是一些使用指令的方法：

运行该实用程序使其不会生成突变

要防止该实用程序在 GraphQL API 中生成突变，请在 `neptune-for-graphql` 命令中使用 `--output-schema-no-mutations` 选项。

@alias 指令

@alias 指令可以应用于 GraphQL 架构类型或字段。它在图形数据库和 GraphQL 架构之间映射不同的名称。语法如下：

```
@alias(property: (property name))
```

在下面的示例中，airport 是映射到 Airport GraphQL 类型的图形数据库节点标签，desc 是映射到 description 字段的图形节点属性（请参阅航线示例）：

```
type Airport @alias(property: "airport") {
  city: String
  description: String @alias(property: "desc")
}
```

请注意，标准 GraphQL 格式要求使用 Pascal 大小写的类型名称和驼峰式大小写的字段名称。

@relationship 指令

@relationship 指令将嵌套的 GraphQL 类型映射到图形数据库边缘。语法如下：

```
@relationship(edgeType: (edge name), direction: (IN or OUT))
```

以下是一个示例命令：

```
type Airport @alias(property: "airport") {
  ...
  continentContainsIn: Continent @relationship(edgeType: "contains", direction: IN)
  countryContainsIn: Country @relationship(edgeType: "contains", direction: IN)
  airportRoutesOut(filter: AirportInput, options: Options): [Airport]
 @relationship(edgeType: "route", direction: OUT)
  airportRoutesIn(filter: AirportInput, options: Options): [Airport]
 @relationship(edgeType: "route", direction: IN)
}
```

您可以在 TODO 示例和航线示例中找到 @relationship 指令。

@graphQuery 和 @cypher 指令

您可以定义 openCypher 查询来解析字段值、添加查询或添加突变。例如，这将在 Airport 类型中添加新的 outboundRoutesCount 字段来计算出站路线：

```
type Airport @alias(property: "airport") {
  ...
  outboundRoutesCount: Int @graphQuery(statement: "MATCH (this)-[r:route]->(a) RETURN
 count(r)")
}
```

以下是新查询和突变的示例：

```
type Query {
  getAirportConnection(fromCode: String!, toCode: String!): Airport \
    @cypher(statement: \
      "MATCH (:airport{code: '$fromCode'})-[:route]->(this:airport)-[:route]-
>(:airport{code:'$toCode'})")
}

type Mutation {
  createAirport(input: AirportInput!): Airport @graphQuery(statement: "CREATE
 (this:airport {$input}) RETURN this")
  addRoute(fromAirportCode:String, toAirportCode:String, dist:Int): Route \
    @graphQuery(statement: \
     "MATCH (from:airport{code:'$fromAirportCode'}),
 (to:airport{code:'$toAirportCode'}) \
      CREATE (from)-[this:route{dist:$dist}]->(to) \
      RETURN this")
}
```

请注意，如果省略 RETURN，则解析器会假定关键字 `this` 是返回范围。

也可以使用 Gremlin 查询添加查询或突变：

```
type Query {
  getAirportWithGremlin(code:String): Airport \
    @graphQuery(statement: "g.V().has('airport', 'code', '$code').elementMap()")  #
 single node
  getAirportsWithGremlin: [Airport] \
    @graphQuery(statement: "g.V().hasLabel('airport').elementMap().fold()")       #
 list of nodes
  getCountriesCount: Int \
    @graphQuery(statement: "g.V().hasLabel('country').count()")                   #
 scalar
}
```

目前，Gremlin 查询仅限于返回标量值的查询，或 `elementMap()` 对应于单个节点，或 `elementMap().fold()` 对应于节点列表。

`@id` 指令

`@id` 指令标识映射到 `id` 图形数据库实体的字段。像 Amazon Neptune 这样的图形数据库对于在批量导入期间分配或自动生成的节点和边缘始终具有唯一的 `id`。例如：

```
type Airport {
  _id: ID! @id
  city: String
  code: String
}
```

保留类型、查询和突变名称

该实用程序会自动生成查询和突变以创建可正常运行的 GraphQL API。这些名称的模式被解析器识别并保留。以下是类型 `Airport` 和连接类型 `Route` 的示例：

`Options` 类型已保留。

```
input Options {
  limit: Int
}
```

`filter` 和 `options` 函数参数已保留。

```
type Query {
  getNodeAirports(filter: AirportInput, options: Options): [Airport]
}
```

查询名称的 getNode 前缀已保留，诸如 `createNode`、`updateNode`、`deleteNode`、`connectNode`、`deleteNode`、`updateEdge` 和 `deleteEdge` 之类的突变名称的前缀已保留。

```
type Query {
  getNodeAirport(id: ID, filter: AirportInput): Airport
  getNodeAirports(filter: AirportInput): [Airport]
}

type Mutation {
```

```
  createNodeAirport(input: AirportInput!): Airport
  updateNodeAirport(id: ID!, input: AirportInput!): Airport
  deleteNodeAirport(id: ID!): Boolean
  connectNodeAirportToNodeAirportEdgeRout(from: ID!, to: ID!, edge: RouteInput!): Route
  updateEdgeRouteFromAirportToAirport(from: ID!, to: ID!, edge: RouteInput!): Route
  deleteEdgeRouteFromAirportToAirport(from: ID!, to: ID!): Boolean
}
```

将更改应用于 GraphQL 架构

您可以修改 GraphQL 源架构并再次运行该实用程序，从 Neptune 数据库中获取最新架构。每当该实用程序在数据库中发现新架构时，它都会生成一个新的 GraphQL 架构。

您也可以手动编辑 GraphQL 源架构，然后使用源架构而不是 Neptune 数据库端点作为输入再次运行该实用程序。

最后，您可以使用以下 JSON 格式将更改放入文件中：

```
[
  {
    "type": "(GraphQL type name)",
    "field": "(GraphQL field name)",
    "action": "(remove or add)",
    "value": "(value)"
  }
]
```

例如：

```
[
  {
    "type": "Airport",
    "field": "outboundRoutesCountAdd",
    "action": "add",
    "value":"outboundRoutesCountAdd: Int @graphQuery(statement: \"MATCH (this)-
[r:route]->(a) RETURN count(r)\")"
  },
  {
    "type": "Mutation",
    "field": "deleteNodeVersion",
    "action": "remove",
    "value": ""
```

```
  },
  {
    "type": "Mutation",
    "field": "createNodeVersion",
    "action": "remove",
    "value": ""
  }
]
```

然后，当您在命令中使用 `--input-schema-changes-file` 参数对此文件运行该实用程序时，该实用程序会立即应用您的更改。

GraphQL 实用程序的命令行参数

- **`--help, -h`** – 将 GraphQL 实用程序的帮助文本返回到控制台。

- **`--input-schema`** ***(schema text)*** – 要用作输入的 GraphQL 架构，带或不带指令。

- **`--input-schema-file`** ***(file URL)*** – 包含要用作输入的 GraphQL 架构的文件的 URL。

- **`--input-schema-changes-file`** ***(file URL)*** – 包含要对 GraphQL 架构所做的更改的文件的 URL。如果您多次对 Neptune 数据库运行该实用程序，并且还手动更改 GraphQL 源架构（可能添加自定义查询），则手动更改将丢失。为避免这种情况，请将您的更改放在更改文件中，然后使用此参数将其传入。

  更改文件采用以下 JSON 格式：

  ```
  [
    {
      "type": "(GraphQL type name)",
      "field": "(GraphQL field name)",
      "action": "(remove or add)",
      "value": "(value)"
    }
  ]
  ```

  有关更多信息，请参阅 Todo 示例。

- **`--input-graphdb-schema` *`(schema text)`*** – 您可以用文本形式表达 graphdb 架构以用作输入，而不是对 Neptune 数据库运行该实用程序。graphdb 架构具有如下所示的 JSON 格式：

```
{
  "nodeStructures": [
    { "label":"nodelabel1",
      "properties": [
        { "name":"name1", "type":"type1" }
      ]
    },
    { "label":"nodelabel2",
      "properties": [
          { "name":"name2", "type":"type1" }
      ]
    }
  ],
  "edgeStructures": [
    {
      "label":"label1",
      "directions": [
        { "from":"nodelabel1", "to":"nodelabel2", "relationship":"ONE-ONE|ONE-MANY|
MANY-MANY"  }
      ],
      "properties": [
        { "name":"name1", "type":"type1" }
      ]
    }
  ]
}
```

- **`--input-graphdb-schema-file` *`(file URL)`*** – 您可以将 graphdb 架构保存在文件中以用作输入，而不是对 Neptune 数据库运行该实用程序。有关 graphdb 架构文件的 JSON 格式示例，请参阅上面的 `--input-graphdb-schema`。

- **`--input-graphdb-schema-neptune-endpoint` *`(endpoint URL)`*** – Neptune 数据库端点，实用程序应从中提取 graphdb 架构。

- **`--output-schema-file` *`(file name)`*** – GraphQL 架构的输出文件名。如果未指定，则默认值为 `output.schema.graphql`，除非使用 `--create-update-aws-pipeline-name` 设置了管道名称，在这种情况下，默认文件名为 *`(pipline name)`*`.schema.graphql`。

- **`--output-source-schema-file` *`(file name)`*** – 带有指令的 GraphQL 架构的输出文件名。如果未指定，则默认值为 `output.source.schema.graphql`，除非使用 `--create-update-aws-pipeline-name` 设置了管道名称，在这种情况下，默认名称为 *`(pipeline name)`*`.source.schema.graphql`。

- **`--output-schema-no-mutations`** – 如果存在此参数，则该实用程序不会在 GraphQL API 中生成任何突变，只生成查询。

- **`--output-neptune-schema-file` *`(file name)`*** – 实用程序发现的 Neptune graphdb 架构的输出文件名。如果未指定，则默认值为 `output.graphdb.json`，除非使用 `--create-update-aws-pipeline-name` 设置了管道名称，在这种情况下，默认文件名为 *`(pipeline name)`*`.graphdb.json`。

- **`--output-js-resolver-file` *`(file name)`*** – 解析器代码副本的输出文件名。如果未指定，则默认值为 `output.resolver.graphql.js`，除非使用 `--create-update-aws-pipeline-name` 设置了管道名称，在这种情况下，文件名为 *`(pipeline name)`*`.resolver.graphql.js`。

  此文件压缩在上传到运行解析器的 Lambda 函数的代码包中。

- **`--output-resolver-query-sdk`** – 此参数指定该实用程序的 Lambda 函数应使用 Neptune 数据 SDK 来查询 Neptune，该 SDK 从 Neptune 引擎版本 1.2.1.0.R5（这是默认版本）开始推出。但是，如果该实用程序检测到较旧的 Neptune 引擎版本，则它建议改用 HTTPS Lambda 选项，您可以使用 `--output-resolver-query-https` 参数调用该选项。

- **`--output-resolver-query-https`** – 此参数指定该实用程序的 Lambda 函数应使用 Neptune HTTPS API 查询 Neptune。

- **`--create-update-aws-pipeline`**— 此参数会触发创建供 GraphQL API 使用的AWS资源，包括 GraphQL API 和运行 AppSync 解析器的 Lambda。

- **`--create-update-aws-pipeline-name`** ***`(pipeline name)`***— 此参数设置管道的名称，例如 Lambda `pipeline-name` 函数`pipeline-name`的 API AppSync 或函数。如果未指定名称，则 `--create-update-aws-pipeline` 使用 `Neptune` 数据库名称。
- **`--create-update-aws-pipeline-region`** ***`(AWS region)`*** – 此参数设置在其中创建 GraphQL API 的管道的 AWS 区域。如果未指定，则默认区域为 `us-east-1` 或 Neptune 数据库所在区域（从数据库端点提取）。
- **`--create-update-aws-pipeline-neptune-endpoint`** ***`(endpoint URL)`*** – 此参数设置 Lambda 函数用于查询数据库的 Neptune 数据库端点。如果未设置，则使用由 `--input-graphdb-schema-neptune-endpoint` 设置的端点。
- **`--remove-aws-pipeline-name`** ***`(pipeline name)`*** – 此参数移除使用 `--create-update-aws-pipeline` 创建的管道。要删除的资源列在名为 *`(pipeline name)`*`.resources.json` 的文件中。
- **`--output-aws-pipeline-cdk`**— 此参数会触发一个 CDK 文件的创建，该文件可用于为 GraphQL API 创建AWS资源，包括 GraphQL API 和运行解析器的 Lambda 函数。 AppSync
- **`--output-aws-pipeline-cdk-neptume-endpoint`** ***`(endpoint URL)`*** – 此参数设置 Lambda 函数用于查询 Neptune 数据库的 Neptune 数据库端点。如果未设置，则使用由 `--input-graphdb-schema-neptune-endpoint` 设置的端点。
- **`--output-aws-pipeline-cdk-name`** ***`(pipeline name)`***— 此参数设置要使用的 AppSync API 和 Lambda 管道名称函数的管道名称。如果未指定，则 `--create-update-aws-pipeline` 使用 Neptune 数据库名称。
- **`--output-aws-pipeline-cdk-region`** ***`(AWS region)`*** – 这会设置在其中创建 GraphQL API 的管道的 AWS 区域。如果未指定，则它默认为 `us-east-1` 或 Neptune 数据库所在的区域（从数据库端点提取）。
- **`--output-aws-pipeline-cdk-file`** ***`(file name)`*** – 这将设置 CDK 文件名。如果未设置，则默认为 *`(pipeline name)`*`-cdk.js`。

Neptune 服务错误

Amazon Neptune 有两组不同的错误：

- 仅针对 Neptune 数据库集群端点的图形引擎错误。
- 这些错误与用于通过 AWS SDK 和 AWS Command Line Interface (AWS CLI) 创建和修改 Neptune 资源的 API 关联。

主题

图形引擎错误消息和代码

在遇到时，Amazon Neptune 端点返回 Gremlin 和 SPARQL 的标准错误。

也可以从同一个端点返回特定于 Neptune 的错误。本节介绍 Neptune 错误消息、代码和建议的操作。

> **Note**
>
> 这些错误仅针对 Neptune 数据库集群端点。通过 AWS SDK 和 AWS CLI 创建和修改 Neptune 资源的 API 的常见错误集并不相同。有关这些错误的更多信息，请参阅the section called “API 错误”。

图形引擎错误格式

Neptune 错误消息返回相关的 HTTP 错误代码和 JSON 格式的响应。

```
HTTP/1.1 400 Bad Request
x-amzn-RequestId: LDM6CJP8RMQ1FHKSC1RBVJFPNVV4KQNSO5AEMF66Q9ASUAAJG
Content-Type: application/x-amz-json-1.0
Content-Length: 465
Date: Thu, 15 Mar 2017 23:56:23 GMT
```

```
{
    "requestId": "0dbcded3-a9a1-4a25-b419-828c46342e47",
    "code": "ReadOnlyViolationException",
    "detailedMessage": "The request is rejected because it violates some read-only
 restriction, such as a designation of a replica as read-only."
}
```

图形引擎查询错误

下表包含错误代码、消息和 HTTP 状态。

它还指示是否可以重试请求。一般而言，如果在新尝试中请求可能成功，则可以重试请求。

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`AccessDeniedException`	403	No	Authentication or authorization failure.
`BadRequestException`	400	No	The request could not be completed.
`BadRequestException`	400	No	Request size exceeds max allowed value of 157286400 bytes.
`CancelledByUserException`	500	Yes	The request processing was cancelled by an authorized client.
`ConcurrentModificationException`	500	Yes	The request processing did not succeed due to a modification conflict. The client should retry the request.
`ConstraintViolationException`	400	Yes	The query engine discovered, during the execution of the

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
			request, that the completion of some operation is impossible without violating some data integrity constraints, such as persistence of in- and out-vertices while adding an edge. Such conditions are typically observed if there are concurrent modifications to the graph, and are transient. The client should retry the request.
`FailureByQueryException`	500	Yes	Calling fail() caused request processing to fail.
`InternalFailureException`	500	Yes	The request processing has failed.
`InvalidNumericDataException`	400	No	Invalid use of numeric data which cannot be represented in 64-bit storage size.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`InvalidParameterException`	400	No	An invalid or out-of-range value was supplied for some input parameter or invalid syntax in a supplied RDF file.
`MalformedQueryException`	400	No	The request is rejected because it contains a query that is syntactically incorrect or does not pass additional validation.
`MemoryLimitExceededException`	500	Yes	The request processing did not succeed due to lack of memory, but can be retried when the server is less busy.
`MethodNotAllowedException`	405	No	The request is rejected because the chosen HTTP method is not supported by the used endpoint.
`MissingParameterException`	400	No	A required parameter for the specified action is not supplied.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`QueryLimitExceededException`	500	Yes	The request processing did not succeed due to the lack of a limited resource, but can be retried when the server is less busy.
`QueryLimitException`	400	No	Size of query exceeds system limit.
`QueryTooLargeException`	400	No	The request was rejected because its body is too large.
`ReadOnlyViolationException`	400	No	The request is rejected because it violates some read-only restriction, such as a designation of a replica as read-only.
`ThrottlingException`	500	Yes	Rate of requests exceeds the maximum throughput. OK to retry.
`TimeLimitExceededException`	500	Yes	The request processing timed out.
`TooManyRequestsException`	429	Yes	The rate of requests exceeds the maximum throughput. OK to retry.

Neptune Service Error Code	HTTP status	Ok to Retry?	Message
`UnsupportedOperationException`	400	No	The request uses a currently unsupported feature or construct.

IAM 身份验证错误

这些错误特定于已启用 IAM 身份验证的集群。

下表包含错误代码、消息和 HTTP 状态。

Neptune Service Error Code	HTTP status	Message
Incorrect IAM User/Policy	403	You do not have sufficient access to perform this action.
Incorrect or Missing Region	403	Credential should be scoped to a valid Region, not '*region*'.
Incorrect or Missing Service Name	403	Credential should be scoped to correct service: '`neptune-db `'.
Incorrect or Missing Host Header / Invalid Signature	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for details. Host header is missing or hostname is incorrect.
Missing `X-Amz-Security-Token`	403	'`x-amz-security-token `' is named as a `SignedHea`

Neptune Service Error Code	HTTP status	Message
		`der` , but it does not exist in the HTTP request
Missing Authorization Header	403	The request did not include the required authorization header, or it was malformed.
Missing Authentication Token	403	Missing Authentication Token.
Old Date	403	Signature expired: *`20181011T213907Z`* is now earlier than *`20181011T213915Z`* (*`20181011T214415Z`* - *`5 ##`*.)
Future Date	403	Signature not yet current: *`20500224T213559Z`* is still later than *`20181108T225925Z`* (*`20181108T225425Z`* + *`5 ##`*.)
Incorrect Date Format	403	Date must be in ISO-8601 'basic format'. Got '*`date`*'. See https://en.wikipedia.org/wiki/ISO_8601.
Unknown/Missing Access Key or Session Token	403	The security token included in the request is invalid.
Unknown/Missing Secret Key	403	The request signature we calculated does not match the signature you provided. Check your AWS Secret Access Key and signing method. Consult the service documentation for details. Host header is missing or hostname is incorrect.

Neptune Service Error Code	HTTP status	Message
TooManyRequestsException	429	The rate of requests exceeds the maximum throughput. OK to retry.

数据库集群管理 API 错误消息和代码

这些 Amazon Neptune 错误与用于借助 AWS SDK 和 AWS CLI 创建和修改 Neptune 资源的 API 关联。

下表包含错误代码、消息和 HTTP 状态。

Neptune Service Error Code	HTTP status	Message
AccessDeniedException	403	您没有足够的访问权限，无法执行该操作。
IncompleteSignature	400	请求签名不符合 AWS 标准。
InternalFailure	500	由于未知错误、异常或故障导致请求处理失败。
InvalidAction	400	所请求的操作无效。验证操作是否已正确键入。
InvalidClientTokenId	403	在我们的记录中没有所提供的 X.509 证书或 AWS 访问密钥 ID。
InvalidParameterCombination	400	不得共用的参数被一起使用。
InvalidParameterValue	400	为输入参数提供的值无效或超出范围。
InvalidQueryParameter	400	为输入参数提供的值无效或超出范围。

Neptune Service Error Code	HTTP status	Message
MalformedQueryString	400	查询字符串包含语法错误。
MissingAction	400	请求中遗漏了一个操作或必需参数。
MissingAuthenticationToken	403	请求中必须包含有效的（已注册的）AWS 访问密钥 ID 或 X.509 证书。
MissingParameter	400	未提供用于指定操作的必需参数。
OptInRequired	403	AWS 访问密钥 ID 需要订阅服务。
RequestExpired	400	请求到达服务的时间超过请求上的日期戳或请求到期日期（如针对预签名 URL）15 分钟，或者请求上的日期戳离到期还有 15 分钟以上。
ServiceUnavailable	503	由于服务器发生临时故障而导致请求失败。
ThrottlingException	500	由于请求限制而导致请求被拒绝。
ValidationError	400	输入未能满足 AWS 服务指定的约束。

Neptune 加载程序错误和源消息

Neptune 加载程序的 `status` 终端节点返回了以下消息。有关更多信息，请参阅获取状态 API。

下表包含加载程序源代码和描述。

Error or Feed Code	Description
LOAD_NOT_STARTED	加载已记录但未启动。
LOAD_IN_PROGRESS	表示正在进行加载并指定当前正在加载的文件数。 加载程序解析文件时，会创建一个或多个要并行加载的块。由于单个文件可以生成多个块，因此，此消息中包含的文件计数通常少于批量加载过程使用的线程数。
LOAD_COMPLETED	加载已完成且无任何错误或错误在可接受的阈值内。
LOAD_CANCELLED_BY_USER	用户已取消加载。
LOAD_CANCELLED_DUE_TO_ERRORS	由于错误，系统已取消加载。
LOAD_UNEXPECTED_ERROR	加载失败，返回意外错误。
LOAD_FAILED	加载因一个或多个错误而失败。
LOAD_S3_READ_ERROR	源由于间歇性或临时 Amazon S3 连接问题而失败。如果任何馈送收到此错误，整体加载状态将设置为 LOAD_FAILED。
LOAD_S3_ACCESS_DENIED_ERROR	已拒绝对 S3 存储桶的访问。如果任何馈送收到此错误，整体加载状态将设置为 LOAD_FAILED。
LOAD_COMMITTED_W_WRITE_CONFLICTS	加载数据已提交，但具有未解决的写入冲突。 加载程序将尝试解决单独事务中的写入冲突并随着加载进行更新馈送状态。如果最终源状态为 LOAD_COMMITTED_W_WRITE_CONFLICTS，则将尝试恢复加载，并且此操作可能成功且无写入冲突。写入冲突一般与错误输入数据无关，但数据重复可能增加写入冲突的可能性。

Error or Feed Code	Description
LOAD_DATA_DEADLOCK	Load was automatically rolled back due to deadlock.
LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETED	源由于文件在加载开始后已被删除或更新而失败。
LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED	加载请求未执行，因为其依赖关系检查失败。
LOAD_IN_QUEUE	加载请求已排队，正在等待执行。
LOAD_FAILED_INVALID_REQUEST	加载失败，因为请求无效（例如，指定的源/存储桶可能不存在，或者文件格式无效）。

Amazon Neptune 的引擎版本

Amazon Neptune 定期发布引擎更新。

您可以使用实例状态 API 或 Neptune 控制台确定当前已安装的引擎发行版。通过版本号，您可以得知自己运行的是原始主要版本、次要版本还是补丁版本。有关版本编号的更多信息，请参阅引擎版本号。

有关更新的更多一般信息，请参阅集群维护。

从引擎版本 1.3.0.0 开始，引擎版本的结构将如下表所示。次要版本号是经过评估以进行 `AutoMinorVersionUpgrade` 处理的版本号。

版本	产品版本	主要版本	次要版本	修补版本	Status	Released	使用寿命结束	升级到：
1.3.2.1	1	3	2	1	处于活动状态	2024-06-20	2025-11-30	不适用
1.3.2.0	1	3	2	0	处于活动状态	2024-06-10	2025-11-30	1.3.2.1
1.3.1.0	1	3	1	0	处于活动状态	2024-03-06	2025-11-30	1.3.2.1
1.3.0.0	1	3	0	0	处于活动状态	2023-11-15	2025-11-30	1.3.2.1

下表列出了自 1.0.1.0 以来的所有引擎版本以及版本信息。 end-of-life您可以参照此表中的日期来规划您的测试和升级周期。

版本	主要版本	次要版本	Status	Released	使用寿命结束	升级到：
1.2.1.1	1.2	1.1	处于活动状态	2024-03-11	2025-03-06	1.3.0.0
1.2.1.0	1.2	1.0	处于活动状态	2023-03-08	2025-03-06	1.3.0.0

版本	主要版本	次要版本	Status	Released	使用寿命结束	升级到：
1.2.0.2	1.2	0.2	处于活动状态	2022-11-16	2025-03-06	1.3.0.0
1.2.0.1	1.2	0.1	处于活动状态	2022-10-26	2025-03-06	1.3.0.0
1.2.0.0	1.2	0.0	处于活动状态	2022-07-21	2025-03-06	1.3.0.0
1.1.1.0	1.1	1.0	处于活动状态	2022-04-19	2024-10-31	1.2.1.0
1.1.0.0	1.1	0.0	处于活动状态	2021-11-19	2024-10-31	1.1.1.0
1.0.5.1	1.0	5.1	已弃用	2021-10-01	2023-01-30	1.1.0.0
1.0.5.0	1.0	5.0	已弃用	2021-07-27	2023-01-30	1.1.0.0
1.0.4.2	1.0	4.2	已弃用	2021-06-01	2023-01-30	1.1.0.0
1.0.4.1	1.0	4.1	已弃用	2020-12-08	2023-01-30	1.1.0.0
1.0.4.0	1.0	4.0	已弃用	2020-10-12	2023-01-30	1.1.0.0
1.0.3.0	1.0	3.0	已弃用	2020-08-03	2023-01-30	1.1.0.0
1.0.2.2	1.0	2.2	已弃用	2020-03-09	2022-07-29	1.0.3.0
1.0.2.1	1.0	2.1	已弃用	2019-11-22	2022-07-29	1.0.3.0
1.0.2.0	1.0	2.0	已弃用	2019-11-08	2020-05-19	1.0.3.0
1.0.1.2	1.0	1.2	已弃用	2019-10-15	—	—
1.0.1.1	1.0	1.1	已弃用	2019-08-13	—	—
1.0.1.0.*	1.0	1.0.*	已弃用	2019-07-02 及之前	—	—

主要发动机版本 end-of-life 规划

Neptune 引擎版本几乎总是在日历季度末达到使用寿命的尽头。只有在出现重大的安全或可用性问题时，才会出现例外。

当引擎版本达到其使用寿命结束时，您需要将 Neptune 数据库升级到较新的版本。

通常，Neptune 引擎版本将继续可用，如下所示：

- 次要引擎版本：次要引擎版本在发布后的至少 6 个月内保持可用。
- 主要引擎版本：主要引擎版本在发布后的至少 12 个月内保持可用。

在引擎版本终止使用前至少 3 个月，系统 AWS 会自动向与您的 AWS 账户关联的电子邮件地址发送电子邮件通知，并将相同的消息发布到您的 Healt AWS h Das hboard。这将使您有时间计划和准备升级。

当引擎版本达到其使用寿命结束时，您将无法再使用该版本创建新的集群或实例，自动缩放功能也无法使用该版本创建实例。

实际达到使用寿命结束的引擎版本将在维护时段期间自动升级。在引擎版本使用寿命结束前 3 个月发送给您的邮件将包含有关此自动更新将涉及的内容的详细信息，包括您将自动升级到的版本、对数据库集群的影响以及我们建议的操作。

Important

您有责任使数据库引擎版本保持最新。 AWS 敦促所有客户将其数据库升级到最新的引擎版本，以便从最新的安全、隐私和可用性保护措施中受益。如果您在弃用日期之后在不受支持的引擎或软件（“旧版引擎”）上运行数据库，则更有可能面临安全、隐私和运营风险，包括停机事件。

在任何引擎上操作您的数据库均受管理您使用 AWS 服务的协议的约束。传统引擎并非普遍可用。 AWS 不再为传统引擎提供支持，如果 AWS 确定旧版引擎对服务、其关联公司或任何第三方构成安全或责任风险或损害风险，则 AWS 可以随时限制对任何传统引擎的访问或使用。AWS您决定继续在旧版引擎中运行您的内容可能会导致您的内容不可用、损坏或无法恢复。在旧版引擎上运行的数据库受服务水平协议 (SLA) 例外情况的约束。

在旧版引擎上运行的数据库和相关软件包含漏洞、错误、缺陷和/或有害组件。因此，无论协议或服务条款中有任何相反之处，都 AWS 是“按原样”提供传统引擎。

亚马逊 Neptune Engine 版本 1.3.2.1 (2024-06-20)

截至2024年06月20日，引擎版本1.3.2.1已全面部署。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 引擎版本 1.3.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.3.0.0 之前的引擎版本升级到引擎版本 1.3.0.0 或更高版本，则必须使用参数组系列 `neptune1.3` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1` 或 `neptune1.2`，而这些参数组不适用于版本 1.3.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

此引擎版本中修复的缺陷

openCypher 修复

- 在包含具有和`LIMIT`作为参数的内部`WITH`子句的参数化查询的查询计划缓存功能中检测到一个错误。SKIPSKIP/LIMIT 值的参数化不正确，因此，后续执行具有不同参数值的相同缓存查询计划仍会返回与第一次执行相同的结果。此问题已得到修复。

```
# insert some nodes
UNWIND range(1, 10) as i CREATE (s {name: i}) RETURN s

# sample query
MATCH (p)
WITH p ORDER BY p.name SKIP $s LIMIT $l
RETURN p.name as res

# first time executing with {"s": 2, "l": 1}
{
  "results" : [ {
    "res" : 3
  } ]
}

# second time executing with {"s": 2, "l": 10}
# due to bug, produces
{
  "results" : [ {
```

```
    "res" : 3
  } ]
}
# with fix, produces correct results:
{
  "results" : [ {
    "res" : 3
  }, {
    "res" : 4
  }, {
    "res" : 5
  }, {
    "res" : 6
  }, {
    "res" : 7
  }, {
    "res" : 8
  }, {
    "res" : 9
  }, {
    "res" : 10
  } ]
}%
```

- 修复了当传递的参数不存在于数据库中InternalFailureException时，参数化突变查询会抛出一个错误。
- 修复了参数化的 Bolt 查询在查询资源清理期间遇到争用条件后卡住的错误。

1.3.2.1 中的变化是从 1.3.2.0 延续下来的

引擎版本 1.3.2.0 中延续的改进

常规改进

- 支持 TLS 版本 1.3 包括密码套件 TLS_AES_128_GCM_SHA256 和 TLS_AES_256_GCM_SHA384。TLS 1.3 是一个选项——TLS 1.2 仍然是最低限度。
- OpenCypher 对日期时间格式的扩展支持处于此版本的 lab_mode 中。我们鼓励您对其进行测试。

Gremlin 改进

- TinkerPop 3.7.x 升级

  - 提供了 Gremlin 语言的大幅扩展。
    - 处理字符串、列表和日期的新步骤。
    - 用于在步骤中指定基数的新语法。`mergeV()`
    - `union()`现在可以用作起始步骤。
    - 要了解有关 3.7.x 中变更的更多信息，请参阅TinkerPop 升级文档。
  - 在升级适用于 Java 的客户端 Gremlin 语言驱动程序时，请注意，序列化器类已经进行了一些重命名。如果已指定，则需要更新配置文件和代码中的包和类命名。
- `StrictTimeoutValidation`（仅当通过 labmode `StrictTimeoutValidation` 通过包含启用时`StrictTimeoutValidation=enabled`）：当`StrictTimeoutValidation`参数的值为时`enabled`，指定为请求选项或查询提示的每个查询的超时值不能超过参数组中全局设置的值。在这种情况下，Neptune 会扔一个。`InvalidParameterException`当值为时，可以在`/status`端点的响应中确认此设置`disabled`，而在 Neptune 版本 1.3.2.0 和 1.3.2.1 中，此参数的默认值为。`Disabled`

openCypher 改进

- 低延迟查询和吞吐量性能改进：低延迟 OpenCypher 查询的整体性能改进。新版本还提高了此类查询的吞吐量。当使用参数化查询时，这些改进更为显著。
- 对查询计划缓存的支持：将查询提交给 Neptune 时，查询字符串会被解析、优化并转换为查询计划，然后由引擎执行。应用程序通常以常见的查询模式为后盾，这些模式使用不同的值进行实例化。查询计划缓存可以通过缓存查询计划来减少总体延迟，从而避免对此类重复模式进行解析和优化。
- DISTINCT 聚合查询的性能改进。
- 涉及可空变量的联接的性能改进。
- 涉及不等于 id（节点/关系）谓词的查询的性能改进。
- 扩展了对日期时间功能的支持（仅`DatetimeMillisecond`通过包含`DatetimeMillisecond=enabled`在实验室模式下启用。有关更多信息，请参阅 Neptune OpenCypher 实现中的时间支持（Neptune Analytics 和 Neptune Database 1.3.2.0 及更高版本）。

引擎版本 1.3.2.0 中延续的缺陷修复

常规改进

- 更新了验证对 Graphlytics 存储桶的访问权限时出现的 Neptuneml 错误消息。

Gremlin 修复

- 修复了 DFE 查询翻译中缺少标签信息的问题，适用于非路径贡献步骤包含标签的情况。例如：

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'marko').
  has("name", TextP.regex("mark.*")).as("p1").
  not(out().has("name", P.within("peter"))).
  out().as('p2').
  dedup('p1', 'p2')
```

- 修复了 DFE 查询转换中的一个`NullPointerException`错误，该错误发生在两个 DFE 片段中执行查询，并且第一个片段被优化为无法满足的节点时。例如：

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'doesNotExists').
  has("name", TextP.regex("mark.*")).
  inject(1).
  V().
  out().
  has('name', 'vadas')
```

- 修复了当查询中包含 by () 调制器且其输入为 Map `InternalFailureException` ValueTraversal 时，Neptune 可能会抛出一个错误。例如：

```
g.V().
  hasLabel("person").
  project("age", "name").by("age").by("name").
  order().by("age")
```

openCypher 修复

- 改进了 UNWIND 操作（例如将值列表扩展为单个值），以帮助防止出现内存不足 (OOM) 的情况。例如：

```
MATCH (n)-->(m)
WITH collect(m) AS list
UNWIND list AS m
```

```
RETURN m, list
```

- 修复了在多个 MERGE 操作中通过 UNWIND 注入 id 时的自定义 ID 优化问题。例如：

```
UNWIND [{nid: 'nid1', mid: 'mid1'}, {nid: 'nid2', mid: 'mid2'}] as ids
MERGE (n:N {`~id`: ids.nid})
MERGE (m:M {`~id`: ids.mid})
```

- 修复了在规划具有属性访问权限的复杂查询和具有双向关系的多跳时内存爆炸的问题。例如：

```
MATCH (person1:person)-[:likes]->(res)-[:partOf]->(group)-[:knows]-(:entity {name:
 'foo'}),
      (person1)-[:knows]->(person2)-[:likes]-(res2), (comment)-[:presentIn]->(:Group
 {name: 'barGroup'}),
      (person1)-[:commented]->(comment2:comment)-[:partOf]->(post:Post), (comment2)-
[:presentIn]->(:Group {name: 'fooGroup'}),
      (comment)-[:contains]->(info:Details)-[:CommentType]->(:CommentType {name:
 'Positive'}),
      (comment2)-[:contains]->(info2:Details)-[:CommentType]->(:CommentType {name:
 'Positive'})
WHERE datetime('2020-01-01T00:00') <= person1.addedAfter <=
 datetime('2023-01-01T23:59') AND comment.approvedBy = comment2.approvedBy
MATCH (comment)-[:contains]->(info3:Details)-[:CommentType]->(:CommentType {name:
 'Neutral'})
RETURN person1, group.name, info1.value,  post.ranking, info3.value
```

- 修复了按变量分组为 null 的聚合查询。例如：

```
MATCH (n)
RETURN null AS group, sum(n.num) AS result
```

SPARQL 修复

- 修复了 SPARQL 解析器以缩短大型查询的解析时间，例如包含许多三元组和大标记的 INSERT DATA。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.3.2.1 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`

- 支持的 Gremlin 最新版本：`3.7.1`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.3.2.1 的升级路径

您可以从引擎版本 1.2.0.0 或更高版本升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.2.1 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.2.1 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级， allow-major-version-upgrade 参数是必需的。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何疑问或疑虑，AWS 可通过社区论坛和AWS 高级支持与支持团队联系。

亚马逊 Neptune Engine 版本 1.3.2.0 (2024-06-10)

自 2024 年 6 月 10 日起，引擎版本 1.3.2.0 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 引擎版本 1.3.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.3.0.0 之前的引擎版本升级到引擎版本 1.3.0.0 或更高版本，则必须使用参数组系列 `neptune1.3` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1` 或 `neptune1.2`，而这些参数组不适用于版本 1.3.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

> **Warning**
>
> 当在内部`WITH`子句中使用`skip`或`limit`并进行参数化时，我们在查询计划缓存中检测到一个问题。为防止出现此问题，请在提交包含参数化跳过和/或限制子句的查询`QUERY:PLANCACHE "disabled"`时添加查询提示。或者，您可以将值硬编码到查询中。有关更多信息，请参阅 缓解查询计划缓存问题。

此引擎版本中的改进

常规改进

- 支持 TLS 版本 1.3 包括密码套件 TLS_AES_128_GCM_SHA256 和 TLS_AES_256_GCM_SHA384。TLS 1.3 是一个选项——TLS 1.2 仍然是最低限度。

Gremlin 改进

- TinkerPop 3.7.x 升级
  - 提供了 Gremlin 语言的大幅扩展。
    - 处理字符串、列表和日期的新步骤。
    - 用于在步骤中指定基数的新语法。`mergeV()`
    - `union()`现在可以用作起始步骤。
    - 要了解有关 3.7.x 中变更的更多信息，请参阅TinkerPop 升级文档。

  - 在升级适用于 Java 的客户端 Gremlin 语言驱动程序时，请注意，序列化器类已经进行了一些重命名。如果已指定，则需要更新配置文件和代码中的包和类命名。
- `StrictTimeoutValidation`（仅当通过 labmode `StrictTimeoutValidation` 通过包含启用时`StrictTimeoutValidation=enabled`）：当`StrictTimeoutValidation`参数的值为时`enabled`，指定为请求选项或查询提示的每个查询的超时值不能超过参数组中全局设置的值。在这种情况下，Neptune 会扔一个。`InvalidParameterException`当值为时，可以在`/status`端点的响应中确认此设置`disabled`，而在 Neptune 版本 1.3.2.0 中，此参数的默认值为。`Disabled`

openCypher 改进

- 低延迟查询和吞吐量性能改进：低延迟 OpenCypher 查询的整体性能改进。新版本还提高了此类查询的吞吐量。当使用参数化查询时，这些改进更为显著。
- 对查询计划缓存的支持：将查询提交给 Neptune 时，查询字符串会被解析、优化并转换为查询计划，然后由引擎执行。应用程序通常以常见的查询模式为后盾，这些模式使用不同的值进行实例化。查询计划缓存可以通过缓存查询计划来减少总体延迟，从而避免对此类重复模式进行解析和优化。
- DISTINCT 聚合查询的性能改进。
- 涉及可空变量的联接的性能改进。
- 涉及不等于 id（节点/关系）谓词的查询的性能改进。
- 扩展了对日期时间功能的支持（仅`DatetimeMillisecond`通过包含`DatetimeMillisecond=enabled`在实验室模式下启用。有关更多信息，请参阅 Neptune OpenCypher 实现中的时间支持（Neptune Analytics 和 Neptune Database 1.3.2.0 及更高版本）。

此引擎版本中修复的缺陷

常规改进

- 更新了验证对 Graphlytics 存储桶的访问权限时出现的 Neptuneml 错误消息。

Gremlin 修复

- 修复了 DFE 查询翻译中缺少标签信息的问题，适用于非路径贡献步骤包含标签的情况。例如：

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'marko').
  has("name", TextP.regex("mark.*")).as("p1").
```

```
  not(out().has("name", P.within("peter"))).
  out().as('p2').
  dedup('p1', 'p2')
```

- 修复了 DFE 查询转换中的一个NullPointerException错误，该错误发生在两个 DFE 片段中执行查询，并且第一个片段被优化为无法满足的节点时。例如：

```
g.withSideEffect('Neptune#useDFE', true).
  V().
  has('name', 'doesNotExists').
  has("name", TextP.regex("mark.*")).
  inject(1).
  V().
  out().
  has('name', 'vadas')
```

- 修复了当查询中包含 by () 调制器且其输入为 Map InternalFailureException ValueTraversal 时，Neptune 可能会抛出一个错误。例如：

```
g.V().
  hasLabel("person").
  project("age", "name").by("age").by("name").
  order().by("age")
```

openCypher 修复

- 改进了 UNWIND 操作（例如将值列表扩展为单个值），以帮助防止出现内存不足 (OOM) 的情况。例如：

```
MATCH (n)-->(m)
WITH collect(m) AS list
UNWIND list AS m
RETURN m, list
```

- 修复了在多个 MERGE 操作中通过 UNWIND 注入 id 时的自定义 ID 优化问题。例如：

```
UNWIND [{nid: 'nid1', mid: 'mid1'}, {nid: 'nid2', mid: 'mid2'}] as ids
MERGE (n:N {`~id`: ids.nid})
MERGE (m:M {`~id`: ids.mid})
```

- 修复了在规划具有属性访问权限的复杂查询和具有双向关系的多跳时内存爆炸的问题。例如：

```
MATCH (person1:person)-[:likes]->(res)-[:partOf]->(group)-[:knows]-(:entity {name:
 'foo'}),
      (person1)-[:knows]->(person2)-[:likes]-(res2), (comment)-[:presentIn]->(:Group
 {name: 'barGroup'}),
      (person1)-[:commented]->(comment2:comment)-[:partOf]->(post:Post), (comment2)-
[:presentIn]->(:Group {name: 'fooGroup'}),
      (comment)-[:contains]->(info:Details)-[:CommentType]->(:CommentType {name:
 'Positive'}),
      (comment2)-[:contains]->(info2:Details)-[:CommentType]->(:CommentType {name:
 'Positive'})
WHERE datetime('2020-01-01T00:00') <= person1.addedAfter <=
 datetime('2023-01-01T23:59') AND comment.approvedBy = comment2.approvedBy
MATCH (comment)-[:contains]->(info3:Details)-[:CommentType]->(:CommentType {name:
 'Neutral'})
RETURN person1, group.name, info1.value,  post.ranking, info3.value
```

- 修复了按变量分组为 null 的聚合查询。例如：

```
MATCH (n)
RETURN null AS group, sum(n.num) AS result
```

SPARQL 修复

- 修复了 SPARQL 解析器以缩短大型查询的解析时间，例如包含许多三元组和大标记的 INSERT DATA。

缓解查询计划缓存问题

对于版本 1.3.2.0，当在内部`WITH`子句中使用`skip`或`limit`并进行参数化时，我们检测到查询计划缓存中存在问题。例如：

```
MATCH (n:Person)
WHERE n.age > $age
WITH n skip $skip LIMIT $limit
RETURN n.name, n.age

parameters={"age": 21, "skip": 2, "limit": 3}
```

在这种情况下，第一个计划中的 skip 和 limit 的参数值也将应用于后续查询，从而导致意想不到的结果。

缓解方法

为防止出现此问题，请在提交包含参数化跳过和/或限制子句的查询QUERY:PLANCACHE "disabled"时添加查询提示。或者，您可以将值硬编码到查询中。

选项 1：使用查询提示禁用计划缓存：

```
Using QUERY:PLANCACHE "disabled"
MATCH (n:Person) WHERE n.age > $age
WITH n skip $skip LIMIT $limit
RETURN n.name, n.age

parameters={"age": 21, "skip": 2, "limit": 3}
```

选项 2：使用硬编码值进行跳过和限制：

```
MATCH (n:Person)
WHERE n.age > $age
WITH n skip 2 LIMIT 3
RETURN n.name, n.age

parameters={"age": 21}
```

此版本支持的查询语言版本

在将数据库集群升级到版本 1.3.2.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：3.6.2
- 支持的 Gremlin 最新版本：3.7.1
- openCypher 版本：Neptune-9.0.20190305-1.0
- SPARQL 版本：1.1

引擎版本 1.3.2.0 的升级路径

您可以从引擎版本 1.2.0.0 或更高版本升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.2.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.2.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级， allow-major-version-upgrade 参数是必需的。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何疑问或疑虑， AWS 可通过社区论坛和AWS 高级支持与支持团队联系。

亚马逊 Neptune Engine 版本 1.3.1.0 (2024-03-06)

自 2024 年 3 月 6 日起，引擎版本 1.3.1.0 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 引擎版本 1.3.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.3.0.0 之前的引擎版本升级到引擎版本 1.3.0.0 或更高版本，则必须使用参数组系列 `neptune1.3` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1` 或 `neptune1.2`，而这些参数组不适用于版本 1.3.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

此引擎版本中的改进

常规改进

- Neptune 改进了配置文件/解释中显示的警告。
- 从 TLS 协商期间使用的默认命名组中删除了过时的 NIST EC 曲线。移除的曲线是 sect409k1、sect409r1 和 sect571k1。

Gremlin 改进

- 改进了 DFE 统计数据计算，以避免无服务器实例的 NCU 过高。
- Gremlin 在 INSIDE 中的性能有所改进。

此引擎版本中修复的缺陷

Gremlin 修复

- 对 Gremlin DFE 查询计划进行了其他改进。
- 修复了带有可选遍历的 Gremlin 查询的错误，例如，以 `g.v () .hasLabel ('person') .group () .by (id ()) .by (__.in ('friend') .id () .id () .id () .fold ()) `形式进行的查询，其中没有好友边缘的人不会被分组。
- 修复了在by调制器中包含合并步骤的 Gremlin 查询在使用 DFE 引擎执行时会导致返回错误的错误。
- 修复了在 Gremlin 会话中运行的只读查询在连接到只读副本时无法运行的错误。
- 错误修复了 Gremlin 的初始 websocket 连接请求成功后，审核日志中不存在 IAM ARN 的错误。
- 合并步骤，使用 DFE 确定步数覆盖范围。
- 整个 DFE 计划的特征集优化。

openCypher 修复

- 修复了 OpenCypher SET 子句中的错误，允许对非变量表达式进行设置（即：match (n: Test) 集（n.prop = 2 然后 n 结束的情况）.prop = 3 返回 n.prop。
- 修复了涉及聚合和排序依据的 OpenCypher 查询失败的错误。
- 改进了包含静态地图的大型列表的 UNWIND。
- 错误修复 OpenCypher MERGE 查询使用具有重复值的自定义 ID。

SPARQL 修复

- 修复了有关可选模式中变量作用域的 SPARQL 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.3.1.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.5`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.3.1.0 的升级路径

您可以从引擎版本 1.2.0.0 或更高版本升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.1.0 \
    --allow-major-version-upgrade \
```

```
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.1.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级， allow-major-version-upgrade 参数是必需的。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何疑问或疑虑， AWS 可通过社区论坛和AWS 高级支持与支持团队联系。

Amazon Neptune 引擎版本 1.3.0.0（2023 年 11 月 15 日）

截至 2023 年 11 月 15 日，引擎版本 1.3.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 引擎版本 1.3.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.3.0.0 之前的引擎版本升级到引擎版本 1.3.0.0 或更高版本，则必须使用参数组系列 `neptune1.3` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1` 或 `neptune1.2`，而这些参数组不适用于版本 1.3.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

此引擎版本中的新增功能

- 发布了 Neptune 数据 API。

Amazon Neptune 数据 API 为 Neptune 的 40 多种数据操作提供 SDK 支持，包括数据加载、查询执行、数据查询和机器学习。它支持所有三种 Neptune 查询语言（Gremlin、openCypher 和 SPARQL），并可用于所有 SDK 语言中。它自动签署 API 请求，极大地简化了将 Neptune 集成到应用程序中的过程。

- 增加了对OpenSearch无服务器与 Neptune 集成的支持。

此引擎版本中的改进

对 Neptune 引擎更新的改进

Neptune 改变了发布引擎更新的方式，因此，您可以更好地控制更新过程。Neptune 现在不发布针对不间断更改的补丁，而是发布次要版本，这些版本可以使用 `AutoMinorVersionUpgrade` 实例字段进行控制，并且您可以通过订阅 `RDS-EVENT-0156` 事件来接收有关该版本的通知。

有关这些更改的更多信息，请参阅维护 Amazon Neptune 数据库集群。

传输中加密改进

Neptune 不再支持以下密码套件：

- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`

在 TLS 1.2 中，Neptune 仅支持以下强密码套件：

- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
- `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`

Gremlin 改进

- 在 DFE 引擎中添加了对以下 Gremlin 步骤的支持：
  - `FoldStep`
  - `GroupStep`

  - GroupCountStep
  - TraversalMapStep
  - UnfoldStep
  - LabelStep
  - PropertyKeyStep
  - PropertyValueStep
  - AndStep
  - OrStep
  - ConstantStep
  - CountGlobalStep
- 优化了 Gremlin DFE 查询计划，以免在使用 by() 调制时进行全顶点扫描。
- 极大地提高了低基数和低延迟查询的性能。
- 添加了对 TinkerPop 0r过滤器谓词的 DFE 支持。
- 改进了 DFE 对相同键上筛选器的遍历支持，适用于如下查询：

```
g.withSideEffect("Neptune#useDFE", true)
 .V()
 .has('name', 'marko')
 .and(
   or(
     has('name', eq("marko")),
     has('name', eq("vardas"))
   )
 )
```

- 改进了 fail() 步骤的错误处理。

openCypher 改进

- 极大地提高了低基数和低延迟查询的性能。
- 提高了在查询包含许多节点类型时的查询计划性能。
- 减少了所有 VLP 查询的延迟。
- 通过删除单节点模式查询的冗余管道联接来提高性能。
- 提高了包含带循环的多跳模式的查询的性能，如下所示：

```
MATCH (n)-->()-->()-->(m)
RETURN n m
```

SPARQL 改进

- 引入了一个新的 SPARQL 运算符：`PipelineHashIndexJoin`。
- 提高了 SPARQL 查询的 URI 验证性能。
- 通过批量解析字典术语，提高了 SPARQL 全文搜索查询的性能。

在此引擎版本中修复的缺陷

Gremlin 修复

- 修复了一个 Gremlin 错误，即在检查 Gremlin 查询状态端点以查找子遍历中具有谓词的查询是否具有未在 DFE 引擎中原生处理的步骤时，会发生事务泄漏。
- 修复了一个 Gremlin 错误，即 `valueMap()` 无法在 DFE 引擎的 `by()` 遍历下进行优化。
- 修复了一个 Gremlin 错误，即附加到 `UnionStep` 的步骤标签无法分别传播到其子遍历的最后一个路径元素。
- 修复了 Gremlin 错误，其中查询会因为包含太多 TinkerPop 步骤而失败，然后无法清理。
- 修复了一个 Gremlin 错误，即 `NullPointerException` 会引发 `mergeV` 和 `mergeE` 步骤。
- 修复了一个 Gremlin 错误，即当其中一些字符串输出包含空格字符时，`order()` 无法对字符串输出正确排序。
- 修复了在 DFE 引擎中处理 `valueMap` 步骤时出现的 Gremlin 正确性问题。
- 修复了在键遍历中嵌套 `GroupStep` 或 `GroupCountStep` 时出现的 Gremlin 正确性问题。

openCypher 修复

- 修复了一个 openCypher 错误，该错误涉及有关空字符的错误处理。
- 修复了 openCypher Bolt 事务处理中的一个错误。

SPARQL 修复

- 修复了一个 SPARQL 错误，即递归函数中的值无法正确解析。

- 修复了在通过 VALUES 子句注入大量值时可能导致性能下降的 SPARQL 错误。
- 修复了一个 SPARQL 错误，即在使用带有语言标签的文本上调用 REGEX 运算符时从不会成功。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.3.0.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.3.0.0 的升级路径

您可以从引擎版本 1.2.0.0 或更高版本升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.3.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.3.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级， allow-major-version-upgrade 参数是必需的。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何疑问或疑虑， AWS 可通过社区论坛和AWS 高级支持与支持团队联系。

亚马逊 Neptune 引擎版本 1.2.1.1 (2024-03-11)

自 2024 年 3 月 11 日起，引擎版本 1.2.1.1 已全面部署。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且`UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果该`UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可以帮助您探索其他降低该指标的策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径

可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

一般改进

Neptune 改进了配置文件/解释中显示的警告。

Gremlin 改进

- 改进了 DFE 统计数据计算，以避免无服务器实例的 NCU 过高。
- Gremlin 在 INSIDE 中的性能有所改进。

在此引擎版本中修复的缺陷

Gremlin 修复

- 修复了排序 Gremlin DFE 引擎查询计划时出现的错误。
- 修复了最初报告为 Gremlin out-of-memory 错误时的错误。 InternalFailureException
- 错误修复了初始 websocket 连接请求成功后，审核日志中不存在 IAM ARN 的错误。
- 修复了启用 TinkerPop 会话的 Gremlin 查询的错误，即便所有查询均为只读并连接到阅读器实例，但会话中的查询也会失败。

openCypher 修复

- 修复了 OpenCypher SET 子句中的错误，允许在非变量表达式上进行设置（即：match (n: Test) 集（n.prop = 2 然后 n 结束的情况）.prop = 3 返回 n.prop
- 修复了涉及聚合和排序依据的 OpenCypher 查询失败的错误。
- 改进了包含静态地图的大型列表的 UNWIND。
- 错误修复 OpenCypher MERGE 查询使用具有重复值的自定义 ID。

SPARQL 修复

- 修复了 SPARQL DFE 查询计划器中的错误。

- 修复了 SPARQL 与 BIND 和可选关键字一起使用时的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.1 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.1.1 的升级路径

您可以从引擎版本 1.2.0.0 或更高版本升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.1 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.1 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级， allow-major-version-upgrade 参数是必需的。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
--db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
--db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
> ```

```
    proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何疑问或疑虑， AWS 可通过社区论坛和AWS 高级支持与支持团队联系。

Amazon Neptune 引擎版本 1.2.1.0（2023 年 3 月 8 日）

截至 2023 年 3 月 8 日，引擎版本 1.2.1.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此版本的后续补丁版本

- 版本：1.2.1.0.R2（2023 年 5 月 2 日）
- 版本：1.2.1.0.R3（2023 年 6 月 13 日）
- 版本：1.2.1.0.R4（2023 年 8 月 10 日）
- 版本：1.2.1.0.R5（2023 年 9 月 2 日）
- 版本：1.2.1.0.R6（2023 年 9 月 12 日）
- 版本：1.2.1.0.R7（2023 年 10 月 6 日）

此引擎版本中的新增功能

- 增加了对 TinkerPop 3.6.2 的支持，它增加了许多新的 Gremlin 特征，例如新的 `mergeV()`、`mergeE()`、`element()` 和 `fail()` 步骤。`mergeV()` 和 `mergeE()` 步骤特别值得注意，因为它们为执行类似更新插入的操作提供了期待已久的声明性选项，这应该可以大大简化现有的代码模式并使 Gremlin 更易于阅读。3.6.x 版本还添加了正则表达式谓词，对采用 `Map` 的 `property()` 步骤进行了新的重载，并对 `by()` 调制行为进行了重大修订，此行为在使用它的所有步骤中都更加一致。

  有关版本 3.6 中的更改以及升级时需要考虑的事项的信息，请参阅 TinkerPop 更改日志和升级页面。

  如果您使用 `fold().coalesce(unfold(), <mutate>)` 进行条件插入，我们建议您迁移到此处和此处描述的新 `mergeV/E()` 语法。Neptune 对 `Merge` 使用的锁定模式比对 `Coalesce` 使用的锁定模式更窄，这可以减少并发修改异常 (CME)。

  有关此 TinkerPop 版本中提供的新特征的更多信息，请参阅 Stephen Mallette 的博客探索 Amazon Neptune 中 Apache TinkerPop 3.6.x 的新特征。
- 增加了对由第三代 Intel Xeon 可扩展处理器提供支持的 R6i 实例类型的支持。它们非常适合内存密集型工作负载，与同类 R5 实例类型相比，计算/性价比最多可提高 15%，每个 vCPU 的内存带宽最多可提高 20%。
- 为属性图和 RDF 图形添加了图形摘要 API 端点，可让您快速获得有关图形的摘要报告。

  对于属性 (PG) 图，图形摘要 API 会提供节点和边缘标签以及属性键的只读列表，以及节点、边缘和属性的计数。对于 RDF 图形，它提供了类和谓词键的列表，以及四元组、主语和谓词的计数。

  以下更改与新的图形摘要 API 一起推出：

  - 添加了新的 GetGraphSummary 数据面板操作。
  - 添加了一个新的 `rdf/statistics` 端点来替换 `sparql/statistics` 端点（该端点现已弃用）。
  - 将统计数据状态响应中 `summary` 字段的名称更改为 `signatureInfo`，以免将其与图形摘要信息混淆。之前的引擎版本将继续在 JSON 响应中使用 `summary`。
  - 将统计数据状态响应中 `date` 字段的精度从分钟更改为毫秒。以前的格式是 `2020-05-07T23:13Z`（分钟精度），而新的格式是 `2023-01-24T00:47:43.319Z`（毫秒精度）。两者都符合 ISO 8601 标准，但此更改可能会破坏现有代码，具体取决于日期的解析方式。
  - 在 Workbench 中添加了新的 `%statistics` 行魔术命令，可以让您检索 DFE 引擎的统计数据。
  - 在 Workbench 中添加了新的 `%summary` 行魔术命令，可以让您检索图形摘要信息。
- 为记录执行时间超过指定阈值的查询添加了慢速查询日志记录。使用两个新的动态参数，即 neptune_enable_slow_query_log 和 neptune_slow_query_log_threshold，可以启用和控制慢速查询日志记录。
- 增加了对两个动态参数的支持，即新的集群参数 neptune_enable_slow_query_log 和 neptune_slow_query_log_threshold。当您对动态参数进行更改时，它会立即生效，无需重启任何实例。
- 添加了 Neptune 特定的 removeKeyFromMap() 函数，该函数从映射中删除指定的键并返回生成的新映射。

此引擎版本中的改进

- 将 Gremlin DFE 支持扩展到具有本地范围的 `limit` 步骤。
- 在 DFE 引擎中增加了对 Gremlin `DedupGlobalStep` 的 `by()` 调制支持。
- 增加了对 Gremlin `SelectStep` 和 `SelectOneStep` 的 DFE 支持。
- 各种 Gremlin 运算符的性能改进和正确性修复，包括 `repeat`、`coalesce`、`store` 和 `aggregate`。
- 提高了涉及 `MERGE` 和 `OPTIONAL MATCH` 的 openCypher 查询的性能。
- 改进了 openCypher 查询的性能，这些查询涉及文本值映射列表的 `UNWIND`。
- 改进了对 `id` 具有 `IN` 筛选条件的 openCypher 查询的性能。例如：

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- 增加了使用 BASE 语句为 SPARQL 查询指定基本 IRI 的功能（请参阅用于查询和更新的默认基本 IRI）。

- 缩短了 Gremlin 和 openCypher 仅限边缘批量加载的加载处理等待时间。
- 使批量加载在 Neptune 重启时异步恢复，以避免在恢复尝试失败之前因 Amazon S3 连接问题而导致的漫长等待时间。
- 改进了对 SPARQL DESCRIBE 查询的处理，这些查询将 describeMode 查询提示设置为 `"CBD"`（简洁的界限描述），并且涉及大量空白节点。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 `"null"`，而不是 null 值。
- 修复了列表推导中的 openCypher 错误，该错误会生成 null 值而不是为列表元素提供的值。
- 修复了字节值未正确序列化的 openCypher 错误。
- 修复了 `UnionStep` 中的一个 Gremlin 错误，该错误发生在输入是边缘遍历到子遍历内的顶点时。
- 修复了一个 Gremlin 错误，该错误导致与 `UnionStep` 关联的步骤标签无法正确传播到每个子遍历的最后一步。
- 修复了 `dedup` 步骤的 Gremlin 错误，其中 `repeat` 步骤后面有标签，附加到 `dedup` 步骤的标签无法在查询中进一步使用。
- 修复了一个 Gremlin 错误，即转换 `union` 步骤中的 `repeat` 步骤失败并出现内部错误。
- 修复了 Gremlin 正确性问题，此类问题通过回退到 Tinkerpop 影响将 `limit` 作为非并集步骤的子遍历的 DFE 查询。以下形式的查询会受到影响：

```
g.withSideEffect('Neptune#useDFE', true).V().as("a").select("a").by(out().limit(1))
```

- 修复了一个 SPARQL 错误，即 `SPARQL GRAPH` 模式不会考虑 `FROM NAMED` 子句提供的数据集。
- 修复了一个 SPARQL 错误，即包含一些 `FROM` 和/或 `FROM NAMED` 子句的 SPARQL `DESCRIBE` 并不总是正确使用默认图形中的数据，有时还会引发异常。请参阅SPARQL DESCRIBE 相对于默认图形的行为。
- 修复了一个 SPARQL 错误，以便在拒绝 null 字符时返回正确的异常消息。
- 修复了一个 SPARQL explain 错误，该错误会影响包含 PipelinedHashIndexJoin 运算符的计划。
- 修复了在提交返回常量值的查询时引发内部错误的错误。
- 修复了死锁探测器逻辑偶尔会导致引擎无响应的问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.1`
- SPARQL 版本：`1.1`

引擎版本 1.2.1.0 的升级路径

您可以从任何高于或等于 1.1.0.0 的先前 Neptune 引擎版本手动升级到此版本。

> **Note**
>
> 从引擎版本 1.2.0.0 开始，您在 `1.2.0.0` 之前的引擎版本中使用的所有自定义参数组和自定义集群参数组现在都必须使用参数组系列 `neptune1.2` 重新创建。之前的版本使用参数组系列 `neptune1`，而这些参数组将不适用于 `1.2.0.0` 以后的版本。请参阅Amazon Neptune 参数组了解更多信息。

您不会自动升级到此主要发行版本。

升级到此版本

Amazon Neptune 1.2.1.0 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
> ```

```
    proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R7（2023 年 10 月 6 日）

截至 2023 年 10 月 6 日，引擎版本 1.2.1.0.R7 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

在此引擎版本中修复的缺陷

- 修复了在某些情况下失败的事务没有正确关闭的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R7 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

升级到此版本

Amazon Neptune 1.2.1.0.R7 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R6（2023 年 9 月 12 日）

截至 2023 年 9 月 12 日，引擎版本 1.2.1.0.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的新增功能

- 发布了 Neptune 数据 API。

  Amazon Neptune 数据 API 为加载数据、运行查询、获取有关数据的信息以及运行机器学习操作提供 SDK 支持。它支持 Neptune 中的 Gremlin 和 openCypher 查询语言，并可用于所有 SDK 语言中。它自动签署 API 请求，极大地简化了将 Neptune 集成到应用程序中的过程。

在此引擎版本中修复的缺陷

- 修复了启用慢速查询日志后，在高负载下可能导致 CPU 峰值的错误。
- 修复了一个 Gremlin 错误，即添加边缘及其属性，然后添加 `inV()` 或 `outV()` 会导致 `InternalFailureException`。
- 修复了在某些情况下导致批量加载程序性能下降的 IAM 角色链中的几个问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R6 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

升级到此版本

Amazon Neptune 1.2.1.0.R6 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R5（2023 年 9 月 2 日）

截至 2023 年 9 月 2 日，引擎版本 1.2.1.0.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的新增功能

- 发布了 Neptune 数据 API。

  Amazon Neptune 数据 API 为加载数据、运行查询、获取有关数据的信息以及运行机器学习操作提供 SDK 支持。它支持 Neptune 中的 Gremlin 和 openCypher 查询语言，并可用于所有 SDK 语言中。它自动签署 API 请求，极大地简化了将 Neptune 集成到应用程序中的过程。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即添加边缘及其属性，然后添加 `inV()` 或 `outV()` 会导致 `InternalFailureException`。
- 修复了在某些情况下导致批量加载程序性能下降的 IAM 角色链中的几个问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R5 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

升级到此版本

Amazon Neptune 1.2.1.0.R5 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R4（2023 年 8 月 10 日）

截至 2023 年 8 月 10 日，引擎版本 1.2.1.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

在某些情况下，此引擎版本中引入的更改可能会导致您观察到批量加载性能下降。因此，到此版本的升级已临时暂停，直到此问题得到解决。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 添加了对于 Gremlin 的 GraphSON-1.0 支持。要使用 GraphSON-1.0，请使用以下值传递 `Accept header`：

```
application/vnd.gremlin-v1.0+json;types=false
```

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即在检查 Gremlin 查询状态端点以查找子遍历中具有谓词的查询是否具有未原生处理的步骤时，会发生事务泄漏。
- 修复了 Bolt 事务处理中的一个 openCypher 错误。
- 修复了存储层上可能导致崩溃的并发问题。
- 修复了慢速查询日志中的一个错误，以确保它们在禁用时处于非活动状态。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R4 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.5`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.1.0.R4 的升级路径

升级到此版本

Amazon Neptune 1.2.1.0.R4 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
```

```
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R3（2023 年 6 月 13 日）

截至 2023 年 6 月 13 日，引擎版本 1.2.1.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

在某些情况下，此引擎版本中引入的更改可能会导致您观察到批量加载性能下降。因此，到此版本的升级已临时暂停，直到此问题得到解决。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的新增功能

- 增加了对使用 IAM 角色链进行跨账户批量加载的支持。

此引擎版本中的改进

- 改进了 Gremlin 的 `fail()` 步骤，以区分它产生的异常与泛型 `InternalFailureException`，并确保向其提供的任何用户提供的消息都能传播回调用方。
- 改进了 Gremlin 查询引擎对 `store`、`aggregate`、`cap`、`limit` 和 `hasLabel` 的优化。
- 添加了对 openCypher 三角函数的支持：
  - `acos()`
  - `asin()`
  - `atan()`
  - `atan2()`
  - `cos()`
  - `cot()`
  - `degrees()`
  - `pi()`
  - `radians()`
  - `sin()`
  - `tan()`
- 添加了对多个 openCypher 聚合函数的支持：
  - `percentileDisc()`
  - `stDev()`

- 添加了对将 `datetime` 转换为 `epochmillis` 的 openCypher `epochmillis()` 函数的支持。例如：

```
MATCH (n) RETURN epochMillis(n.someDateTime)
1698972364782
```

- 添加了对 openCypher 模数 (%) 运算符的支持。
- 添加了对 openCypher 静态调试解释工具的支持。
- 添加了对 openCypher `randomUUID()` 函数的支持。
- 提高了 openCypher 性能：
  - 改进了解析器和查询计划器。
  - 提高了 DFE 引擎中的 CPU 利用率。
  - 提高了包含多个重用相同变量的更新子句的查询的性能。示例为：

```
MERGE (n {name: 'John'})
  or
MERGE (m {name: 'Jim'})
  or
MERGE (n)-[:knows {since: 2023}]#(m)
```

  - 针对多跳查询模式优化了查询计划，例如：

```
MATCH (n)-->()-->()-->(m)
RETURN n m
```

  - 通过参数化查询提高了列表和映射注入的性能。例如：

```
UNWIND $idList as id MATCH (n {`~id`: id})
RETURN n.name
```

  - 通过使 `WITH` 成为适当的屏障，改进了包含它的查询执行。
  - 经过优化，可避免在 `Unfold` 和聚合函数中对值进行冗余具体化。
- 提高了 `VALUES` 子句中包含大量静态输入的 SPARQL 查询的性能，例如：

```
SELECT ?n WHERE { VALUES (?name) { ("John") ("Jim") ... many values ... } ?n a ?
n_type . ?n ?name . }
```

- 提高了 SPARQL CBD 查询性能。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即在查询计划阶段，深度嵌套的长查询会导致 CPU 使用率过高和查询超时。
- 修复了使用 `mergeV` 或 `mergeE` 时可能引发无效 `NullPointerException` 的 Gremlin 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.1.0.R3 的升级路径

升级到此版本

Amazon Neptune 1.2.1.0.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.1.0.R2（2023 年 5 月 2 日）

截至 2023 年 5 月 2 日，引擎版本 1.2.1.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 为所有 Neptune ML API 添加了 `enableInterContainerTrafficEncryption` 参数，您可以使用该参数在训练或超参数调整任务中启用和禁用容器间流量加密。
- 增加了对 Gremlin `mergeV()` 和 `mergeE()` 的多标签支持。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即更新和返回查询无法正确处理 `orderBy`、`limit` 或 `skip`。
- 修复了一个 openCypher 错误，该错误允许一个请求中包含的参数被另一个同时请求中包含的参数所覆盖。
- 修复了一个 openCypher 错误，即慢速查询日志不包含正确的查询时间。
- 修复了一个 Gremlin 错误，即当包含 `GroupCountStep` 的查询作为字符串提交时，可能会发生事务泄漏。
- 修复了启用慢速查询日志时 WebSocket 查询失败的 Gremlin 错误。
- 修复了一个 Gremlin 错误，即 WebSocket 请求的慢速查询日志中缺少存储计数器调试日志。
- 修复了几个涉及 `mergeV()` 和 `mergeE()` 的 Gremlin 错误。
- 修复了一个 SPARQL 错误，即命名图形查询成本被错误估计，从而导致次优查询计划和内存不足错误。
- 修复了在支持 IAM 的集群上影响 Gremlin 和 openCypher 查询授权的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.1.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.6.2`
- 支持的 Gremlin 最新版本：`3.6.2`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.1.0.R2 的升级路径

升级到此版本

Amazon Neptune 1.2.1.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
```

```
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2（2022 年 11 月 20 日）

截至 2022 年 11 月 20 日，引擎版本 1.2.0.2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此版本的后续补丁版本

- 版本：1.2.0.2.R2（2022 年 12 月 15 日）
- 版本：1.2.0.2.R3（2023 年 3 月 27 日）
- 版本：1.2.0.2.R4（2023 年 5 月 8 日）
- 版本：1.2.0.2.R5（2023 年 8 月 16 日）
- 版本：1.2.0.2.R6（2023 年 9 月 12 日）

此引擎版本中的新增功能

- 在 Neptune ML 中引入了 Gremlin 的实时归纳推理。
- 引入了 openCypher 扩展，该扩展支持为实体指定自定义 ID 值，而不是 Neptune 原本应生成的 UUID。分配自定义 ID 的功能使从 Neo4j 迁移到 Neptune 变得更加容易。

  > **Warning**
  >
  > openCypher 规范的这一扩展不向后兼容，因为 `~id` 现在视为保留的属性名称。如果您已在数据和查询中将 `~id` 用作属性，则在升级到此版本之前，必须将 `~id` 属性迁移到新的属性键。

- 添加了几种新的 SPARQL `DESCRIBE` 模式以及用于配置它们的查询提示。

此引擎版本中的改进

- 提高了 openCypher 的性能，特别是对于 VLP 查询。
- 改进了具有非终端限制的 Gremlin 查询的 DFE 性能，例如：

```
g.withSideEffect('Neptune#useDFE',true).V().hasLabel('Student').limit(5).out('takesCourse')
```

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2 的升级路径

升级到此版本

Amazon Neptune 1.2.0.2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2.R6（2023 年 9 月 12 日）

截至 2023 年 9 月 12 日，引擎版本 1.2.0.2.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

在此引擎版本中修复的缺陷

- 修复了一个 SPARQL 错误，即在使用带有语言标签的文本上调用 REGEX 运算符时从不会成功。
- 修复了导致批量加载性能下降的问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2.R6 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.5`

- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2.R6 的升级路径

如果您正在运行引擎版本 `1.2.0.2`，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此维护补丁版本。

升级到此版本

Amazon Neptune 1.2.0.2.R6 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2.R5（2023 年 8 月 16 日）

截至 2023 年 8 月 16 日，引擎版本 1.2.0.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

在某些情况下，此引擎版本中引入的更改可能会导致您观察到批量加载性能下降。因此，到此版本的升级已临时暂停，直到此问题得到解决。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即当其中一些字符串输出包含空格字符时，`order()` 无法对字符串输出正确排序。

- 修复了一个 Gremlin 错误，即在检查 Gremlin 查询状态端点以查找子遍历中具有谓词的查询是否具有未原生处理的步骤时，会发生事务泄漏。
- 修复了 Bolt 事务处理中的一个 openCypher 错误。
- 修复了存储层上可能导致崩溃的并发问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2.R5 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.5`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2.R5 的升级路径

如果您正在运行引擎版本 `1.2.0.2`，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此维护补丁版本。

升级到此版本

Amazon Neptune 1.2.0.2.R5 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
```

```
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2.R4（2023 年 5 月 8 日）

截至 2023 年 5 月 8 日，引擎版本 1.2.0.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

在此引擎版本中修复的缺陷

- 修复了通过 `VALUES` 子句注入的大量值可能导致性能下降的 SPARQL 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2.R4 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.6`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2.R4 的升级路径

如果您正在运行引擎版本 `1.2.0.2`，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此维护补丁版本。

升级到此版本

Amazon Neptune 1.2.0.2.R4 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2.R3（2023 年 3 月 27 日）

截至 2023 年 3 月 27 日，引擎版本 1.2.0.2.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 对于无服务器数据库集群，将最小容量设置更改为 1.0 NCU，并将最低有效最大容量设置更改为 2.5 NCU。请参阅 Neptune 无服务器数据库集群中的容量扩展。
- 为所有 Neptune ML API 添加了 `enableInterContainerTrafficEncryption` 参数，您可以使用该参数在训练或超参数调整任务中启用和禁用容器间流量加密。

在此引擎版本中修复的缺陷

- 修复了 `option(Predicate)` 未被识别为有效 Gremlin 语法的 Gremlin 错误。
- 修复了一个 Gremlin 错误，该错误会导致查询由于包含太多步骤而失败时无法正确清理查询。
- 修复了一个 Gremlin 正确性问题，该问题通过回退到 Tinkerpop 来影响将 `limit` 作为非并集步骤的子遍历的 DFE 查询。下面是此类查询的示例：

```
g.withSideEffect('Neptune#useDFE', true).V().as("a").select("a").by(out().limit(1))
```

- 修复了当以字符串形式提交的查询包含 `GroupCountStep` 时可能出现的 Gremlin 事务泄漏问题。
- 修复了一个 openCypher 错误，即在列表或映射列表中未正确推断出参数值的类型。
- 修复了一个 openCypher 错误，即更新和返回查询无法正确处理 `orderBy`、`limit` 或 `skip`。
- 修复了一个 openCypher 错误，该错误允许一个请求中包含的参数被另一个同时请求中包含的参数所覆盖。
- 修复了在 `VALUES` 子句中注入大量的值可能导致性能下降的 SPARQL 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2.R3 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.6`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2.R3 的升级路径

如果您正在运行引擎版本 `1.2.0.2`，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此维护补丁版本。

升级到此版本

Amazon Neptune 1.2.0.2.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.2.R2（2022 年 12 月 15 日）

截至 2022 年 12 月 15 日，引擎版本 1.2.0.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 提高了涉及 `MERGE` 和 `OPTIONAL MATCH` 的 openCypher 查询的性能。
- 改进了 openCypher 查询的性能，这些查询涉及文本值映射列表的 `UNWIND`。
- 改进了对 `id` 具有 `IN` 筛选条件的 openCypher 查询的性能。例如：

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- 各种 Gremlin 运算符的性能改进和正确性修复，包括 `repeat`、`coalesce`、`store` 和 `aggregate`。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 `"null"`，而不是 null 值。
- 修复了一个 Gremlin 错误，该错误导致附加到 `UnionStep` 的步骤标签无法传播到其子遍历的最后一个路径元素。
- 修复了一个 Gremlin 错误，该错误导致 `valueMap()` 无法在 DFE 引擎的 `by()` 遍历下进行优化。
- 修复了一个 Gremlin 错误，即作为较长 Gremlin 事务的一部分执行的读取查询不会锁定行。
- 修复了一个审计日志错误，该错误导致记录不必要的信息以及日志中缺少某些字段。
- 修复了一个审计日志错误，即未记录针对启用了 IAM 的数据库集群发出的 HTTP 请求的 IAM ARN。
- 修复了一个查找缓存错误，以限制用于写入缓存的增量内存。
- 修复了一个查找缓存错误，该错误涉及在写入失败时为查找缓存设置只读模式。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.2.R2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.2.R2 的升级路径

如果您正在运行引擎版本 `1.2.0.2`，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此维护补丁版本。

升级到此版本

Amazon Neptune 1.2.0.2.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.1（2022 年 10 月 26 日）

截至 2022 年 10 月 26 日，引擎版本 1.2.0.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。
>
>   如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
> - 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此版本的后续补丁版本

- 维护版本：1.2.0.1.R2（2022 年 12 月 13 日）
- 维护版本：1.2.0.1.R3（2023 年 9 月 27 日）

此引擎版本中的新增功能

- 推出了 Amazon Neptune 无服务器，这是一种按需自动扩缩配置，可以横向扩展数据库集群以满足处理需求的增加，然后在需求减少时再缩减。

此引擎版本中的改进

- 改进了 Gremlin `order-by` 查询的性能。`NeptuneGraphQueryStep` 末尾带有 `order-by` 的 Gremlin 查询现在使用更大的块大小以获得更好的性能。这不适用于查询计划的内部（非根）节点上的 `order-by`。
- 改进了 Gremlin 更新查询的性能。现在，在添加边缘或属性时，必须锁定顶点和边缘以防止删除。此更改消除了事务中的重复锁定，从而提高了性能。
- 通过将 dedup 向下推送到原生执行层，改进了在 `repeat()` 子查询内使用 `dedup()` 的 Gremlin 查询的性能。
- 为 IAM 身份验证错误添加了用户友好的错误消息。这些消息现在会显示您的 IAM 用户或角色 ARN、资源 ARN 以及针对该请求的未经授权的操作列表。未经授权的操作列表可帮助您查看正在使用的 IAM policy 中可能缺少或显式拒绝的内容。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即在升级到 TinkerPop 3.5 后错误地使用 `PartitionStrategy` 导致了一个错误，消息为“PartitionStrategy 不适用于匿名遍历”，从而阻止了遍历的执行。
- 修复了涉及 `WherePredicateStep` 翻译的 Gremlin 正确性错误，即 Neptune 的查询引擎为使用 `where(P.neq('x'))` 及其变体的查询生成了错误的结果。
- 修复了 `MERGE` 子句中的 openCypher 错误，该错误在某些情况下会导致创建重复的节点和边缘。
- 修复了在处理 `OPTIONAL` 子句中包含 `(NOT) EXISTS` 的查询时的一个 SPARQL 错误，在某些情况下，查询结果会丢失。
- 修复了在大量插入负载下导致性能回归的批量加载程序错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.1 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`

- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.1 的升级路径

升级到此版本

Amazon Neptune 1.2.0.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.1.R3（2023 年 9 月 27 日）

截至 2023 年 9 月 27 日，引擎版本 1.2.0.1.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列

neptune1.2 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 neptune1，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。
- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 为所有 Neptune ML API 添加了 `enableInterContainerTrafficEncryption` 参数，您可以使用该参数在训练或超参数调整任务中启用和禁用容器间流量加密。
- 对于无服务器数据库集群，将最小容量设置更改为 1.0 NCU，并将最低有效最大容量设置更改为 2.5 NCU。请参阅Neptune 无服务器数据库集群中的容量扩展##########################。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即 update-and-return 查询无法正确处理 `orderBy`、`limit` 或 `skip`。
- 修复了一个 openCypher 错误，该错误允许一个请求中包含的参数被另一个同时请求中包含的参数所覆盖。
- 修复了 Bolt 事务处理中的一个 openCypher 错误。
- 修复了 Gremlin 正确性问题，此类问题通过回退到 Tinkerpop 影响将 `limit` 作为非并集步骤的子遍历的 DFE 查询。例如，对于如下查询：

```
g.withSideEffect('Neptune#useDFE', true)
 .V()
 .as("a")
 .select("a")
 .by(out()
 .limit(1))
```

- 修复了一个 Gremlin 错误，即查询因为包含太多 TinkerPop 步骤而失败，然后无法清理。
- 修复了一个 Gremlin 错误，即当其中一些字符串输出包含空格字符时，`order()` 无法对字符串输出正确排序。
- 修复了一个 Gremlin 错误，即当查询以字符串形式提交并包含 `GroupCountStep` 时，可能会发生事务泄漏。
- 修复了一个 Gremlin 错误，即在检查 Gremlin 查询状态端点以查找子遍历中具有谓词的查询是否具有未原生处理的步骤时，会发生事务泄漏。
- 修复了一个 Gremlin 错误，即添加边缘及其属性，然后添加 `inV()` 或 `outV()` 会导致 `InternalFailureException`。
- 修复了存储层中的并发问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.1.R3 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.6`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.1.R3 的升级路径

如果您正在运行引擎版本 1.2.0.1，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Amazon Neptune 1.2.0.1.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.1.R2（2022 年 12 月 13 日）

截至 2022 年 12 月 13 日，引擎版本 1.2.0.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 改进了 openCypher 查询的性能，这些查询涉及文本值映射列表上的 `UNWIND`。
- 各种 Gremlin 运算符的性能改进和正确性修复，包括 `repeat`、`coalesce`、`store` 和 `aggregate`。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 `"null"`，而不是 null 值。
- 修复了一个审计日志错误，该错误导致记录不必要的信息以及日志中缺少某些字段。
- 修复了一个查找缓存错误，以限制用于写入缓存的增量内存。
- 修复了一个查找缓存错误，该错误涉及在写入失败时为查找缓存设置只读模式。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.1.R2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.1.R2 的升级路径

如果您正在运行引擎版本 1.2.0.1，您的 Neptune 数据库集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Amazon Neptune 1.2.0.1.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.0（2022 年 7 月 21 日）

截至 2022 年 7 月 21 日，引擎版本 1.2.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列

neptune1.2 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 neptune1，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。

- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 UndoLogsListSize CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：request.setResourcePath("/openCypher"));。在其它语言中，可以将 /openCypher 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此版本的后续补丁版本

- 版本：1.2.0.0.R2（2022 年 10 月 14 日）
- 版本：1.2.0.0.R3（2022 年 12 月 15 日）
- 版本：1.2.0.0.R4（2023 年 9 月 29 日）

此引擎版本中的新增功能

- 增加了对全球数据库的支持。一个 Neptune 全球数据库跨多个 AWS 区域，并由一个区域中的一个主数据库集群和其它区域中最多五个辅助数据库集群组成。
- 根据数据面板操作，在 Neptune IAM policy 中增加了对比之前更精细的访问控制的支持。这是一个重大变化，因为必须调整基于已弃用的 connect 操作的现有 IAM policy 以使用更精细的数据面板操作。请参阅IAM policy 的类型。
- 改进了读取器实例的可用性。以前，当写入器实例重启时，Neptune 集群中的所有读取器实例也会自动重启。从引擎版本 1.2.0.0 开始，读取器实例在写入器重启后保持活动状态，这提高了读取器的

可用性。读取器实例可以单独重启以获取参数组的更改。请参阅在 Amazon Neptune 中重启数据库实例。

- 添加了一个新的 neptune_streams_expiry_days 数据库集群参数，允许您设置在删除流记录之前在服务器上保留它们的天数。范围为 1 到 90，默认值为 7。

此引擎版本中的改进

- 提高了字节码查询的 Gremlin 序列化性能。
- Neptune 现在使用 DFE 引擎处理文本谓词，以提高性能。
- Neptune 现在使用 DFE 引擎处理 Gremlin `limit()` 步骤，包括非终端和子遍历限制。
- 更改了 DFE 对 Gremlin `union()` 步骤的处理以与其它新特征配合使用，这意味着引用节点会按预期显示在查询配置文件中。
- 通过并行化 DFE 中一些昂贵的联接操作，将性能提高多达 5 倍。
- 对于 Gremlin DFE 引擎，添加了对于 `OrderGlobalStep order(global)` 的 `by()` 调制支持。
- 在 DFE 的 explain 详细信息中添加了注入的静态值的显示。
- 提高了修剪重复模式时的性能。
- 在 Gremlin DFE 引擎中添加了顺序保存支持。
- 提高了带有空筛选条件的 Gremlin 查询的性能，例如：

```
g.V().hasId(P.within([]))
```

```
g.V().hasId([])
```

- 改进了 SPARQL 查询使用的数值太大而使 Neptune 无法在内部表示时的错误消息。
- 通过减少禁用流时的索引搜索，提高了删除带有关联边缘的顶点的性能。
- 将 DFE 支持扩展到 `has()` 步骤的更多变体，特别是扩展到 `hasKey()`、`hasLabel()`，以及 `has()` 中字符串/URI 的范围谓词。这会影响如下查询：

```
// hasKey() on properties
g.V().properties().hasKey("name")
g.V().properties().has(T.key, TextP.startingWith("a"))
g.E().properties().hasKey("weight")
g.E().properties().hasKey(TextP.containing("t"))

// hasLabel() on vertex properties
```

```
g.V().properties().hasLabel("name")

// range predicates on ID and Label fields
g.V().has(T.label, gt("person"))
g.E().has(T.id, lte("(an ID value)"))
```

- 添加了 Neptune 特定的 openCypher join() 函数，该函数可将列表中的字符串串联成单个字符串。
- 更新了 Neptune 托管式策略，以包含数据访问权限和新的全球数据库 API 的权限。

在此引擎版本中修复的缺陷

- 修复了未指定内容类型的 HTTP 请求会自动失败的错误。
- 修复了查询优化器中阻止在查询中使用服务调用的 SPARQL 错误。
- 修复了 Turtle RDF 解析器中的一个 SPARQL 错误，即特定的 Unicode 数据组合会导致失败。
- 修复了 SPARQL 错误，即 GRAPH 和 SELECT 子句的特定组合会生成错误的查询结果。
- 修复了一个 Gremlin 错误，该错误会导致在联合步骤中使用任何筛选步骤的查询出现正确性问题，例如：

```
g.V("1").union(hasLabel("person"), out())
```

- 修复了一个 Gremlin 错误，即 both().simplePath() 的 count() 将导致在没有 count() 的情况下返回的实际结果数翻倍。
- 修复了一个 openCypher 错误，即服务器在向启用了 IAM 身份验证的集群发出 Bolt 请求时生成了错误的签名不匹配异常。
- 修复了一个 openCypher 错误，即如果在请求失败后提交，则使用 HTTP 保持活动状态的查询可能会被错误关闭。
- 修复了一个 openCypher 错误，该错误可能导致在提交返回常量值的查询时引发内部错误。
- 修复了 explain 详细信息中的一个错误，以便 DFE 子查询 Time(ms) 现在可以正确计算出 DFE 子查询中运算符的 CPU 时间。以下面的 explain 输出摘录为例：

```
subQuery1
#######################################################################################################
# ID # Out #1 # Out #2 # Name                    # Arguments                                          #
 Mode # Units In # Units Out # Ratio # Time (ms) #
#######################################################################################################
  ...
```

```
##########################################################################################
# 1  # 2      # -      # DFEChunkLocalSubQuery # subQuery=...graph#336e.../graph_1 #
 -    # 1        # 1         # 1.00  # 0.38      #
#    #        #        #                       # coordinationTime(ms)=0.026        #
     #          #           #       #           #
##########################################################################################
  ...
subQuery=...graph#336e.../graph_1
##########################################################################################
# ID # Out #1 # Out #2 # Name                  # Arguments                          #
 Mode # Units In # Units Out # Ratio # Time (ms) #
##########################################################################################
# 0  # 1      # -      # DFESolutionInjection  # solutions=[?100 -> [-10^^<LONG>]] #
 -    # 0        # 1         # 0.00  # 0.04      #
#    #        #        #                       # outSchema=[?100]                   #
     #          #           #       #           #
##########################################################################################
# 1  # 3      # -      # DFERelationalJoin     # joinVars=[]                        #
 -    # 2        # 1         # 0.50  # 0.29      #
##########################################################################################
# 2  # 1      # -      # DFESolutionInjection  # outSchema=[]                      #
 -    # 0        # 1         # 0.00  # 0.01      #
##########################################################################################
# 3  # -      # -      # DFEDrain              # -                                  #
 -    # 1        # 0         # 0.00  # 0.02      #
##########################################################################################
```

下表最后一列中的子查询时间加起来为 0.36 毫秒 (.04 + .29 + .01 + .02 = .36)。当您将该子查询的协调时间相加 (.36 + .026 = .386) 时，您会得到一个结果，该结果接近于上表最后一列中记录的子查询的时间，即 0.38 毫秒。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：3.5.2
- 支持的 Gremlin 最新版本：3.5.4
- openCypher 版本：Neptune-9.0.20190305-1.0
- SPARQL 版本：1.1

引擎版本 1.2.0.0 的升级路径

由于这是主要引擎版本，因此无法对其进行自动升级。

您只能从引擎版本 1.1.1.0 的最新补丁版本手动升级到版本 1.2.0.0。早期引擎版本必须先升级到 1.1.1.0 的最新版本，然后才能升级到 1.2.0.0。

因此，在尝试升级到此版本之前，请确认您当前正在运行版本 1.1.1.0 的最新补丁版本。否则，请先升级到 1.1.1.0 的最新补丁版本。

升级之前，您还必须使用参数组系列 neptune1.2 重新创建您在先前版本中使用的任何自定义数据库集群参数组。请参阅Amazon Neptune 参数组了解更多信息。

如果您先升级到版本 1.1.1.0，然后立即升级到 1.2.0.0，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成（请参阅维护 Amazon Neptune 数据库集群）。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
```

```
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.0.R4（2023 年 9 月 29 日）

截至 2023 年 9 月 29 日，引擎版本 1.2.0.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Note

如果从 1.2.0.0 之前的引擎版本升级：

- 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
- 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

  您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

  如果 `UndoLogsListSize` CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 为所有 Neptune ML API 添加了 `enableInterContainerTrafficEncryption` 参数，您可以使用该参数在训练或超参数调整任务中启用和禁用容器间流量加密。
- 对于无服务器数据库集群，将最小容量设置更改为 1.0 NCU，并将最低有效最大容量设置更改为 2.5 NCU。请参阅Neptune 无服务器数据库集群中的容量扩展##########。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即 update-and-return 查询无法正确处理 `orderBy`、`limit` 或 `skip`。
- 修复了一个 openCypher 错误，该错误允许一个请求中包含的参数被另一个同时请求中包含的参数所覆盖。
- 修复了 Bolt 事务处理中的一个 openCypher 错误。
- 修复了 Gremlin 正确性问题，此类问题通过回退到 Tinkerpop 影响将 `limit` 作为非并集步骤的子遍历的 DFE 查询。例如，对于如下查询：

```
g.withSideEffect('Neptune#useDFE', true)
 .V()
 .as("a")
 .select("a")
 .by(out()
 .limit(1))
```

- 修复了一个 Gremlin 错误，即查询因为包含太多 TinkerPop 步骤而失败，然后无法清理。
- 修复了一个 Gremlin 错误，即当其中一些字符串输出包含空格字符时，`order()` 无法对字符串输出正确排序。
- 修复了一个 Gremlin 错误，即当查询以字符串形式提交并包含 `GroupCountStep` 时，可能会发生事务泄漏。

- 修复了一个 Gremlin 错误，即在检查 Gremlin 查询状态端点以查找子遍历中具有谓词的查询是否具有未原生处理的步骤时，会发生事务泄漏。
- 修复了一个 Gremlin 错误，即添加边缘及其属性，然后添加 `inV()` 或 `outV()` 会导致 `InternalFailureException`。
- 修复了存储层中的并发问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.0.R4 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.6`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.0.R4 的升级路径

如果您正在运行引擎版本 `1.2.0.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您只能从引擎版本 1.1.1.0 的最新补丁版本手动升级到版本 `1.2.0.0`。早期引擎版本必须先升级到 `1.1.1.0` 的最新版本，然后才能升级到 `1.2.0.0`。

如果您先升级到版本 `1.1.1.0`，然后立即升级到 `1.2.0.0`，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.0.R3（2022 年 12 月 15 日）

截至 2022 年 12 月 15 日，引擎版本 1.2.0.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从

1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。

您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 提高了涉及 `MERGE` 和 `OPTIONAL MATCH` 的 openCypher 查询的性能。
- 改进了 openCypher 查询的性能，这些查询涉及文本值映射列表上的 `UNWIND`。
- 改进了对 `id` 具有 `IN` 筛选条件的 openCypher 查询的性能。例如：

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- 各种 Gremlin 运算符的性能改进和正确性修复，包括 `repeat`、`coalesce`、`store` 和 `aggregate`。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 `"null"`，而不是 null 值。
- 修复了一个 openCypher 错误，以便能够在值为列表或映射列表时正确地解释参数类型。
- 修复了一个审计日志错误，该错误导致记录不必要的信息以及日志中缺少某些字段。
- 修复了一个审计日志错误，即未记录针对启用了 IAM 的数据库集群发出的 HTTP 请求的 IAM ARN。
- 修复了一个查找缓存错误，以限制用于写入缓存的增量内存。
- 修复了一个查找缓存错误，该错误涉及在写入失败时为查找缓存设置只读模式。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.0.R3 的升级路径

如果您正在运行引擎版本 `1.2.0.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您只能从引擎版本 1.1.1.0 的最新补丁版本手动升级到版本 `1.2.0.0`。早期引擎版本必须先升级到 `1.1.1.0` 的最新版本，然后才能升级到 `1.2.0.0`。

如果您先升级到版本 `1.1.1.0`，然后立即升级到 `1.2.0.0`，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.2.0.0.R2（2022 年 10 月 14 日）

截至 2022 年 10 月 14 日，引擎版本 1.2.0.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Note**
>
> 如果从 1.2.0.0 之前的引擎版本升级：
>
> - 引擎版本 1.2.0.0 为自定义参数组和自定义集群参数组引入了一种新格式。因此，如果您要从 1.2.0.0 之前的引擎版本升级到引擎版本 1.2.0.0 或更高版本，则必须使用参数组系列 `neptune1.2` 重新创建所有现有的自定义参数组和自定义集群参数组。早期版本使用参数组系列 `neptune1`，而这些参数组不适用于版本 1.2.0.0 及更高版本。请参阅Amazon Neptune 参数组了解更多信息。
> - 引擎版本 1.2.0.0 还引入了一种新的撤消日志格式。因此，必须清除早期引擎版本创建的所有撤消日志，并且 `UndoLogsListSize` CloudWatch 指标必须降至零，然后才能开始从 1.2.0.0 之前的版本进行任何升级。如果您尝试开始更新时撤消日志记录过多（200000 或更多），则在等待清除撤消日志完成时，升级尝试可能会超时。
>
>   您可以通过升级集群的写入器实例（清除发生的地方）来加快清除速率。在尝试升级之前执行此操作可能会在开始之前减少撤消日志的数量。将写入器的大小增加到 24XL 实例类型，可以将清除率提高到每小时超过一百万条记录。

如果 UndoLogsListSize CloudWatch 指标非常大，那么提出支持案例可能会帮助您探索降低该指标的其它策略。

- 最后，1.2.0.0 版本中有一项重大变化，会影响之前使用 Bolt 协议和 IAM 身份验证的代码。从版本 1.2.0.0 开始，Bolt 需要一个用于 IAM 签名的资源路径。在 Java 中，设置资源路径可能如下所示：`request.setResourcePath("/openCypher"));`。在其它语言中，可以将 `/openCypher` 附加到端点 URI 之后。有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 改进了 Gremlin `order-by` 查询的性能。`NeptuneGraphQueryStep` 末尾带有 `order-by` 的 Gremlin 查询现在使用更大的块大小以获得更好的性能。这不适用于查询计划的内部（非根）节点上的 `order-by`。
- 改进了 Gremlin 更新查询的性能。现在，在添加边缘或属性时，必须锁定顶点和边缘以防止删除。此更改消除了事务中的重复锁定，从而提高了性能。
- 通过将 dedup 向下推送到原生执行层，改进了在 `repeat()` 子查询内使用 `dedup()` 的 Gremlin 查询的性能。
- 添加了 Gremlin `Neptune#cardinalityEstimates` 查询提示。如果设置为 `false`，这会禁用基数估计。
- 为 IAM 身份验证错误添加了用户友好的错误消息。这些消息现在会显示您的 IAM 用户或角色 ARN、资源 ARN 以及针对该请求的未经授权的操作列表。未经授权的操作列表可帮助您查看正在使用的 IAM policy 中可能缺少或显式拒绝的内容。

在此引擎版本中修复的缺陷

- 修复了涉及 `WherePredicateStep` 翻译的 Gremlin 正确性错误，即 Neptune 的查询引擎为使用 `where(P.neq('x'))` 及其变体的查询生成了错误的结果。
- 修复了一个 Gremlin 错误，即在升级到 TinkerPop 3.5 后错误地使用 `PartitionStrategy` 导致了一个错误，消息为“PartitionStrategy 不适用于匿名遍历”，从而阻止了遍历的执行。
- 修复了与最终联接的 `joinTime` 以及 `Project.ASK` 子组内的统计数据相关的各种 Gremlin 错误。
- 修复了 `MERGE` 子句中的 openCypher 错误，该错误在某些情况下会导致创建重复的节点和边缘。
- 修复了一个事务错误，也即，即使回滚了相应的并发字典插入，会话也可以插入图形数据并提交。
- 修复了在大量插入负载下导致性能回归的批量加载程序错误。

- 修复了在处理 OPTIONAL 子句中包含 (NOT) EXISTS 的查询时的一个 SPARQL 错误，在某些情况下，查询结果会丢失。
- 修复了一个错误，即在评估开始之前由于超时而导致请求被取消的情况下，驱动程序可能显示为挂起。如果在请求队列中的项目发生超时时使用了服务器上的所有查询处理线程，则可能进入此状态。由于请求队列中的超时并未立即发送消息，因此对于客户端而言，响应仍处于待处理状态。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.2.0.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.2.0.0.R2 的升级路径

如果您正在运行引擎版本 `1.2.0.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您只能从引擎版本 1.1.1.0 的最新补丁版本手动升级到版本 `1.2.0.0`。早期引擎版本必须先升级到 `1.1.1.0` 的最新版本，然后才能升级到 `1.2.0.0`。

如果您先升级到版本 `1.1.1.0`，然后立即升级到 `1.2.0.0`，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.2.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.2.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0（2022 年 4 月 19 日）

截至 2022 年 4 月 19 日，引擎版本 1.1.1.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Important**
>
> 从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
> 为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 preupgrade 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。
此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

此版本的后续补丁版本

- 维护版本：1.1.1.0.R2（2022 年 5 月 16 日）
- 版本：1.1.1.0.R3（2022 年 6 月 7 日）
- 版本：1.1.1.0.R4（2022 年 6 月 23 日）
- 版本：1.1.1.0.R5（2022 年 7 月 21 日）
- 版本：1.1.1.0.R6（2022 年 9 月 23 日）
- 版本：1.1.1.0.R7（2023 年 1 月 23 日）

此引擎版本中的新增功能

- openCypher 查询语言现在可普遍用于生产环境。

> **Warning**
>
> 此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。在 openCypher 的 Neptune 预览版中，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> 从该引擎版本开始，必须省略协议：
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 有关示例，请参阅 使用 Bolt 协议。

- 增加了对 TinkerPop `3.5.2` 的支持。此版本中的更改包括对远程事务的支持和对会话的字节码支持（使用 `g.tx`），以及在 Gremlin 语言中添加了 `datetime()` 函数。

> **Warning**
>
> TinkerPop 3.5.0、3.5.1 和 3.5.2 中引入了几项重大更改，这些更改可能会影响您的 Gremlin 代码。例如，如下所示的使用由 GraphTraversalSource 生成的遍历作为子项将不再起作用：`g.V().union(identity(), g.V())`。
> 现在改用如下所示的匿名遍历：`g.V().union(identity(), __.V())`。

- 增加了对 AWS 全局条件键的支持，您可以在 IAM 数据访问策略中使用这些条件键，这些策略控制对存储在 Neptune 数据库集群中的数据的访问。
- Neptune DFE 查询引擎现在可以通过 openCypher 查询语言在生产环境中普遍使用，但尚未可用于 Gremlin 和 SPARQL 查询。现在，您可以使用它自己的 neptune_dfe_query_engine 实例参数而不是实验室模式参数来启用它。

此引擎版本中的改进

- 向 openCypher 添加了新特征，例如参数化查询支持、参数化查询的抽象语法树 (AST) 缓存、可变长度路径 (VLP) 改进以及新的运算符和子句。有关当前的语言支持级别，请参阅亚马逊 Neptune 中符合 OpenCypher 规范。
- 对 openCypher 进行了显著的性能改进，适用于简单的读写工作负载，与版本 1.1.0.0 相比，吞吐量更高。

- 移除了 openCypher 处理可变长度路径的双向和深度限制。
- 在 DFE 引擎中完成了对 Gremlin `within` 和 `without` 谓词的支持，包括它们与其它谓词运算符组合的情况。例如：

```
g.V().has('age', within(12, 15, 18).or(gt(30)))
```

- 当范围为全局（即不是 `order(local)`）且不使用 `by()` 调制器时，在 DFE 引擎中扩展了对 Gremlin `order` 步骤的支持。例如，此查询现在将具有 DFE 支持：

```
 g.V().values("age").order()
```

- 在 Neptune 流更改日志响应格式中添加了一个 `isLastOp` 字段，以指示记录是其事务中的最后一个操作。
- 启用审计日志记录后，显著提高了审计日志记录的性能并减少了延迟。
- 在审计日志中将 Gremlin WebSocket 字节码和 HTTP 查询转换为用户可读的格式。现在可以直接从审计日志中复制查询，以便在 Neptune 笔记本和其它地方执行。请注意，对当前审计日志格式的这一更改构成了重大更改。

在此引擎版本中修复的缺陷

- 修复了一个罕见的 Gremlin 错误，即在组合使用嵌套 `filter()` 和 `count()` 步骤时未返回任何结果，例如在以下查询中：

```
g.V("1").filter(out("knows")
        .filter(in("knows")
        .hasId("notExists"))
        .count())
```

- 修复了一个 Gremlin 错误，即在将 `to()` 或 `from()` 遍历中聚合步骤存储的顶点与 `addE` 步骤结合使用时返回错误。此类查询的示例为：

```
g.V("id").aggregate("v").out().addE().to(select("v").unfold()))
```

- 修正了使用 DFE 引擎时，`not` 步骤在边缘情况下失败的 Gremlin 错误。例如：

```
g.V().not(V())
```

- 修正了 `sideEffect` 值在 `to()` 和 `from()` 遍历中不可用的 Gremlin 错误。

- 修复了偶尔会导致快速重置以触发实例失效转移的错误。
- 修复了批量加载程序错误，即在开始下一个加载任务之前不会关闭失败的事务。
- 修复了批量加载程序错误，即内存不足可能会导致系统崩溃。
- 添加了重试功能以修复批量加载程序错误，即在失效转移后，加载程序等待的时间不够长，IAM 凭证无法变为可用。
- 修复了未正确清除非查询端点（例如 `status` 端点）的内部凭证缓存的错误。
- 修复了一个流错误，以确保流提交序列号的顺序正确。
- 修复了在启用 IAM 的集群上长时间运行的连接在十天内终止的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。请注意，主版本引擎 (1.1.0.0) 之前的版本需要更长的时间才能升级到此版本。

您不会自动升级到此版本。

升级到此版本

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。
此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine neptune \
    --engine-version 1.1.1.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine neptune ^
    --engine-version 1.1.1.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0.R7（2023 年 1 月 23 日）

截至 2023 年 1 月 23 日，引擎版本 1.1.1.0.R7 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：

- Applying off-line patches to DB instance
- DB instance shutdown
- Finished applying off-line patches to DB instance
- DB instance restarted

此引擎版本中的改进

- 提高了涉及 MERGE 和 OPTIONAL MATCH 的 openCypher 查询的性能。
- 改进了 openCypher 查询的性能，这些查询涉及文本值映射列表的 UNWIND。
- 改进了对 id 具有 IN 筛选条件的 openCypher 查询的性能。例如：

```
MATCH (n) WHERE id(n) IN ['1', '2', '3'] RETURN n
```

- 各种 Gremlin 运算符的性能改进和正确性修复，包括 repeat、coalesce、store 和 aggregate。

在此引擎版本中修复的缺陷

- 修复了一个 openCypher 错误，即如果在请求失败后提交，则使用 HTTP 保持活动状态的请求可能会被错误关闭。
- 修复了一个 openCypher 错误，即列表或映射列表的参数类型并非总是能正确地解释。
- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 "null"，而不是 null 值。
- 修复了查询超时失败和内存不足错误的 openCypher 错误代码和错误消息。
- 修复了一个 Gremlin 错误，该错误导致 valueMap() 无法在 DFE 引擎的 by() 遍历下进行优化。
- 修复了死锁探测器逻辑偶尔会导致引擎无响应的问题。
- 修复了一个审计日志错误，该错误导致记录不必要的信息以及日志中缺少某些字段。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R7 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：3.5.2

- 支持的 Gremlin 最新版本：`3.5.3`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R7 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0.R6（2022 年 9 月 23 日）

截至 2022 年 9 月 23 日，引擎版本 1.1.1.0.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Important**
>
> 从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
>
> 为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。
>
> 升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
>
> 当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。
>
> 此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> 此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。到目前为止，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> 从引擎版本 `1.1.1.0` 开始，必须省略协议：
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 改进了 Gremlin `order-by` 查询的性能。`NeptuneGraphQueryStep` 末尾带有 `order-by` 的 Gremlin 查询现在使用更大的块大小以获得更好的性能。这不适用于查询计划的内部（非根）节点上的 `order-by`。
- 改进了 Gremlin 更新查询的性能。现在，在添加边缘或属性时，必须锁定顶点和边缘以防止删除。此更改消除了事务中的重复锁定，从而提高了性能。

在此引擎版本中修复的缺陷

- 修复了 MERGE 子句中的 openCypher 错误，该错误在某些情况下会导致创建重复的节点和边缘。
- 修复了在处理 OPTIONAL 子句中包含 (NOT) EXISTS 的 SPARQL 查询时的一个错误，也即，在某些情况下，查询结果会丢失。
- 修复了在批量加载过程中延迟服务器重启的错误。
- 修复了一个错误，即在关系属性上使用筛选条件的 openCypher 可变长度模式双向遍历会导致错误。这种可变长度模式的一个例子是 `(n)-[r*1..2]->(m)`。
- 修复了与如何将缓存的数据发送回客户端相关的错误，该错误在某些情况下会导致意想不到的长延迟。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R6 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R6 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

> **Important**
>
> 从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
>
> 升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。
此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0.R5（2022 年 7 月 21 日）

截至 2022 年 7 月 21 日，引擎版本 1.1.1.0.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> 此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。到目前为止，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> 从引擎版本 `1.1.1.0` 开始，必须省略协议：
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 进行了改进以支持死锁检测。

在此引擎版本中修复的缺陷

- 修复了在某些条件下无法完全从容关闭数据库集群的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R5 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R5 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

> **Important**
>
> 从 **1.1.0.0** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
>
> 升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
>
> 当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。
>
> 此过程会生成以下事件：
>
> - 每个集群的事件消息：
>   - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
>   - `Database cluster major version has been upgraded`
> - 每个实例的事件消息：
>   - `Applying off-line patches to DB instance`
>   - `DB instance shutdown`
>   - `Finished applying off-line patches to DB instance`
>   - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0.R4（2022 年 6 月 23 日）

截至 2022 年 6 月 23 日，引擎版本 1.1.1.0.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`

- DB instance restarted

> **Note**
>
> 此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。到目前为止，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> 从引擎版本 `1.1.1.0` 开始，必须省略协议：
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 更新了 `x2g` 实例类型的实例配置。
- 提高了顶点删除的性能。

在此引擎版本中修复的缺陷

- 修复了 Gremlin 错误，即对于多次调用的查询，或者对于跨某些类型的 ASK 联接的多个读取器调用的查询，解决方案无法保持稳定的顺序。
- 此外，缩小了先前版本中导致 Gremlin 中某些类型的 ASK 联接性能下降的更改范围。
- 修复了在子遍历中存在边缘输入和顶点遍历时 `union()` 步骤中出现的 Gremlin 错误。
- 修复了 Gremlin profile 错误，即一些步骤报告为未经过优化，而实际上已优化。
- 修复了一个 SPARQL 错误，即嵌套在 `UNION` 子句中的 `FILTER` 表达式中使用的变量被分配了无效的作用域信息。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R4 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R4 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the
> instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.1.0.R3（2022 年 6 月 7 日）

截至 2022 年 6 月 7 日，引擎版本 1.1.1.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

> **Important**
>
> 从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
> 升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
> 当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。
> 此过程会生成以下事件：
>
> - 每个集群的事件消息：
>   - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
>   - `Database cluster major version has been upgraded`

- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> 此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。到目前为止，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：
>
> ```
> "Host":"bolt://(host URL):(port)"
> ```
>
> 从引擎版本 `1.1.1.0` 开始，必须省略协议：
>
> ```
> "Host":"(host URL):(port)"
> ```
>
> 有关示例，请参阅 使用 Bolt 协议。

此引擎版本中的改进

- 增加了对基于 Graviton2 的 `x2g` 实例类型的支持，此实例类型针对内存密集型工作负载进行了优化。这些最初仅在四个 AWS 区域中提供：
  - 美国东部（弗吉尼亚州北部）(`us-east-1`)
  - 美国东部（俄亥俄州）(`us-east-2`)
  - 美国西部（俄勒冈州）(`us-west-2`)
  - 欧洲地区（爱尔兰）(`eu-west-1`)

  有关更多信息，请参阅 Neptune 定价页面。
- 改进了涉及多个边缘或顶点遍历、属性查找或标签查找的 Gremlin 步骤的性能。

在此引擎版本中修复的缺陷

- 修复了在处理子遍历中的 `otherV()` 步骤时出现的 Gremlin 错误。
- 修复了查询中的一个 Gremlin 错误，即 `union` 只有作为子级的筛选步骤。例如：

```
g.V().union(has("name"), out("knows")).out()
```

- 修复了一个 SPARQL 错误，即嵌套在 `UNION` 子句中的 `FILTER` 表达式中使用的变量被分配了无效的作用域信息。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R3 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 维护版本，版本 1.1.1.0.R2（2022 年 5 月 16 日）

截至 2022 年 5 月 16 日，引擎版本 1.1.1.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

Note

此版本对使用 openCypher 和 IAM 身份验证的代码进行了重大更改。到目前为止，IAM 签名中的主机字符串包含协议，例如 `bolt://`，如下所示：

```
"Host":"bolt://(host URL):(port)"
```

从引擎版本 `1.1.1.0` 开始，必须省略协议：

```
"Host":"(host URL):(port)"
```

有关示例，请参阅 使用 Bolt 协议。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.1.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- 支持的 Gremlin 最早版本：`3.5.2`
- 支持的 Gremlin 最新版本：`3.5.4`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.1.0.R2 的升级路径

如果您正在运行引擎版本 `1.1.1.0`，您的集群将在下一个维护时段内自动升级到此维护补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：

    - Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-*(autogenerated snapshot ID)*]
    - Database cluster major version has been upgraded
  - 每个实例的事件消息：
    - Applying off-line patches to DB instance
    - DB instance shutdown
    - Finished applying off-line patches to DB instance
    - DB instance restarted

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.1.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.1.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.1.0.0（2021 年 11 月 19 日）

截至 2021 年 11 月 19 日，引擎版本 1.1.0.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

Note

从此引擎版本开始，Neptune 不再支持 R4 实例类型。如果您在数据库集群中使用 R4 实例，则在升级到此版本之前，必须手动将其替换为其它实例类型。如果您的写入器实例是 R4，请按照以下说明进行移动。

此版本的后续补丁版本

- 维护版本：1.1.0.0.R2（2022 年 5 月 16 日）
- 维护版本：1.1.0.0.R3（2022 年 12 月 23 日）

此引擎版本中的新增功能

- 引入了由 AWS Graviton2 处理器提供支持的通用 T4g 和内存优化型 R6g 数据库实例。与当前一代基于 x86 的同类实例相比，基于 Graviton2 的实例可为各种工作负载提供明显更好的性价比。在这些新的实例类型上，应用程序可以正常运行，升级到它们时无需移植应用程序代码。

  有关定价和区域可用性的更多信息，请参阅 Amazon Neptune 定价页面。
- 在 Neptune ML 中引入了自定义模型。
- 在 Neptune ML 中增加了对 SPARQL 推理查询的支持。
- 为属性图数据添加了一个新的流端点，即：

```
https://Neptune-DNS:8182/propertygraph/stream
```

  这个端点的输出格式（名为 PG_JSON）与旧 gremlin/stream 输出的 GREMLIN_JSON 格式完全相同。

  新的 propertygraph/stream 端点将 Neptune 流支持扩展到 openCypher，并将 gremlin/stream 端点替换为其关联的 GREMLIN_JSON 输出格式。

此引擎版本中的改进

- 对 Neptune 流进行了改进：
  - 在 Neptune 流更改日志响应格式的 records 对象中添加了一个 commitTimestamp 字段，以便为更改日志流中的每个记录提供时间戳。
  - 为 iteratorType 参数添加了 LATEST 值，允许您从流中检索最后一个有效的 eventId。请参阅调用 Streams API。
- 增加了对在 Gremlin 节点分类和回归查询中获取推理置信度分数的支持。
- 添加了对 openCypher 中 OPTIONAL MATCH 子句的支持。
- 添加了对 openCypher 中 MERGE 子句的支持。

- 添加了对在 openCypher 的 `WITH` 子句中使用 `ORDER BY` 的支持。
- 在 openCypher 中增加了对模式理解的支持，并扩展了对存在性检查之外的模式表达式的支持。
- 扩展了对 openCypher 中 `DELETE` 和 `DELETE DETACH` 子句的支持，以便它们现在可以与其它更新子句一起使用。
- 扩展了对 openCypher 中与 `RETURN` 一起使用的 `CREATE` 和 `UPDATE` 子句的支持。
- 在 DFE 引擎中添加了对 Gremlin `limit`、`range` 和 `skip` 步骤的支持。
- 改进了 DFE 引擎中既没有请求 `explain` 也没有请求 `profile` 时的查询执行。
- 改进了 DFE 引擎中 `value` 表达式的查询执行。
- 改进了许多链接的 Gremlin 条件插入模式，以避免并发修改异常并允许链接这样的查询模式：
  - 按 ID 进行有条件的顶点插入，例如：

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1").property(id,ID))
```

  - 使用多个标签进行有条件的顶点插入，例如：

```
g.V(ID).fold().coalesce(unfold(), g.addV("L1::L2").property(id,ID))
```

  - 按 ID 进行有条件的边缘插入，例如：

```
g.E(ID).fold().coalesce(unfold(), V(from).addE(label).to(V(to)).property(id, ID))
```

  - 使用多个标签进行有条件的边缘插入，例如：

```
g.E(ID).fold().coalesce(unfold(),
 g.addE(label).from(V(from)).to(V(to)).property(id, ID))
```

  - 有条件的插入，后跟查询，例如：

```
g.V(ID).fold().coalesce(unfold(),
 g.addV("L1").property(id,ID)).project("myvalues").by(valueMap())
```

  - 带有附加属性的有条件插入，例如：

```
g.V(ID).fold().coalesce(unfold(),
 g.addV("L1").property(id,ID).property("name","pumba"))
```

在此引擎版本中修复的缺陷

- 已对 T3.medium 实例类型禁用统计数据特征，此实例类型无法支持该特征。
- 修复了 explain 中使用非常量值的 IN 函数的 SPARQL 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.0.0 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：3.4.11
- SPARQL 版本：1.1

引擎版本 1.1.0.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --allow-major-version-upgrade \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --allow-major-version-upgrade ^
    --apply-immediately
```

您可以指定 `--no-apply-immediately`，而不是 `--apply-immediately`。要执行主要版本升级，需要使用 allow-major-upgrade 参数。另外，请务必包括引擎版本，否则您的引擎可能会升级到其它版本。

如果集群使用自定义集群参数组，请确保包含以下参数以指定此参数组：

```
    --db-cluster-parameter-group-name (name of the custom DB cluster parameter group)
```

同样，如果集群中的任何实例使用自定义数据库参数组，请确保包含此参数以指定此参数组：

```
    --db-instance-parameter-group-name (name of the custom instance parameter group)
```

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> ```

```
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 维护版本，版本 1.1.0.0.R3（2022 年 12 月 23 日）

截至 2022 年 12 月 23 日，引擎版本 1.1.0.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

为了成功完成升级，每个可用区 (AZ) 中的每个子网对于每个 Neptune 实例都必须至少有一个可用的 IP 地址。例如，如果子网 1 中有一个写入器实例和两个读取器实例，子网 2 中有两个读取器实例，则在开始升级之前，子网 1 必须至少有 3 个空闲的 IP 地址，子网 2 必须至少有 2 个空闲的 IP 地址。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：

- `Applying off-line patches to DB instance`
- `DB instance shutdown`
- `Finished applying off-line patches to DB instance`
- `DB instance restarted`

此引擎版本中的改进

- 各种 Gremlin 运算符的性能改进和正确性修复，包括 `repeat`、`coalesce`、`store` 和 `aggregate`。

在此引擎版本中修复的缺陷

- 修复了 CPU 峰值问题。
- 修复了一个 openCypher 错误，即在 Bolt 和 SPARQL-JSON 中，查询返回字符串 `"null"`，而不是 null 值。
- 修复了一个审计日志错误，该错误导致记录不必要的信息以及日志中缺少某些字段。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.0.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.0.0.R3 的升级路径

如果您正在运行引擎版本 `1.1.0.0`，您的集群将在下一个维护时段内自动升级到此维护补丁版本。

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 preupgrade 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。
此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-(autogenerated snapshot ID)]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`
  - `DB instance shutdown`
  - `Finished applying off-line patches to DB instance`
  - `DB instance restarted`

> **Note**
>
> 从此引擎版本开始，Neptune 不再支持 R4 实例类型。如果您在数据库集群中使用 R4 实例，则在升级到此版本之前，必须手动将其替换为其它实例类型。如果您的写入器实例是 R4，请按照以下说明进行移动。

升级到此版本

Amazon Neptune 1.1.0.0.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
```

```
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 维护版本，版本 1.1.0.0.R2（2022 年 5 月 16 日）

截至 2022 年 5 月 16 日，引擎版本 1.1.0.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

Important

从早于 **`1.1.0.0`** 的版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。

升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。

当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在几分钟内不可用。升级完成后，您可以恢复写入工作负载。

此过程会生成以下事件：

- 每个集群的事件消息：
  - `Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-`*`(autogenerated snapshot ID)`*`]`
  - `Database cluster major version has been upgraded`
- 每个实例的事件消息：
  - `Applying off-line patches to DB instance`

- DB instance shutdown
- Finished applying off-line patches to DB instance
- DB instance restarted

在此引擎版本中修复的缺陷

- 修复了未正确清除非查询端点（例如状态端点）的内部凭证缓存的错误。
- 修复了引擎升级后导致复制滞后增加的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.1.0.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- openCypher 版本：`Neptune-9.0.20190305-1.0`
- SPARQL 版本：`1.1`

引擎版本 1.1.0.0.R2 的升级路径

如果您正在运行引擎版本 `1.1.0.0`，您的集群将在下一个维护时段内自动升级到此维护补丁版本。

Important

从 **`1.1.0.0`** 之前的任何版本升级到此引擎版本还会触发数据库集群中所有实例的操作系统升级。由于不会处理操作系统升级过程中发生的活动写入请求，因此在开始升级之前，必须暂停对正在升级的集群的所有写入工作负载，包括批量数据加载。
升级开始时，Neptune 会根据您的数据库集群信息生成一个快照，其名称由 `preupgrade` 后跟自动生成的标识符组成。您无需为此快照付费，如果升级过程中出现任何问题，您可以使用它来还原数据库集群。
当引擎升级本身完成后，新的引擎版本将在旧操作系统上短暂可用，但不到 5 分钟后，集群中的所有实例将同时开始操作系统升级。此时，您的数据库集群将在大约 6 分钟内不可用。升级完成后，您可以恢复写入工作负载。
此过程会生成以下事件：

- 每个集群的事件消息：

    - Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-*(autogenerated snapshot ID)*]
    - Database cluster major version has been upgraded
  - 每个实例的事件消息：
    - Applying off-line patches to DB instance
    - DB instance shutdown
    - Finished applying off-line patches to DB instance
    - DB instance restarted

> **Note**
>
> 从此引擎版本开始，Neptune 不再支持 R4 实例类型。如果您在数据库集群中使用 R4 实例，则在升级到此版本之前，必须手动将其替换为其它实例类型。如果您的写入器实例是 R4，请按照以下说明进行移动。

升级到此版本

Amazon Neptune 1.1.0.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.1.0.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.1.0.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.1（2021 年 10 月 1 日）

截至 2021 年 10 月 1 日，引擎版本 1.0.5.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.5.1.R2（2021 年 10 月 26 日）
- 版本：1.0.5.1.R3（2022 年 1 月 13 日）
- 维护版本：1.0.5.1.R4（2022 年 5 月 16 日）

此引擎版本中的新增功能

- 添加了用于缓存指定查询的结果的结果缓存。
- 在 Neptune openCypher 中添加了日期/时间支持。
- 增加了对 Neptune openCypher 中元素的 `List` 和 `Map` 访问支持。

此引擎版本中的改进

- 使 Neptune openCypher 端点名称不区分大小写。
- 改进了 openCypher explain。
- 改进了 Gremlin 单个更新插入查询模式，以 `iterate()` 和 `profile()` 步骤结尾。
- 改进了 Gremlin `keys()` 和 `property()` 函数的性能。
- 当 Gremlin `dedup()` 步骤与全局范围一起使用时，该步骤会在 DFE 中运行。
- 启用 DFE 引擎后，以下 Gremlin HAS 谓词将在 DFE 引擎中运行：
  - `EQ`
  - `NEQ`
  - `LT`
  - `LTE`
  - `GT`
  - `GTE`
  - `BETWEEN`
  - `INSIDE`

  - `OUTSIDE`
  - `WITHIN`
  - `AND (connectives)`
  - `OR (connectives)`
- 提高了 LIMIT 查询性能。
- 提高了 openCypher 常规聚合查询的性能。

在此引擎版本中修复的缺陷

- 修复了允许将边缘连接到另一个边缘的 Gremlin 错误。
- 修复了导致选择次优联接策略的 Gremlin 错误。
- 修复了一个 Gremlin 错误，当存在超过 100 个属性时，该错误会导致节点和关系的序列化停止。
- 修复了一个错误，该错误会减慢具有大型图形模式的查询的查询执行计划。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.1 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 维护版本，版本 1.0.5.1.R4（2022 年 5 月 16 日）

截至 2022 年 5 月 16 日，引擎版本 1.0.5.1.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.1.R4 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.1.R4 的升级路径

如果您正在运行引擎版本 `1.0.5.1`，您的集群将在下一个维护时段内自动升级到此维护补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.1.R4 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.1.R3（2022 年 1 月 13 日）

截至 2022 年 1 月 13 日，引擎版本 1.0.5.1.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了当查询无法获取它所需的所有资源时可能导致资源泄漏的错误。
- 修复了在查询执行期间由无人认领的内存分配导致的少量内存泄漏。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.1.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.1.R3 的升级路径

如果您正在运行引擎版本 `1.0.5.1`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.1.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
>   We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.1.R2（2021 年 10 月 26 日）

截至 2021 年 10 月 26 日，引擎版本 1.0.5.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了一个错误，当在可重复读取隔离下创建较旧版本的图形元素的情况下发生临时错误时，该错误导致服务器重启。Neptune 现在改为返回错误，以便客户端可以重试。
- 修复了在单个基数更新期间发生临时错误时导致服务器重启的错误。Neptune 现在改为返回错误，以便客户端可以重试。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.1.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.1.R2 的升级路径

如果您正在运行引擎版本 `1.0.5.1`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.1.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.0（2021 年 7 月 27 日）

截至 2021 年 7 月 27 日，引擎版本 1.0.5.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.5.0.R2（2021 年 8 月 16 日）
- 版本：1.0.5.0.R3（2021 年 9 月 15 日）
- 维护版本：1.0.5.0.R5（2022 年 5 月 16 日）

此引擎版本中的新增功能

- Neptune ML 已发布供生产使用，具有许多新特征，并且不再处于实验室模式。

- 在实验室模式下添加了对 openCypher 查询语言的初始支持。openCypher 是 Cypher 查询语言的开源标准。它的语法在 Cypher 查询语言参考（版本 9）中指定，并由 openCypher 项目维护。

  有关该语言的 Neptune 实现的信息，请参阅使用 openCypher 访问 Neptune 图形。

  还支持 Bolt 协议，Neptune 客户端使用该协议进行 openCypher 查询。请参阅使用 Bolt 协议向 Neptune 进行 openCypher 查询。

  现在会自动启用对 openCypher 的支持，但这取决于 Neptune DFE 引擎，它目前仅在实验室模式下可用。现在，`neptune_lab_mode` 数据库集群参数中的默认 `DFEQueryEngine` 设置为 `DFEQueryEngine=viaQueryHint`，这意味着引擎已启用，但仅用于存在 `useDFE` 查询提示并设置为 `true` 的查询。如果通过设置 `DFEQueryEngine=disabled` 禁用 DFE 引擎，则无法使用 openCypher。
- 增加了对 SPARQL 1.1 图形存储 HTTP 协议的支持。请参阅在 Amazon Neptune 中使用 SPARQL 1.1 图形存储 HTTP 协议 (GSP)。
- 已将Neptune DFE 引擎的默认实验室模式设置更改为 `viaQueryHint`，这意味着 DFE 引擎现在默认处于启用状态，但仅用于 `useDFE` 查询提示存在且设置为 `true` 的查询。
- 添加了一个新的 Amazon CloudWatch 指标 `StatsNumStatementsScanned`，用于监控 Neptune DFE 引擎的统计数据计算。请参阅使用`StatsNumStatementsScanned` CloudWatch 指标监控统计数据计算。

此引擎版本中的改进

- 增加了对 Apache TinkerPop 3.4.11 的支持。

> **Important**
>
> TinkerPop 版本 3.4.11 中进行了一项更改，可提高查询处理方式的正确性，但目前有时会严重影响查询性能。
> 例如，这种查询的运行速度可能会慢得多：
>
> ```
> g.V().hasLabel('airport').
>   order().
>     by(out().count(),desc).
>   limit(10).
>   out()
> ```

由于 TinkerPop 3.4.11 的这一更改，极限步骤之后的顶点现在以非最佳方式获取。为避免这种情况，您可以通过在 `order().by()` 之后的任何点添加 barrier() 步骤来修改查询。例如：

```
g.V().hasLabel('airport').
  order().
    by(out().count(),desc).
  limit(10).
  barrier().
  out()
```

- Neptune DFE 替代查询引擎现在支持 SPARQL `joinOrder` 查询提示。
- Neptune 状态 API 的输出经过扩展和重组，可以更清楚地了解数据库集群的设置和特征。

  新的输出有一个顶级 `features` 对象，其中包含有关数据库集群的特征的状态信息，还有一个包含设置信息的顶级 `settings` 对象。要审核新格式，请参阅实例状态命令的输出示例。
- 如果服务器上最后一个事件 ID 已经过期，则当通过此事件 ID 请求 AFTER_SEQUENCE_NUMBER 个流时，改进了对流更改日志的处理。如果请求的事件 ID 是服务器上最近清除的事件 ID，则服务器不会再引发事件 ID 过期错误。

在此引擎版本中修复的缺陷

- 修复了与数值排序有关的 Gremlin 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.0 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.0 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 维护版本，版本 1.0.5.0.R5（2022 年 5 月 16 日）

截至 2022 年 5 月 16 日，引擎版本 1.0.5.0.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.0.R5 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.0.R5 的升级路径

如果您正在运行引擎版本 1.0.5.0，您的集群将在下一个维护时段内自动升级到此维护补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.0.R5 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.0.R3（2021 年 9 月 15 日）

截至 2021 年 9 月 15 日，引擎版本 1.0.5.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了在以下任一情况下导致引擎无响应的错误：
  - 批量加载是在进行自动统计数据计算的同时进行的。
  - 在已经进行统计数据计算的同时，有人手动请求进行统计数据计算。
- 修复了死锁检测和锁定获取中可能导致引擎崩溃的错误。
- 修复了一个 Gremlin 错误，也即，引擎在 Gremlin 推理查询中遇到来自远程 ML 端点的未知数据时会引发错误。
- 修复了机器学习模型管理 API 中与模型转换任务和实例推荐相关的几个错误。

- 修复了在生成节点和边缘 ID 时会导致引擎崩溃的错误。
- 修复了一个错误，该错误会减慢具有大型图形模式的查询的查询计划生成速度。
- 修复了一个 openCypher 错误，该错误可能导致在检索具有 100 个以上属性的节点时查询停止。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.0.R3 的升级路径

如果您正在运行引擎版本 1.0.5.0，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.0.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.5.0.R2（2021 年 8 月 16 日）

截至 2021 年 8 月 16 日，引擎版本 1.0.5.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 禁用了引擎版本 `1.0.5.0` 中进行的优化，该优化使 Neptune 查找缓存当引擎在副本上重启后仍然有效。现在，副本重启会清除查找缓存。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.5.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.11`
- SPARQL 版本：`1.1`

引擎版本 1.0.5.0.R2 的升级路径

如果您正在运行引擎版本 `1.0.5.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.5.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.5.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.5.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

Note

如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
```

```
proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.2（2021 年 6 月 1 日）

> **Note**
>
> 引擎发行版本 1.0.4.2.R2 是 1.0.4.2 中第一个实际发行的版本。

主题

Amazon Neptune 引擎版本 1.0.4.2.R5（2021 年 8 月 16 日）

截至 2021 年 8 月 16 日，引擎版本 1.0.4.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 禁用了引擎版本 `1.0.4.2.R4` 中进行的优化，该优化使 Neptune 查找缓存当引擎在副本上重启后仍然有效。现在，副本重启会清除查找缓存。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.2.R5 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：3.4.10
- SPARQL 版本：1.1

引擎版本 1.0.4.2.R5 的升级路径

如果您正在运行引擎版本 1.0.4.2，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

Amazon Neptune 引擎版本 1.0.4.2.R4（2021 年 7 月 23 日）

截至 2021 年 7 月 23 日，引擎版本 1.0.4.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的改进

- 改进了查找缓存的行为，以避免在副本上运行快速重置后清除冗余缓存。
- 如果服务器上最后一个事件 ID 已经过期，则当通过此事件 ID 请求 AFTER_SEQUENCE_NUMBER 个流时，改进了对流更改日志的处理。如果请求的事件 ID 是服务器上最近清除的事件 ID，则服务器不会再引发事件 ID 过期错误。

在此引擎版本中修复的缺陷

- 修复了 1.0.4.0.R1 中引入的一个错误，即查询不返回大于 760 个字符的全部字符串值。受此错误影响的项目是 RDF 文本和 URI，或 Gremlin ID、键和字符串值。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.2.R4 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：3.4.10
- SPARQL 版本：1.1

引擎版本 1.0.4.2.R4 的升级路径

如果您正在运行引擎版本 1.0.4.2，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

Amazon Neptune 引擎版本 1.0.4.2.R3（2021 年 6 月 28 日）

截至 2021 年 6 月 28 日，引擎版本 1.0.4.2.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的已知问题

问题：

一个 SPARQL 错误，即如果有空格，则无法支持 Accept 标头中的媒体类型。

例如，使用 `-H "Accept: text/csv; q=1.0, */*; q=0.1"` 的查询会返回 JSON 输出而不是 CSV 输出。

解决办法：

如果删除标头的 Accept 子句中的空格，则引擎将以正确的请求格式返回输出。换句话说，不使用 `-H "Accept: text/csv; q=1.0, */*; q=0.1"`，而是使用：

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

在此引擎版本中修复的缺陷

- 修复了快速重置后清除副本上的查找缓存的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.2.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.10`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.2.R3 的升级路径

除非您的数据库集群使用一个或多个 R5d 实例，否则此补丁版本是可选的。如果您的集群有 R5d 实例，它将在下一个维护时段自动升级。否则，它不会自动升级到此补丁版本。

您可以使用 AWS CLI apply-pending-maintenance-action 命令 (ApplyPendingMaintenanceAction API) 手动将 1.0.4.2.R3 版本升级到此 1.0.4.2.R2 版本。

Amazon Neptune 引擎版本 1.0.4.2.R2（2021 年 6 月 1 日）

截至 2021 年 6 月 1 日，引擎版本 1.0.4.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.4.2.R3（2021 年 6 月 28 日）

此引擎版本中的已知问题

问题：

一个 SPARQL 错误，即如果有空格，则无法支持 Accept 标头中的媒体类型。

例如，使用 `-H "Accept: text/csv; q=1.0, */*; q=0.1"` 的查询会返回 JSON 输出而不是 CSV 输出。

解决办法：

如果删除标头的 Accept 子句中的空格，则引擎将以正确的请求格式返回输出。换句话说，不使用 `-H "Accept: text/csv; q=1.0, */*; q=0.1"`，而是使用：

```
-H "Accept: text/csv;q=1.0,*/*;q=0.1"
```

此引擎版本中的新增功能

- 添加了新的 R5d 实例类型，其中包括一个查找缓存，用于在涉及大量属性值或 RDF 文本查找的用例中加快读取速度。请参阅Neptune 查找缓存可以加快读取查询的速度。
- 添加了一个新的实验室模式参数，允许仅在每个查询的基础上使用 useDFE 查询提示调用实验性 DFE 引擎。

此引擎版本中的改进

- 增加了对 TinkerPop 3.4.10 的支持。

- 增加了对在发送 Gremlin 脚本请求时使用 `withStrategies( )` 配置步骤的支持。具体而言，`SubgraphStrategy`、`PartitionStrategy`、`ReadOnlyStrategy`、`EdgeLabelVerification` 和 `ReservedKeysVerificationStrategy` 都受支持。
- 为查询中间的 `V()` 遍历添加了优化。以前，在 Neptune 中未对此类遍历进行优化。
- 增加了对 RFC 2141 URN 的支持，以用作批量加载的 `baseUri` 和 `namedGraphUri` 参数。

在此引擎版本中修复的缺陷

- 修复了解析器中的一个 Gremlin 错误，即错误的查询被视为有效。
- 修复了一个 Gremlin 错误，即使用 `cap().unfold()` 将 `aggregate()` 副作用展开到 `valueMap()` 会引发异常。
- 修复了一个 Gremlin 错误，即 `addV()` 步骤后的某些 `property()` 步骤失败并出现“无法转换为字符串”错误。
- 修复了一个 Gremlin 错误，以防止某些条件插入模式引发并发修改异常。
- 修复了一个 Gremlin 错误，以使查询请求的超时现在无法超过会话超时。
- 修复了一个 SPARQL 错误，也即，当远程服务器不可用时，使用 LOAD 或 UNLOAD 进行更新可能会失败，并显示 HTTP 代码 500 而不是 HTTP 代码 400。
- 修复了一个错误，也即，当使用大于 32 位有符号整数限制 (2,147,483,647) 的 `commitNum` 或 opNum 值时，流 API 调用失败。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.2.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.10`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.2.R2 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.4.2.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the
> instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.2.R1（2021 年 5 月 27 日）

从未部署过引擎版本 1.0.4.2.R1。

Amazon Neptune 引擎版本 1.0.4.1（2020 年 12 月 8 日）

截至 2020 年 12 月 8 日，引擎版本 1.0.4.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.4.1.R1.1（2021 年 3 月 22 日）
- 版本：1.0.4.1.R2（2021 年 2 月 24 日）

> **Important**
>
> 版本：1.0.4.0（2020 年 10 月 12 日）规定所有与 Amazon Neptune 的连接都必须使用 TLS 1.2 和 HTTPS。但是，该版本中的一个错误允许以前设置数据库集群参数以防止强制执行 HTTPS 连接的客户继续使用 HTTP 连接和/或过时的 TLS 连接。
> 该错误已在补丁版本 1.0.4.0.R2 和 1.0.4.1.R2 中修复，但是当自动安装补丁时，此修复导致意外的连接失败。因此，这两个补丁都已恢复，只能手动安装，以便您有机会更新 TLS 1.2 的设置。

> 与 Neptune 的所有连接都必须使用 SSL/TLS 会影响您与 Gremlin 控制台、Gremlin 驱动程序、Gremlin Python、.NET、nodeJs、REST API 的连接以及负载均衡器连接。如果您到目前为止一直使用 HTTP 或较早的 TLS 版本来处理其中任何或所有内容，则必须先更新相关的客户端和驱动程序，并将代码更改为仅使用 HTTPS，然后才能将系统更新到最新补丁。

此引擎版本中的新增功能

- 推出了 Neptune ML 特征，该特征可为 Amazon Neptune 带来强大的机器学习功能。请参阅用于在图形上进行机器学习的 Amazon Neptune ML。
- 添加了用于移除从远程源检索到的数据的自定义 SPARQL UNLOAD 操作。请参阅SPARQL UPDATE UNLOAD。

此引擎版本中的改进

- 优化了一些 Gremlin 条件插入模式，以避免并发修改异常。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，该错误可能会导致使用 `as()` 步骤的特定查询模式丢失结果。
- 修复了一个 Gremlin 错误，该错误可能在使用嵌套在另一个步骤（如 `union()`）中的 `project()` 步骤时导致错误。
- 修复了 `project()` 步骤中的一个 Gremlin 错误。
- 修复了基于字符串的遍历中 `none()` 步骤不起作用的 Gremlin 错误。
- 修复了基于字符串的遍历中的一个 Gremlin 错误，即不支持将空映射作为 `inject()` 步骤的参数。
- 修复了 DFE 引擎中基于字符串的遍历执行中的一个 Gremlin 错误，即终端方法（例如 `toList()`）无法正常工作。
- 修复了一个 Gremlin 错误，该错误导致无法关闭在字符串查询中使用 `iterate()` 步骤的事务。
- 修复了一个 Gremlin 错误，该错误可能会导致使用 `is(P.gte(0))` 模式的查询在某些情况下引发异常。
- 修复了一个 Gremlin 错误，该错误可能会导致使用 `order().by(T.id)` 模式的查询在某些情况下引发异常。
- 修复了一个 Gremlin 错误，该错误可能会导致使用 `addV().aggregate()` 模式的查询在某些情况下给出错误的结果。

- 修复了一个 Gremlin 错误，该错误可能会导致使用 `path()` 步骤后跟 `project()` 步骤模式的查询在某些情况下引发异常。
- 修复了一个 SPARQL 错误，即 SUBSTR 函数发出错误信号而不是返回空字符串。
- 修复了 DFE 引擎中的一个错误，该错误可能导致非阻塞查询计划中的联接操作在存在未绑定变量的情况下生成不正确的结果。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.1 的升级路径

如果您正在运行引擎版本 `1.0.4.1`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.4.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1 ^
```

```
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.1.R1.1（2021 年 3 月 22 日）

截至 2021 年 3 月 22 日，引擎版本 1.0.4.1.R1.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 禁用了对 Gremlin 条件插入模式的优化，该模式可以添加或附加到现有标签和属性。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.1.R1.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.1.R1.1 的升级路径

如果您正在运行引擎版本 `1.0.4.1`，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Amazon Neptune 1.0.4.1.R1.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
```

```
    --engine-version 1.0.4.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.1.R2（2021 年 2 月 24 日）

截至 2021 年 2 月 24 日，引擎版本 1.0.4.1.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.4.1.R2.1（2021 年 3 月 11 日）

此引擎版本中的新增功能

- Neptune 现在支持以 bzip2 格式压缩单个文件以进行批量加载。请参阅加载数据格式。

在此引擎版本中修复的缺陷

- 修复了版本：1.0.4.0（2020 年 10 月 12 日）中的一个错误，该错误允许使用 HTTP 或更早的 TLS 版本而不是 HTTPS 和 TLS 1.2 连接到 Neptune。

  > **Important**
  >
  > 所有与 Neptune 的连接都必须使用 SSL/TLS 可能是一个重大变化。它会影响您与 Gremlin 控制台、Gremlin 驱动程序、Gremlin Python、.NET、nodeJs、REST API 的连接以及负载均衡器连接。如果您到目前为止一直使用 HTTP 或较早的 TLS 版本来处理任何或全部内容，则必须在安装此补丁之前更新相关的客户端和驱动程序，并将代码更改为仅使用 HTTPS。

- 修复了一个 Gremlin 错误，即当 `ConcurrentModificationException` 发生时，在某些情况下，`InternalFailureException` 被设置为响应代码。
- 修复了一个 Gremlin 错误，即在某些条件下，更新边缘或顶点可能会导致临时 `InternalFailureException`。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.1.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：3.4.8
- SPARQL 版本：1.1

引擎版本 1.0.4.1.R2 的升级路径

如果您正在运行引擎版本 1.0.4.1，您的集群将在下一个维护时段内自动升级到此补丁版本。

升级到此版本

Amazon Neptune 1.0.4.1.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.1.R2.1（2021 年 3 月 11 日）

截至 2021 年 3 月 11 日，引擎版本 1.0.4.1.R2.1 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 禁用了对 Gremlin 条件插入模式的优化，该模式可以添加或附加到现有标签和属性。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.1.R2.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：3.4.8
- SPARQL 版本：1.1

引擎版本 1.0.4.1.R2.1 的升级路径

如果您正在运行引擎版本 1.0.4.1.R2，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.4.1.R2.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.1.R2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.1.R2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.0（2020 年 10 月 12 日）

截至 2020 年 10 月 12 日，引擎版本 1.0.4.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.4.0.R2（2021 年 2 月 24 日）

此引擎版本中的新增功能

- 为 Gremlin 添加了帧级压缩。

此引擎版本中的改进

- Amazon Neptune 现在要求所有区域中与 Neptune 的所有连接都使用带有 TLSv1.2 协议的安全套接字层 (SSL)，同时使用以下强密码套件：
  - `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
  - `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
  - `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
  - `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256`
  - `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`
  - `TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA`

  与 Neptune 的 REST 和 WebSocket 连接都是如此，这意味着在所有区域中连接到 Neptune 时，必须使用 HTTPS 而不是 HTTP。

  由于任何地方都将不再支持使用 HTTP 或 TLS 1.1 的客户端连接，因此在升级到此引擎版本之前，请确保您的客户端和代码已更新为使用 TLS 1.2 和 HTTPS。

  > **Important**
  >
  > 所有与 Neptune 的连接都必须使用 SSL/TLS 可能是一个重大变化。它会影响您与 Gremlin 控制台、Gremlin 驱动程序、Gremlin Python、.NET、nodeJs、REST API 的连接以及负载均衡器连接。如果您一直使用 HTTP 进行任何这些操作或全部操作，则现在必须更新相关的客户端和驱动程序，并将代码更改为使用 HTTPS，否则连接将失败。
  > 此版本中的一个错误允许以前设置数据库集群参数以防止强制执行 HTTPS 连接的客户继续使用 HTTP 连接和/或过时的 TLS 连接。该错误已在补丁版本 1.0.4.0.R2 和 1.0.4.1.R2 中修复，但是当自动安装补丁时，此修复导致意外的连接失败。
  > 因此，这两个补丁都已恢复，只能手动安装，以便您有机会更新 TLS 1.2 的设置。

- 已将 TinkerPop 升级到版本 3.4.8。这是向后兼容的升级。有关新增内容，请参阅 TinkerPop 更新日志。

- 改进了 Gremlin `properties()` 步骤的性能。
- 在 explain 和 profile 报告中添加了有关 `BindOp` 和 `MultiplexerOp` 的详细信息。
- 添加了数据预提取功能，以在出现缓存未命中时提高性能。
- 在批量加载程序的 `parserConfiguration` 参数中添加了一个新的 `allowEmptyStrings` 设置，允许在 CSV 加载中将空字符串视为有效的属性值（请参阅Neptune 加载程序请求参数）。
- 加载程序现在允许在多值 CSV 列中使用转义分号。

在此引擎版本中修复的缺陷

- 修复了与 `both()` 步骤相关的潜在 Gremlin 内存泄漏问题。
- 修复了由于未正确处理以 '/' 结尾的端点而导致请求指标丢失的错误。
- 修复了在实验室模式下启用 DFE 引擎时，副本在负载过重时会滞后并重启的错误。
- 修复了由于内存不足状况而导致批量加载失败时无法报告正确的错误消息的错误。
- 修复了在 SPARQL 查询响应中将字符编码放在 Content-Encoding 标头中的 SPARQL 错误。现在 `charset` 改为放在 Content-Type 标头中，使 HTTP 客户端能够识别自动使用的字符集。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.0 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.4.0 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.4.0.R2（2021 年 2 月 24 日）

截至 2021 年 2 月 24 日，引擎版本 1.0.4.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了版本：1.0.4.0（2020 年 10 月 12 日）中的一个错误，该错误允许使用 HTTP 或更早的 TLS 版本而不是 HTTPS 和 TLS 1.2 连接到 Neptune。

  > **Important**
  >
  > 所有与 Neptune 的连接都必须使用 SSL/TLS 可能是一个重大变化。它会影响您与 Gremlin 控制台、Gremlin 驱动程序、Gremlin Python、.NET、nodeJs、REST API 的连接以及负载均衡器连接。如果您到目前为止一直使用 HTTP 或较早的 TLS 版本来处理任何或全部内容，则必须在安装此补丁之前更新相关的客户端和驱动程序，并将代码更改为仅使用 HTTPS。

- 修复了 CSV 批量加载中涉及以 # 结尾的标签的错误。
- 修复了一个 Gremlin 错误，即当 `ConcurrentModificationException` 发生时，在某些情况下，`InternalFailureException` 被设置为响应代码。

- 修复了一个 Gremlin 错误，即在某些条件下，更新边缘或顶点可能会导致临时 `InternalFailureException`。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.4.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.4.0.R2 的升级路径

如果您正在运行引擎版本 `1.0.4.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.4.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.4.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.4.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.3.0（2020 年 8 月 3 日）

截至 2020 年 8 月 3 日，引擎版本 1.0.3.0 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.3.0.R2（2020 年 10 月 12 日）
- 版本：1.0.3.0.R3（2021 年 2 月 19 日）

此引擎版本中的新增功能

- Neptune 推出了一款新的替代查询引擎 (DFE)，可以显著加快查询执行速度。请参阅Amazon Neptune 替代查询引擎 (DFE)。
- DFE 依赖于预生成的有关您的 Neptune 图表数据的统计数据，这些统计数据通过新的统计数据端点进行管理。请参阅DFE 统计数据。
- 现在，您可以通过将新的 `includeQueuedLoads` 参数设置为 FALSE，将排队的加载任务从加载程序获取状态 API 返回的加载 ID 列表中排除。请参阅Neptune 加载程序获取状态请求参数。
- Neptune 现在支持 SPARQL 查询响应的尾随标头，如果请求在开始返回响应块后失败，则这些标头可能包含错误代码和消息。请参阅用于多部分 SPARQL 响应的可选 HTTP 尾随标头。
- Neptune 现在还允许您为 Gremlin 查询启用分块响应编码。与 SPARQL 的情况一样，响应块具有尾随标头，如果查询在开始返回响应块后失败，则这些标头可能包含错误代码和消息。请参阅使用可选的 HTTP 尾随标头启用由多部分组成的 Gremlin 响应。

此引擎版本中的改进

- 现在，您可以向 ElasticSearch 提供批量请求的大小，以便在 Gremlin 中进行全文搜索。
- 改进了 SPARQL GROUP BY 查询的内存使用情况。
- 添加了新的 Gremlin 查询优化器来修剪某些未绑定的筛选条件。
- 将使用 IAM 进行身份验证的 WebSocket 连接可以保持打开状态的最长时间从 36 小时延长到 10 天。

在此引擎版本中修复的缺陷

- 修复了一个错误，即如果您在 POST 请求中发送未编码的 URL 参数，Neptune 会返回 HTTP 状态代码 500 和 `InternalServerErrorException`。现在 Neptune 返回 HTTP 状态代码 400 和 `BadRequestException` 以及消息：`Failure to process the POST request parameters`。
- 修复了未正确报告 WebSocket 连接失败的 Gremlin 错误。
- 修复了一个涉及 sideEffect 消失的 Gremlin 错误。
- 修复了未正确支持全文搜索 `batchsize` 参数的 Gremlin 错误。
- 修复了一个 Gremlin 错误，以单独为 `bothE` 上的每个方向处理 `toV` 和 `fromV`。
- 修复了 `hasLabel` 步骤中涉及 `Edge pathType` 的一个 Gremlin 错误。
- 修复了一个 SPARQL 错误，即使用静态绑定的联接重新排序无法正常工作。
- 修复了一个 SPARQL UPDATE LOAD 错误，即没有正确报告不可用的 Amazon S3 桶。
- 修复了一个 SPARQL 错误，即未正确报告子查询中与 SERVICE 节点相关的一个问题。
- 修复了一个 SPARQL 错误，即包含嵌套 FILTER EXISTS 或 FILTER NOT EXISTS 条件的查询没有得到正确评估。
- 修复了一个 SPARQL 错误，以在通过生成查询调用 SPARQL 服务端点时正确处理生成的重复绑定。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.3.0 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`
- SPARQL 版本：`1.1`

引擎版本 1.0.3.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

如果您集群的 `AutoMinorVersionUpgrade` 参数已设置为 `True`，则在维护时段内，您的集群将在此引擎版本发布后的两到三周内自动升级到此引擎版本。

升级到此版本

Amazon Neptune 1.0.3.0 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.3.0.R3（2021 年 2 月 19 日）

截至 2021 年 2 月 19 日，引擎版本 1.0.3.0.R3 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了 CSV 批量加载中涉及以 # 结尾的标签的错误。
- 修复了一个 Gremlin 错误，该错误可能会导致使用 `as()` 步骤的特定查询模式丢失结果。
- 修复了一个 Gremlin 错误，该错误可能在使用嵌套在另一个步骤（如 `union()`）中的 `project()` 步骤时导致错误。
- 修复了当使用 `toList()` 等终端方法时，实验性 DFE 引擎中字符串遍历执行中的 Gremlin 错误。
- 修复了一个 Gremlin 错误，该错误在字符串查询中使用 `iterate()` 步骤时无法关闭事务。
- 修复了一个 Gremlin 错误，该错误可能导致使用 `is(P.gte(0))` 模式的查询在某些条件下引发异常。

- 修复了一个 Gremlin 错误，即当 `ConcurrentModificationException` 发生时，在某些情况下，`InternalFailureException` 被设置为响应代码。
- 修复了一个 Gremlin 错误，即在某些条件下，更新边缘或顶点可能会导致临时 `InternalFailureException`。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.3.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.3.0.R3 的升级路径

如果您正在运行引擎版本 `1.0.3.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.3.0.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.3.0.R2（2020 年 10 月 12 日）

截至 2020 年 10 月 12 日，引擎版本 1.0.3.0.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的改进

- 改进了 Gremlin `properties()` 步骤的性能。
- 在 explain 和 profile 报告中添加了有关 `BindOp` 和 `MultiplexerOp` 的详细信息。
- 对于 SPARQL 查询响应，已将 `charset` 添加到 Content-Type 标头中，使 HTTP 客户端能够识别自动使用的字符集。

在此引擎版本中修复的缺陷

- 修复了未处理 `CancellationException` 的 SPARQL 错误。
- 修复了包含嵌套选项的查询无法正常工作的 SPARQL 错误。
- 修复了 LOAD 中 `ConcurrentModificationException` 可能导致查询挂起的 SPARQL 错误。
- 修复了一个 SPARQL 错误，该错误导致查询响应无法进行 gzip 压缩。
- 修复了 `groupBy()` 步骤中的一个 Gremlin 错误。
- 修复了与在 `local()` 步骤中使用 `aggregate()` 步骤相关的 Gremlin 错误。
- 修复了与使用 `bothE()` 后跟使用聚合值的谓词有关的 Gremlin 错误。
- 修复了与将 `bothE()` 步骤与 `repeat()` 步骤结合使用相关的 Gremlin 错误。
- 修复了与 `both()` 步骤相关的潜在 Gremlin 内存泄漏问题。
- 修复了由于未正确处理以 '/' 结尾的端点而导致请求指标丢失的错误。
- 修复了在请求队列未满时可能引发 `ThrottlingException` 的错误。
- 修复了加载由于 `LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE` 等原因加载失败时获取加载状态的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.3.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`

- SPARQL 版本：1.1

引擎版本 1.0.3.0.R2 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

如果您集群的 AutoMinorVersionUpgrade 参数已设置为 True，则在维护时段内，您的集群将在此引擎版本发布后的两到三周内自动升级到此引擎版本。

升级到此版本

Amazon Neptune 1.0.3.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.3.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.3.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.2（2020 年 3 月 9 日）

截至 2020 年 3 月 9 日，引擎版本 1.0.2.2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此版本的后续补丁版本

- 版本：1.0.2.2.R2（2020 年 4 月 2 日）

- 版本：1.0.2.2.R3（2020 年 7 月 22 日）
- 版本：1.0.2.2.R4（2020 年 7 月 23 日）
- 版本：1.0.2.2.R5（2020 年 10 月 12 日）
- 版本：1.0.2.2.R6（2021 年 2 月 19 日）

此引擎版本中的改进

- 已将有关正在回滚的事务的信息添加到状态 API。请参阅实例状态。
- 已将 Apache TinkerPop 的版本升级到 3.4.3。

  版本 3.4.3 向后兼容 Neptune 所支持的早期版本 (3.4.1)。它确实引入了一个小的行为更改：当您尝试关闭一个不存在的会话时，Gremlin 不再返回错误（请参阅防止在关闭不存在的会话时显示错误）。
- 消除了执行 Gremlin 全文搜索步骤时的性能瓶颈。

在此引擎版本中修复的缺陷

- 修复了处理查询中的空图表模式时出现的 SPARQL 错误。
- 修复了处理 URL 编码的查询中未编码分号时出现的 SPARQL 错误。
- 修复了处理 `Union` 步骤中的重复顶点时出现的 Gremlin 错误。
- 修复了一个 Gremlin 错误，该错误导致一些其 `.repeat()` 包含 `.simplePath()` 或 `.cyclicPath()` 的查询返回错误结果。
- 修复了一个 Gremlin 错误，该错误导致 `.project()` 在其子遍历未返回解决方案的情况下返回错误结果。
- 修复了一个 Gremlin 错误，其中读写冲突产生的错误引发了 `InternalFailureException` 而非 `ConcurrentModificationException`。
- 修复了一个导致 `.group().by(...).by(values("property"))` 失败的 Gremlin 错误。
- 修复了全文搜索步骤的配置文件输出中的 Gremlin 错误。
- 解决了 Gremlin 会话中的资源泄漏问题。
- 修复了一个错误，该错误导致状态 API 在某些情况下无法报告正确的可排序版本。
- 修复了一个批量加载程序错误，该错误导致指向 Amazon S3 之外的某个位置的 URL 可被用作批量加载请求中的源。

- 修复了详细加载状态中的一个批量加载程序错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.2 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

如果您集群的 `AutoMinorVersionUpgrade` 参数已设置为 `True`，则在维护时段内，您的集群将在此引擎版本发布后的两到三周内自动升级到此引擎版本。

升级到此版本

Amazon Neptune 1.0.2.2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.2.R6（2021 年 2 月 19 日）

截至 2021 年 2 月 19 日，引擎版本 1.0.2.2.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin 错误，即当 `ConcurrentModificationException` 发生时，在某些情况下，`InternalFailureException` 被设置为响应代码。
- 修复了一个 Gremlin 错误，即在某些条件下，更新边缘或顶点可能会导致临时 `InternalFailureException`。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.2.R6 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.8`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.2.R6 的升级路径

如果您正在运行引擎版本 `1.0.2.2`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.2.R6 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
> ```

```
    proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.2.R5（2020 年 10 月 12 日）

截至 2020 年 10 月 12 日，引擎版本 1.0.2.2.R5 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的改进

- 改进了 Gremlin `properties()` 步骤的性能。
- 在 explain 和 profile 报告中添加了有关 `BindOp` 和 `MultiplexerOp` 的详细信息。
- 对于 SPARQL 查询响应，已将 `charset` 添加到 Content-Type 标头中，使 HTTP 客户端能够识别自动使用的字符集。

在此引擎版本中修复的缺陷

- 修复了未处理 `CancellationException` 的 SPARQL 错误。
- 修复了包含嵌套选项的查询无法正常工作的 SPARQL 错误。
- 修复了 LOAD 中 `ConcurrentModificationException` 可能导致查询挂起的 SPARQL 错误。
- 修复了一个 SPARQL 错误，该错误导致查询响应无法进行 gzip 压缩。
- 修复了 `groupBy()` 步骤中的一个 Gremlin 错误。
- 修复了与在 `local()` 步骤中使用 `aggregate()` 步骤相关的 Gremlin 错误。
- 修复了与使用 `bothE()` 后跟使用聚合值的谓词有关的 Gremlin 错误。
- 修复了与将 `bothE()` 步骤与 `repeat()` 步骤结合使用相关的 Gremlin 错误。
- 修复了与 `both()` 步骤相关的潜在 Gremlin 内存泄漏问题。
- 修复了由于未正确处理以 '/' 结尾的端点而导致请求指标丢失的错误。
- 修复了在请求队列未满时可能引发 `ThrottlingException` 的错误。
- 修复了加载由于 `LOAD_DATA_FAILED_DUE_TO_FEED_MODIFIED_OR_DELETE` 等原因加载失败时获取加载状态的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.2.R5 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.2.R5 的升级路径

如果您正在运行引擎版本 `1.0.2.2`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.2.R5 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.2.R4（2020 年 7 月 23 日）

截至 2020 年 7 月 23 日，引擎版本 1.0.2.2.R4 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的改进

- 通过更频繁地将未使用的内存释放回操作系统来提高内存使用率。
- 还改进了 SPARQL GROUP BY 查询的内存使用率。
- 将使用 IAM 进行身份验证的 WebSocket 连接可以保持打开状态的最长时间从 36 小时延长到 10 天。
- 添加了 `BufferCacheHitRatio` CloudWatch 指标，该指标可用于诊断查询延迟和调整实例类型。请参阅Neptune 指标。

在此引擎版本中修复的缺陷

- 修复了关闭空闲或过期的 IAM WebSocket 连接时出现的错误。现在，Neptune 会在关闭连接之前发送一个关闭帧。
- 修复了在评估包含嵌套 FILTER EXISTS 和/或 FILTER NOT EXISTS 条件的查询时出现的 SPARQL 错误。
- 修复了 SPARQL 查询终止错误，该错误会在某些极端条件下导致服务器上的线程被阻止。
- 修复了 `hasLabel` 步骤中涉及 Edge pathType 的一个 Gremlin 错误。
- 修复了一个 Gremlin 错误，以单独为 `bothE` 上的每个方向处理 `toV` 和 `fromV`。
- 修复了一个涉及 sideEffect 消失的 Gremlin 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.2.R4 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.2.R4 的升级路径

如果您正在运行引擎版本 `1.0.2.2`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.2.R4 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.2.R3（2020 年 7 月 22 日）

引擎版本 1.0.2.2.R3 已合并到引擎版本 1.0.2.2.R4 中。

Amazon Neptune 引擎版本 1.0.2.2.R2（2020 年 4 月 2 日）

截至 2020 年 4 月 2 日，引擎版本 1.0.2.2.R2 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

此引擎版本中的改进

- 现在，您最多可以对 64 个批量加载作业进行排队，而不必等待一个作业完成后才启动下一个作业。还可以使用 `load` 命令的 `dependencies` 参数，使得某个排队加载请求的执行取决于成功地完成一个或多个先前排队的加载作业。请参阅Neptune 加载程序命令。
- 现在可以对全文搜索输出进行排序（请参阅全文搜索参数）。
- 现在有一个用于调用 Neptune 流的数据库集群参数，并且此特征已移出实验室模式。请参阅启用 Neptune Streams。

在此引擎版本中修复的缺陷

- 修复了服务器启动延迟了实例创建的随机故障。

- 修复了一个优化程序问题：也即，查询中的 BIND 语句使优化程序在联合顺序规划中以非选择性模式启动。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.2.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.3`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.2.R2 的升级路径

如果您正在运行引擎版本 `1.0.2.2`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.2.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.2 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.2 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.1（2019 年 11 月 22 日）

此版本的后续补丁版本

- 版本：1.0.2.1.R6（2020 年 4 月 22 日）
- 版本：1.0.2.1.R5（2020 年 4 月 22 日）此修补程序版本未部署。
- 版本：1.0.2.1.R4（2019 年 12 月 20 日）
- 版本：1.0.2.1.R3（2019 年 12 月 12 日）
- 版本：1.0.2.1.R2（2019 年 11 月 25 日）

此引擎版本中的新增功能

- 通过与 Amazon OpenSearch Service 的集成，增加了全文搜索功能。请参阅 Neptune 全文搜索。
- 添加了使用实验室模式为大量谓词创建第四个索引（OSGP 索引）的选项。请参阅OSGP 索引。
- 向 SPARQL Explain 增加了详细信息 模式。有关详细信息，请参阅 使用 SPARQL `explain` 和 详细信息模式输出。
- 将实验模式信息添加到引擎状态报告中。有关详细信息，请参阅实例状态。
- 数据库集群快照现在可以跨 AWS 区域复制。请参阅复制快照。

此引擎版本中的改进

- 提高了处理大量谓词时的性能。
- 增强了查询优化。虽然这对客户来说应该完全透明，但我们建议您在升级之前测试您的应用程序，以确保它们按预期运行。
- 对错误报告进行了小的增强。
- 添加了对 Gremlin `.project()` 和 `.identity()` 步骤的优化。
- 添加了对非终端 Gremlin `.union()` 案例的优化。
- 添加了对 Gremlin `.path().by()` 遍历的原生支持。
- 添加了对 Gremlin `.coalesce()` 的原生支持。
- 进一步优化了批量写入。
- 我们现在要求 HTTPS 连接至少使用 TLS 版本 1.2 或更高版本，以防止使用过时/不安全的密码。

在此引擎版本中修复的缺陷

- 修复了一个 Gremlin `addE()` 内部遍历处理错误。
- 修复了由于 AST 注释从子遍历泄漏到父级导致的 Gemlin 错误。
- 修复了在 `select()` 之后调用 `.otherV()` 时 Gremlin 中出现的错误。
- 修复了一个 Gremlin 错误，该错误导致了某些 `.hasLabel()` 步骤在 `bothE()` 步骤执行后失败。
- 对 Gremlin .sum() 和 .project() 进行了小的修复。
- 修复了在处理缺少右括号的 SPARQL 查询时的错误。
- 修复了 SPARQL Explain 中的一些小错误。
- 修复了处理并发获取加载状态请求的错误。
- 减少了执行一些带有 `.project()` 步骤的 Grenolin 遍历所用的内存。
- 修复了 SPARQL 中特殊值的数值比较。请参阅标准合规性。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.1 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.1 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
```

```
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：

```
  We're sorry, your request to modify DB cluster (cluster identifier) has
failed.
  Cannot modify engine version because instance (instance identifier) is
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.1.R6（2020 年 4 月 22 日）

截至 2020 年 4 月 22 日，引擎版本 1.0.2.1.R6 正在普遍部署中。请注意，新版本在每个区域的发布需要几天的时间。

在此引擎版本中修复的缺陷

- 修复了一个错误，其中 `ConcurrentModificationConflictException` 和 `TransactionException` 未转换为 `NeptuneGremlinException`，导致向客户返回了 `InternalFailureException`。
- 修复了 Neptune 在服务器完全准备就绪之前报告其状态为正常的错误。
- 修复了在有两个 `value->id` 映射同时插入时字典和用户事务提交顺序混乱的错误。
- 修复了加载状态序列化中的一个错误。
- 修正了一个 Gremlin 会话错误。
- 修复了在服务器无法启动时 Neptune 未能引发异常的错误。
- 修复了 Neptune 无法在关闭通道之前发送 WebSocket 关闭帧的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.1.R6 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.1.R6 的升级路径

如果您正在运行引擎版本 `1.0.2.1`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.1.R6 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.1.R5（2020 年 4 月 22 日）

从未部署过引擎版本 1.0.2.1.R5。

Amazon Neptune 引擎版本 1.0.2.1.R4（2019 年 12 月 20 日）

此引擎版本中的改进

- Neptune 现在总是尝试首先在执行管道中放置任何全文搜索调用。这可以减少对 OpenSearch 的调用量，从而显著提高性能。请参阅全文搜索查询执行。

- 如果您尝试访问不存在的属性、顶点或边缘，Neptune 现在会引发 `IllegalArgumentException`。以前，Neptune 在这种情况下会引发 `UnsupportedOperationException`。

  例如，如果您尝试添加引用不存在顶点的边缘，则现在将引发 `IllegalArgumentException`。

在此引擎版本中修复的缺陷

- 修复了一个 Grenomlin 错误，即 `union` 在 `project-by` 内遍历不会返回结果或者返回错误的结果。
- 修复了导致嵌套的 `.project().by()` 步骤返回错误结果的 Gemlin 错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.1.R4 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.1.R4 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

但是，不支持自动更新到此版本。

升级到此版本

Amazon Neptune 1.0.2.1.R4 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
> ```

```
  running on an old configuration. Apply any pending maintenance actions on the
instance before
  proceeding with the upgrade.
```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.1.R3（2019 年 12 月 12 日）

在此引擎版本中修复的缺陷

- 修复了一个错误，该错误的表现为：即使使用 `neptune_lab_mode` 参数中的 `ObjectIndex` 值正确启用了实验室模式中的功能，也禁用 OSGP 索引。
- 修复了一个错误，该错误会影响在 `.project().by()` 步骤内带有 `.fold()` 的 Gremlin 查询。例如，它导致以下查询返回不完整的结果：

```
g.V().project("a").by(valueMap().fold())
```

- 修复了 RDF 数据批量加载中的性能瓶颈。
- 修复了在启用流并在主服务器之前重启副本时导致副本崩溃的错误。
- 修复了实例上轮换的 SSL 证书在没有重新启动实例的情况下不会被选取的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.1.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.1.R3 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

但是，不支持自动更新到此版本。

升级到此版本

Amazon Neptune 1.0.2.1.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.1.R2（2019 年 11 月 25 日）

在此引擎版本中修复的缺陷

- 修复了影响所有具有非循环 by-traversals 和非 `path()` by-traversals 的 `project().by()` 查询的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.1.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.1.R2 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

但是，不支持自动更新到此版本。

升级到此版本

Amazon Neptune 1.0.2.1.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.1 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.1 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has failed.
> Cannot modify engine version because instance (instance identifier) is
> running on an old configuration. Apply any pending maintenance actions on the instance before
> proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.0（2019 年 11 月 8 日）

重要：此引擎版本现在已弃用

从 2020 年 5 月 19 日开始，将不会创建使用该引擎版本的新实例。

此引擎版本现已被版本 1.0.2.1 取代，后者包含此版本中的所有错误修复以及其它特征，例如全文搜索集成、OSGP 索引支持和跨 AWS 区域的数据库快照集群副本。

自 2020 年 6 月 1 日起，Neptune 将在下一个维护时段内自动将运行此引擎版本的任何集群升级到版本 1.0.2.1 的最新补丁。您可以在此之前手动升级，如此处所述。

如果您在升级过程中遇到任何问题，请通过 AWS Support 或 AWS 开发人员论坛与我们联系。

此版本的后续补丁版本

- 版本：1.0.2.0.R3（2020 年 5 月 5 日）

- 版本：1.0.2.0.R2（2019 年 11 月 21 日）

此引擎版本中的新增功能

除了维护更新之外，此版本还添加了新功能用于一次支持多个引擎版本（请参阅 维护 Amazon Neptune 数据库集群）。

因此，引擎版本的编号发生了变化（请参阅 引擎版本 1.3.0.0 之前的版本编号）。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.0 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.0 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

您不会自动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.0 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
```

```
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```

如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.0.R3（2020 年 5 月 5 日）

重要：此引擎版本现在已弃用

从 2020 年 5 月 19 日开始，将不会创建使用该引擎版本的新实例。

此引擎版本现已被版本 1.0.2.1 取代，后者包含此版本中的所有错误修复以及其它特征，例如全文搜索集成、OSGP 索引支持和跨 AWS 区域的数据库快照集群副本。

自 2020 年 6 月 1 日起，Neptune 将在下一个维护时段内自动将运行此引擎版本的任何集群升级到版本 1.0.2.1 的最新补丁。您可以在此之前手动升级，如此处所述。

如果您在升级过程中遇到任何问题，请通过 AWS Support 或 AWS 开发人员论坛与我们联系。

在此引擎版本中修复的缺陷

- 修复了将 `ConcurrentModificationConflictException` 和 `TransactionException` 报告为一般 `InternalFailureException` 的错误。
- 修复了运行状况检查中导致服务器在启动期间频繁重新启动的错误。
- 修复了由于特定条件下提交顺序混乱导致数据在副本上不可见的错误。
- 修复了加载状态序列化中的一个错误，即加载由于缺少 Amazon S3 访问权限而失败。
- 解决了 Gremlin 会话中的资源泄漏问题。
- 修复了运行状况检查中的一个错误，即隐藏管理 IAM 身份验证的组件启动时的不正常状态。
- 修复了 Neptune 无法在关闭通道之前发送 WebSocket 关闭帧的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.0.R3 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.0.R3 的升级路径

如果您正在运行引擎版本 `1.0.2.0`，您的集群将在下一个维护时段内自动升级到此补丁版本。

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

升级到此版本

Amazon Neptune 1.0.2.0.R3 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>  Cannot modify engine version because instance (instance identifier) is
>  running on an old configuration. Apply any pending maintenance actions on the
> instance before
>  proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.2.0.R2（2019 年 11 月 21 日）

重要：此引擎版本现在已弃用

从 2020 年 5 月 19 日开始，将不会创建使用该引擎版本的新实例。

此引擎版本现已被版本 1.0.2.1 取代，后者包含此版本中的所有错误修复以及其它特征，例如全文搜索集成、OSGP 索引支持和跨 AWS 区域的数据库快照集群副本。

自 2020 年 6 月 1 日起，Neptune 将在下一个维护时段内自动将运行此引擎版本的任何集群升级到版本 1.0.2.1 的最新补丁。您可以在此之前手动升级，如此处所述。

如果您在升级过程中遇到任何问题，请通过 AWS Support 或 AWS 开发人员论坛与我们联系。

在此引擎版本中修复的缺陷

- 改进了服务器上脏页面的缓存策略，使得服务器在进入内存不足状态时，`FreeableMemory` 可以更快地进行恢复。
- 修正了服务器上在处理多个并发加载状态和/或启动加载请求时，可能会导致竞争条件和崩溃的错误。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.2.0.R2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

引擎版本 1.0.2.0.R2 的升级路径

您可以将任何以前的 Neptune 引擎版本手动升级到此版本。

但是，不支持自动更新到此版本。

升级到此版本

Amazon Neptune 1.0.2.0.R2 现已正式发布。

如果数据库集群运行的引擎版本有此版本的升级路径，则可以立即对其进行升级。您可以使用控制台上的数据库集群操作或使用 SDK 升级任何符合条件的集群。以下 CLI 命令将立即升级符合条件的集群：

对于 Linux、OS X 或 Unix：

```
aws neptune modify-db-cluster \
    --db-cluster-identifier (your-neptune-cluster) \
    --engine-version 1.0.2.0 \
    --apply-immediately
```

对于 Windows：

```
aws neptune modify-db-cluster ^
    --db-cluster-identifier (your-neptune-cluster) ^
    --engine-version 1.0.2.0 ^
    --apply-immediately
```

更新将同时应用于数据库集群中的所有实例。更新操作要求在所有这些实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以重新使用数据库集群。

升级前始终先测试

发布新的主要或次要 Neptune 引擎版本时，请务必先在该版本上测试您的 Neptune 应用程序，然后再升级到该版本。即使是次要版本升级，也可能引入会影响代码的新特征或行为。

首先，将当前版本的发行说明页面与目标版本的发行说明页面进行比较，以查看查询语言版本是否会发生变化或是否会发生其它重大更改。

在升级生产数据库集群之前测试新版本的最佳方法是克隆生产集群，以便克隆运行新的引擎版本。然后，您可以在不影响生产数据库集群的情况下在克隆上运行查询。

请在升级之前始终创建手动快照

在执行升级之前，我们强烈建议您始终创建数据库集群的手动快照。拥有自动快照只能提供短期保护，而手动快照在您显式删除它之前仍然可用。

在某些情况下，作为升级过程的一部分，Neptune 会为您创建手动快照，但您不应依赖此快照，无论如何都应创建自己的手动快照。

当您确定不需要将数据库集群恢复到其升级前的状态时，可以显式删除自己创建的手动快照以及 Neptune 可能已创建的手动快照。如果 Neptune 创建手动快照，则其名称将以 `preupgrade` 开头，后跟数据库集群的名称、源引擎版本、目标引擎版本和日期。

> **Note**
>
> 如果您在待处理操作正在进行时尝试升级，则可能会遇到如下错误：
>
> ```
> We're sorry, your request to modify DB cluster (cluster identifier) has
> failed.
>   Cannot modify engine version because instance (instance identifier) is
>   running on an old configuration. Apply any pending maintenance actions on the
> instance before
>   proceeding with the upgrade.
> ```
>
> 如果遇到此错误，请等待待处理操作完成，或者立即触发维护时段，让之前的升级完成。

有关升级引擎版本的更多信息，请参阅维护 Amazon Neptune 数据库集群。如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

Amazon Neptune 引擎版本 1.0.1.2（2020 年 6 月 10 日）

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用该引擎版本的新实例。

此引擎版本中的改进

- 如果您尝试访问不存在的属性、顶点或边缘，Neptune 现在会引发 `IllegalArgumentException`。以前，Neptune 在这种情况下会引发 `UnsupportedOperationException`。

  例如，如果您尝试添加引用不存在顶点的边缘，则现在将引发 `IllegalArgumentException`。

在此引擎版本中修复的缺陷

- 修复了在有两个 `value->id` 映射同时插入时字典和用户事务提交顺序混乱的错误。
- 修复了加载状态序列化中的一个错误。
- 修复了服务器启动延迟了实例创建的随机故障。
- 修复了游标泄漏。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.1.2 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.4.1`
- SPARQL 版本：`1.1`

Amazon Neptune 引擎版本 1.0.1.1（2020 年 6 月 26 日）

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用该引擎版本的新实例。

在此引擎版本中修复的缺陷

- 修复了一个导致在并发插入时提交乱序的错误。

- 修复了加载状态序列化中的一个错误。
- 修复了服务器启动延迟了实例创建的随机故障。
- 修复了内存泄漏问题。

此版本支持的查询语言版本

在将数据库集群升级到版本 1.0.1.1 之前，请确保您的项目与以下查询语言版本兼容：

- Gremlin 版本：`3.3.2`
- SPARQL 版本：`1.1`

Amazon Neptune 引擎版本 1.0.1.0（2019 年 7 月 2 日）

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用该引擎版本的新实例。

Amazon Neptune 引擎更新 2019 年 10 月 31 日

版本：1.0.1.0.200502.0

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用此引擎版本的任何新实例。

在此引擎版本中修复的缺陷

- 修复了当客户端使用 `traversal().withRemote(...)`（换句话说，使用 GLV 字节码）连接到 Neptune 时，`tree()` 步骤的响应的序列化中的 Gremlin 错误。

  此版本解决了以下问题：使用 `traversal().withRemote(...)` 连接到 Neptune 的客户端收到对包含 `tree()` 步骤的 Gremlin 查询的无效响应。
- 修复了 `DELETE WHERE LIMIT` 查询中的 SPARQL 错误，其中，由于竞争条件，查询终止过程将挂起，从而导致查询超时。

Amazon Neptune 引擎更新 2019 年 10 月 15 日

版本：1.0.1.0.200463.0

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用此引擎版本的任何新实例。

此引擎版本中的新增功能

- 添加了 Gremlin Explain/Profile 功能（请参阅 使用 Gremlin `explain` 分析 Neptune 查询执行）。
- 添加了 对基于 Gremlin 脚本的会话的支持，以实现在单个事务中执行多个 Gremlin 遍历。
- 在 Neptune 中添加了对 SPARQL 联合查询扩展的支持（请参阅 SPARQL 1.1 联合查询和 Neptune 中使用 `SERVICE` 扩展的 SPARQL 联合查询）。
- 添加了让您能够通过 HTTP URL 参数或通过 SPARQL `queryId` 查询提示将自己的 `queryId` 注入到 Gremlin 或 SPARQL 查询中的功能（请参阅 将自定义 ID 注入到 Neptune Gremlin 或 SPARQL 查询中）。
- 在 Neptune 中添加了 实验室模式 特征，让您能够试用即将发布但尚未准备好在生产环境中使用的特征。
- 添加了即将发布的 Neptune Streams 功能，该功能可将对数据库所做的全部更改可靠地记录到流（保留一周）中。该功能仅在实验室模式下可用。
- 更新了并发事务的形式语义（请参阅 Neptune 中的事务语义）。该功能为并发提供了行业标准保证。

  默认情况下，将启用这些事务语义。在某些情况下，该功能可能会更改当前的加载行为和降低加载性能。可以使用数据库集群 `neptune_lab_mode` 参数（通过在参数值中包含 `ReadWriteConflictDetection=disabled`）恢复为之前的语义。

此引擎版本中的改进

- 通过报告引擎使用的 TinkerPop 版本和 SPARQL 版本，改进了 实例状态 API。
- 改进了 Gremlin 子图运算符的性能。
- 改进了 Gremlin 响应序列化的性能。
- 改进了 Gremlin Union 步骤的性能。
- 缩短了简单 SPARQL 查询的延迟。

在此引擎版本中修复的缺陷

- 修复了将超时错误地作为内部故障返回的 Gremlin 错误。

- 修复了对部分变量集进行 ORDER BY 导致内部服务器错误的 SPARQL 错误。

Amazon Neptune 引擎更新 2019 年 9 月 19 日

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用此引擎版本的任何新实例。

版本：1.0.1.0.200457.0

Amazon Neptune 1.0.1.0.200457.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200457.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令升级数据库集群：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

在此引擎版本中修复的缺陷

- 通过删除联接谓词处理的性能改进，修复了之前引擎版本 (1.0.1.0.200369.0) 引入的 Gremlin 正确性问题。
- 修复了导致使用 `DISTINCT` 的查询和将单个模式封装到 `OPTIONAL` 中会生成 `InternalServerError` 的 SPARQL 错误。

Amazon Neptune 引擎更新 2019 年 8 月 13 日

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用该引擎版本的新实例。

此引擎版本中的新增功能

- 为 Neptune 加载程序命令 的 `parallelism` 参数添加了 `OVERSUBSCRIBE` 选项，这会导致 Neptune 批量加载程序使用所有可用的线程和资源。

此引擎版本中的改进

- 改进了包含简单逻辑 OR 表达式的 SPARQL 筛选条件的性能。
- 改进了处理连接性谓词的 Gremlin 性能。

在此引擎版本中修复的缺陷

- 修复了阻止从 `xsd:date` 中减去 `xsd:duration` 的 SPARQL 错误。
- 修复了导致在存在 `UNION` 时静态内联出现不完整结果的 SPARQL 错误。
- 修复了查询取消中的 SPARQL 错误。
- 修复了在类型提升期间导致溢出的 Gremlin 错误。
- 修复了 `addE().from().to()` 步骤中处理顶点元素的 Gremlin 错误。
- 修复了涉及在单基数插入中处理 NaN 双精度和浮点数的 Gremlin 错误（2019 年 7 月 26 日在引擎版本 1.0.1.0.200366.0 中发布）。
- 修复了生成查询计划的错误（涉及基于属性的搜索）。

Amazon Neptune 引擎更新 2019 年 7 月 26 日

版本：1.0.1.0.200366.0

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用此引擎版本的任何新实例。

此引擎版本中的新增功能

- 已升级到 TinkerPop 3.4.1（请参阅 TinkerPop 升级信息和 TinkerPop 3.4.1 更改日志）。

  对于 Neptune 客户而言，这些更改提供了新的功能和改进，例如：

  - `GraphBinary` 现在可以以序列化格式提供。

- 在 TinkerPop Java 驱动程序中导致内存泄漏的保持活动错误已得到修复，因此不再需要解决方法。

但是，在少数情况下，它们可能会影响 Neptune 中现有的 Gremlin 代码。例如：

- `valueMap()` 现在会返回 `Map<Object,Object>` 而不是 `Map<String,Object>`。
- `within()` 步骤的不一致行为得到修复，因此它将与其他步骤保持一致。以前，类型必须匹配，比较才能正常运行。现在，可以准确地比较不同类型的数量。例如，`33` 现在比较为等于 `33L`，之前不是这样。
- `ReducingBarrierStep` 中的错误得到修复，因此如果没有可用于输出的元素，它现在不返回任何值。
- `select()` 范围的顺序已更改（顺序现在为 `maps`、`side-effects`、`paths`）。这会更改罕见查询的结果，这些查询将 `side-effects` 和 `select` 与和 `select` 具有相同键名称的 `side-effects` 进行组合。
- `bulkSet()` 现在是 GraphSON 协议的一部分。以 `toBulkSet()` 结尾的查询将不适用于较旧的客户端。
- 从3.4 客户端删除了 `Submit()` 步骤的一个参数化。

TinkerPop 3.4 中引入的许多其它更改不会影响当前的 Neptune 行为。例如，Gremlin `io()` 作为 `Traversal` 的一个步骤添加，现在已在 `Graph` 中弃用，但从未在 Neptune 中启用。

- 为 Gremlin 批量加载程序增加了对单基数顶点属性的支持，用于加载属性图数据。
- 添加了一个选项，用于覆盖批量加载程序中单基数属性的现有值。
- 添加了检索 Gremlin 查询状态和取消 Gremlin 查询的功能。
- 为 SPARQL 查询超时添加了查询提示。
- 添加了功能，用于在状态 API 中查看实例角色（请参阅实例状态）。
- 添加了对数据库克隆的支持（请参阅 Neptune 中的数据库克隆）。

此引擎版本中的改进

- 改进了 SPARQL 查询说明，用于显示来自 FROM 子句的图形变量。
- 改进了筛选条件、等于筛选条件、VALUES 子句和范围计数中的 SPARQL 性能。
- 改进了 Gremlin 步骤排序的性能。
- 改进了 Gremlin `.repeat.dedup` 遍历的性能。
- 改进了 Gremlin `valueMap()` 和 `path().by()` 遍历的性能。

在此引擎版本中修复的缺陷

- 修复了 SPARQL 属性路径的多个问题，包括对命名图形的操作。
- 修复了导致内存问题的 SPARQL CONSTRUCT 查询问题。
- 修复了 RDF Turtle 解析器和本地名称的问题。
- 修复了向用户显示的更正错误消息的问题。
- 修复了 Gremlin `repeat()...drop()` 遍历的问题。
- 修复了 Gremlin `drop()` 步骤的问题。
- 修复了 Gremlin 标签筛选的问题。
- 修复了 Gremlin 查询超时的问题。

Amazon Neptune 引擎更新 2019 年 7 月 2 日

重要：此引擎版本现在已弃用

从 2021 年 4 月 27 日开始，将不会创建使用此引擎版本的任何新实例。

在此引擎版本中修复的缺陷

- 修复了一个错误，该错误会导致绑定了属性名称和值的特定模式不进行优化。

较早的 Neptune 引擎版本

主题

Amazon Neptune 引擎更新 2019 年 6 月 12 日

版本：1.0.1.0.200310.0

Amazon Neptune 1.0.1.0.200310.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200310.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200310.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 修复了并发插入和删除边缘会导致多个边缘具有相同 ID 的错误。
- 其他少量修复和改进。

Amazon Neptune 引擎更新 2019 年 5 月 1 日

版本：1.0.1.0.200296.0

Amazon Neptune 1.0.1.0.200296.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200296.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200296.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 向 Neptune SPARQL 查询增加了新的 `explain` 特征，可帮助您可视化查询计划并采取优化计划的步骤（如有必要）。有关信息，请参阅 SPARQL `explain`。
- 以多种方式改进了 SPARQL 性能和报告。
- 以多种方式改进了 Gremlin 性能和行为。
- 改进了长时间运行的 `drop( )` 查询的超时。
- 改进了 `otherV( )` 查询的性能。
- 向您在查询数据库集群或实例的 Neptune 运行状况时返回的信息增加了两个字段，即引擎版本号和集群或实例启动时间。请参阅实例状态。
- Neptune 加载程序 `Get-Status` API 现在返回记录加载任务开始时间的 `startTime` 字段。
- 加载程序命令现在采用可选的 `parallelism` 参数，让您可以限制加载程序使用的线程数。

Amazon Neptune 引擎更新 2019 年 1 月 21 日

版本：1.0.1.0.200267.0

Amazon Neptune 1.0.1.0.200267.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200267.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200267.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- Neptune 等待任何冲突得到解决的时间较长（在指定的查询超时内）。这减少了需要由客户端处理的并行修改异常的数量（请参阅查询错误）。
- 修复了实施 Gremlin 基数有时导致引擎重新启动的问题。
- 改进了 Gremlin 对于`emit.times` 重复查询的性能。
- 修复了一个 Gremlin 问题，其中，`repeat.until` 允许应已被筛选出来的 `.emit` 解决方案。
- 改进了 Gremlin 中的错误处理。

Amazon Neptune 引擎更新 2018 年 11 月 19 日

版本：1.0.1.0.200264.0

Amazon Neptune 1.0.1.0.200264.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200264.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200264.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 添加了对 the section called “查询提示” 的支持。
- 改进了 IAM 身份验证的错误消息。有关更多信息，请参阅the section called “IAM 错误”。
- 改进了具有大量谓词的 SPARQL 查询性能。
- 改进了 SPARQL 属性路径性能。
- 改进了在与 `addV()`、`addE()` 和 `property()` 步骤一起使用时条件更改的 Gremlin 性能，如 `fold().coalesce(unfold(), …)` 模式。
- 改进了 `by()` 和 `sack()` 调制的 Gremlin 性能。
- 改进了 `group()` 和 `groupCount()` 步骤的 Gremlin 性能。
- 改进了 `store()`、`sideEffect()` 和 `cap().unfold()`步骤的 Gremlin 性能。
- 改进了对 Gremlin 单一基数属性约束的支持。
  - 改进了对标记为单一基数属性的边缘属性和顶点属性的单一基数实施。

- 引入了在 Neptune 负载任务期间为现有边缘属性指定其他属性值时出现的错误。

Amazon Neptune 引擎更新 2018 年 11 月 8 日

版本：1.0.1.0.200258.0

Amazon Neptune 1.0.1.0.200258.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200258.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200258.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 添加了对 SPARQL 查询提示 的支持。
- 改进了 SPARQL FILTER (NOT) Exists 查询的性能。
- 改进了 SPARQL DESCRIBE 查询的性能。
- 改进了 Gremlin 中重复直至模式的性能。
- 改进了在 Gremlin 中添加边缘的性能。
- 修复了导致 SPARQL Update DELETE 查询在某些情况下失败的问题。

- 修复了使用 Gremlin WebSocket 服务器处理超时的问题。

Amazon Neptune 引擎更新 2018 年 10 月 29 日

版本：1.0.1.0.200255.0

Amazon Neptune 1.0.1.0.200255.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200255.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200255.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 已将 IAM 身份验证信息添加到审计日志中。
- 添加了对使用 IAM 角色和实例配置文件的临时凭证的支持。
- 添加了在撤消权限或删除 IAM 用户或角色时对为 IAM 身份验证的 WebSocket 连接终止。
- 将每实例的最大 WebSocket 连接数限制为 60000。
- 改进了更小实例类型的批量负载性能。
- 改进了在 Gremlin 中包含 `and()`、`or()`、`not()`、`drop()` 运算符的查询的性能。

- NTriples 解析程序当前拒绝无效的 URI（如包含空格的 URI）。

Amazon Neptune 引擎更新 2018 年 9 月 6 日

版本：1.0.1.0.200237.0

Amazon Neptune 1.0.1.0.200237.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200237.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200237.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 修复了某些 SPARQL `COUNT(DISTINCT)` 查询失败的问题。
- 修复了带有 `DISTINCT` 子句的 `COUNT`、`SUM`、`MIN` 查询内存不足的问题。
- 修复了 `BLOB` 类型数据将导致 Neptune 加载程序作业失败的问题。
- 修复了重复插入会导致事务故障的问题。
- 修复了 `DROP ALL` 查询无法取消的问题。
- 修复了 Gremlin 客户端可能会间歇性挂起的问题。

- 将负载超过 150M 的所有错误代码更新为了 HTTP 400。
- 改进了单三元组模式 COUNT() 查询的性能和准确性。
- 改进了带有 BIND 子句的 SPARQL UNION 查询的性能。

Amazon Neptune 引擎更新 2018 年 7 月 24 日

版本：1.0.1.0.200236.0

Amazon Neptune 1.0.1.0.200236.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200236.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200236.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 更新了 xsd:string 数据类型的 SPARQL 序列化。xsd:string 不再包含在 JSON 序列化中，JSON 序列化现在与其他输出格式一致。
- 修复了 xsd:double/xsd:float 无穷的处理。-INF、NaN 和 INF 值现在可正确识别，并在所有 SPARQL 数据加载器格式、SPARQL 1.1 UPDATE 和 SPARQL 1.1 Query 中进行处理。

- 修复了具有空字符串值的 Gremlin 查询意外失败的问题。
- 修复了空图表上的 Gremlin `aggregate()` 和 `cap()` 意外失败的问题。
- 修复了在基数规范无效时为 Gremlin 返回了不正确的错误响应的问题，例如 `.property(set,id,'10')` 和 `.property(single,id,'10')`。
- 修复了作为 InternalFailureException 返回了无效的 Gremlin 语法的问题。
- 将错误消息中 `TimeLimitExceeededException` 的拼写修复为 `TimeLimitExceededException`。
- 更改了 SPARQL 和 GREMLIN 终端节点在未提供任何脚本时以一致方式进行响应。
- 阐明了过多并发请求的错误消息。

Amazon Neptune 引擎更新 2018 年 6 月 22 日

版本：1.0.1.0.200233.0

Amazon Neptune 1.0.1.0.200233.0 已正式发布。在该区域的引擎更新完成后，将在 Neptune 1.0.1.0.200233.0 中创建所有新的 Neptune 数据库集群，包括从快照恢复的集群。

可以使用控制台上的数据库集群操作或者使用开发工具包来立即将现有集群升级到此版本。您可以使用以下 CLI 命令立即将数据库集群升级到此版本：

```
aws neptune apply-pending-maintenance-action \
    --apply-action system-update \
    --opt-in-type immediate \
    --resource-identifier arn:aws:rds:<region>:<account number>:<resourcetype>:<name>
```

在系统维护时段期间，Neptune 数据库集群将自动升级到引擎版本 1.0.1.0.200233.0。应用更新的时间取决于数据库集群的区域和维护时段设置以及更新的类型。

> **Note**
>
> 实例维护时段不适用于引擎更新。

更新将同时应用于数据库集群中的所有实例。更新需要在数据库集群中的所有实例上重启数据库，因此，会出现从 20-30 秒到几分钟的停机，之后您可以继续使用数据库集群。您可以在 Neptune 控制台上查看或更改维护时段设置。

如果您有任何问题或疑问，可通过社区论坛和 AWS Premium Support 联系 AWS Support 团队。

改进

- 修复了快速连续发布大量批量加载请求导致出错的问题。
- 修复了与数据相关的查询可能会失败并出现 InternalServerError 的问题。以下示例显示受影响的查询类型。

```
g.V("my-id123").as("start").outE("knows").has("edgePropertyKey1",
 P.gt(0)).as("myedge").inV()
               .as("end").select("start", "end", "myedge").by("vertexPropertyKey1")
               .by("vertexPropertyKey1").by("edgePropertyKey1")
```

- 修复了在长时间运行的查询超时之后，Gremlin Java 客户端无法使用相同的 WebSocket 连接来连接到服务器的问题。
- 修复了无法正确处理以下情况的问题：使用 HTTP 的 Gremlin 查询中包含的转义序列；或者使用 WebSocket 连接的基于字符串的查询。

Amazon Neptune API 使用简介

Amazon Neptune 管理 API 为创建、管理和删除 Neptune 数据库集群和实例提供 SDK 支持，而 Neptune 数据 API 则提供 SDK 支持，旨在将数据加载到图形中、运行查询、获取图形中数据的相关信息以及运行机器学习操作。这些 API 在所有 SDK 语言中都可用。通过自动签署 API 请求，它们非常简单地将 Neptune 集成到应用程序中。

本页提供有关如何使用这些 API 的信息。

IAM 操作的名称与 Neptune 数据 API SDK 的名称不同

当您在启用了 IAM 身份验证的集群上调用 Neptune API 方法时，您必须为进行调用的用户或角色附加一个 IAM policy，从而为要执行的操作提供权限。您可以使用相应的 IAM 操作在策略中设置这些权限。还可以使用 IAM 条件键限制可以采取的操作。

大多数 IAM 操作与其对应的 API 方法同名，但数据 API 中的某些方法具有不同的名称，因为有些方法由多个方法共享。下表列出了数据方法及其对应的 IAM 操作：

数据 API 操作名称	IAM 通信信息
CancelGremlinQuery (cancel_gremlin_query)	操作：neptune-db: **CancelQuery**
CancelLoaderJob (cancel_loader_job)	操作：neptune-db:CancelLoaderJob
CancelMLDataProcessingJob (cancel_ml_data_processing_job)	操作：neptune-db:CancelMLDataProcessingJob
CancelMLModelTrainingJob (cancel_ml_model_training_job)	操作：neptune-db:CancelMLModelTrainingJob
CancelOpenCypherQuery (cancel_open_cypher_query)	操作：neptune-db: **CancelQuery**

数据 API 操作名称	IAM 通信信息
CreateMLEndpoint (create_ml_endpoint)	操作：neptune-db:CreateMLEndpoint
DeleteMLEndpoint (delete_ml_endpoint)	操作：neptune-db:DeleteMLEndpoint
DeletePropertygraphStatistics (delete_propertygraph_statistics)	操作：neptune-db: **DeleteStatistics**
DeleteSparqlStatistics (delete_sparql_statistics)	操作：neptune-db: **DeleteStatistics**
ExecuteFastReset execute_fast_reset()	操作：neptune-db: **ResetDatabase**
ExecuteGremlinExplainQuery (execute_gremlin_explain_query)	操作： • neptune-db: **ReadDataViaQuery** • neptune-db: **WriteDataViaQuery** • neptune-db: **DeleteDataViaQuery** 条件键：neptune-db:QueryLanguage:Gremlin
ExecuteGremlinProfileQuery (execute_gremlin_profile_query)	操作：neptune-db: **ReadDataViaQuery** 条件键：neptune-db:QueryLanguage:Gremlin

数据 API 操作名称	IAM 通信信息	
ExecuteGremlinQuery (execute_gremlin_query)	操作： • neptune-db: **ReadDataViaQuery** • neptune-db: **WriteDataViaQuery** • neptune-db: **DeleteDataViaQuery** 条件键：neptune-db:QueryLanguage:Gremlin	
ExecuteOpenCypherExplainQuery (execute_open_cypher_explain_query)	操作：neptune-db: **ReadDataViaQuery** 条件键：neptune-db:QueryLanguage:OpenCypher	
ExecuteOpenCypherQuery (execute_open_cypher_query)	操作： • neptune-db: **ReadDataViaQuery** • neptune-db: **WriteDataViaQuery** • neptune-db: **DeleteDataViaQuery** 条件键：neptune-db:QueryLanguage:OpenCypher	

数据 API 操作名称	IAM 通信信息	
GetEngineStatus (get_engine_status)	操作：neptune-db:GetEngineStatus	
GetGremlinQueryStatus (get_gremlin_query_status)	操作：neptune-db: **:GetQueryStatus** 条件键：neptune-db:QueryLanguage:Gremlin	
GetLoaderJobStatus (get_loader_job_status)	操作：neptune-db:GetLoaderJobStatus	
GetMLDataProcessingJob (get_ml_data_processing_job)	操作：neptune-db: **GetMLDataProcessingJobStatus**	
GetMLEndpoint (get_ml_endpoint)	操作：neptune-db: **GetMLEndpointStatus**	
GetMLModelTrainingJob (get_ml_model_training_job)	操作：neptune-db: **GetMLModelTrainingJobStatus**	
GetMLModelTransformJob (get_ml_model_transform_job)	操作：neptune-db: **GetMLModelTransformJobStatus**	
GetOpenCypherQueryStatus (get_open_cypher_query_status)	操作：neptune-db: **:GetQueryStatus** 条件键：neptune-db:QueryLanguage:OpenCypher	

数据 API 操作名称	IAM 通信信息	
GetPropertygraphStatistics (get_propertygraph_statistics)	操作：neptune-db: **GetStatisticsStatus**	
GetPropertygraphStream (get_propertygraph_stream)	操作：neptune-db: **GetStreamRecords** 条件键： • neptune-db:QueryLanguage:Gremlin • neptune-db:QueryLanguage:OpenCypher	
GetPropertygraphSummary (get_propertygraph_summary)	操作：neptune-db: **GetGraphSummary**	
GetRDFGraphSummary (get_rdf_graph_summary)	操作：neptune-db: **GetGraphSummary**	
GetSparqlStatistics (get_sparql_statistics)	操作：neptune-db: **GetStatisticsStatus**	
GetSparqlStream (get_sparql_stream)	操作：neptune-db: **:GetStreamRecords** 条件键：neptune-db:QueryLanguage:Sparql	
ListGremlinQueries (list_gremlin_queries)	操作：neptune-db: **:GetQueryStatus** 条件键：neptune-db:QueryLanguage:Gremlin	

数据 API 操作名称	IAM 通信信息
ListMLEndpoints (list_ml_endpoints)	操作：neptune-db:ListMLEndpoints
ListMLModelTrainingJobs (list_ml_model_training_jobs)	操作：neptune-db:ListMLModelTrainingJobs
ListMLModelTransformJobs (list_ml_model_transform_jobs)	操作：neptune-db:ListMLModelTransformJobs
ListOpenCypherQueries (list_open_cypher_queries)	操作：neptune-db: **:GetQueryStatus** 条件键：neptune-db:QueryLanguage:OpenCypher
ManagePropertygraphStatistics (manage_propertygraph_statistics)	操作：neptune-db: **ManageStatistics**
ManageSparqlStatistics (manage_sparql_statistics)	操作：neptune-db: **ManageStatistics**
StartLoaderJob (start_loader_job)	操作：neptune-db:StartLoaderJob
StartMLModelDataProcessingJob (start_ml_data_processing_job)	操作：neptune-db:StartMLModelDataProcessingJob
StartMLModelTrainingJob (start_ml_model_training_job)	操作：neptune-db:StartMLModelTrainingJob

数据 API 操作名称	IAM 通信信息	
StartMLModelTransformJob (start_ml_model_transform_job)	操作：neptune-db:StartMLModelTransformJob	

Amazon Neptune 管理 API 参考

本章介绍了您可用于管理和维护 Neptune 数据库集群的 Neptune API 操作。

Neptune 在以复制拓扑形式连接的数据库服务器的集群上运行。因此，管理 Neptune 通常涉及将更改部署到多个服务器，以及确保所有 Neptune 副本与主服务器保持同步。

由于 Neptune 会随着数据增加透明地扩展底层存储，因此管理 Neptune 需要相对较少的磁盘存储管理。同样，由于 Neptune 自动执行持续备份，因此 Neptune 集群无需广泛的规划或用于执行备份的停机时间。

目录

- DBLogFileNotFoundFault（结构）
- DBParameterGroupAlreadyExistsFault（结构）
- DBParameterGroupNotFoundFault（结构）
- DBParameterGroupQuotaExceededFault（结构）
- DBSecurityGroupAlreadyExistsFault（结构）
- DBSecurityGroupNotFoundFault（结构）
- DBSecurityGroupNotSupportedFault（结构）
- DBSecurityGroupQuotaExceededFault（结构）
- DBSnapshotAlreadyExistsFault（结构）
- DBSnapshotNotFoundFault（结构）
- DBSubnetGroupAlreadyExistsFault（结构）
- DBSubnetGroupDoesNotCoverEnoughAZs（结构）
- DBSubnetGroupNotAllowedFault（结构）
- DBSubnetGroupNotFoundFault（结构）
- DBSubnetGroupQuotaExceededFault（结构）
- DBSubnetQuotaExceededFault（结构）
- DBUpgradeDependencyFailureFault（结构）
- DomainNotFoundFault（结构）
- EventSubscriptionQuotaExceededFault（结构）
- GlobalClusterAlreadyExistsFault（结构）
- GlobalClusterNotFoundFault（结构）
- GlobalClusterQuotaExceededFault（结构）
- InstanceQuotaExceededFault（结构）
- InsufficientDBClusterCapacityFault（结构）
- InsufficientDBInstanceCapacityFault（结构）
- InsufficientStorageClusterCapacityFault（结构）
- InvalidDBClusterEndpointStateFault（结构）
- InvalidDBClusterSnapshotStateFault（结构）
- InvalidDBClusterStateFault（结构）
- InvalidDBInstanceStateFault（结构）

- InvalidDBParameterGroupStateFault（结构）
- InvalidDBSecurityGroupStateFault（结构）
- InvalidDBSnapshotStateFault（结构）
- InvalidDBSubnetGroupFault（结构）
- InvalidDBSubnetGroupStateFault（结构）
- InvalidDBSubnetStateFault（结构）
- InvalidEventSubscriptionStateFault（结构）
- InvalidGlobalClusterStateFault（结构）
- InvalidOptionGroupStateFault（结构）
- InvalidRestoreFault（结构）
- InvalidSubnet（结构）
- InvalidVPCNetworkStateFault（结构）
- KMSKeyNotAccessibleFault（结构）
- OptionGroupNotFoundFault（结构）
- PointInTimeRestoreNotEnabledFault（结构）
- ProvisionedIopsNotAvailableInAZFault（结构）
- ResourceNotFoundFault（结构）
- SNSInvalidTopicFault（结构）
- SNSNoAuthorizationFault（结构）
- SNSTopicArnNotFoundFault（结构）
- SharedSnapshotQuotaExceededFault（结构）
- SnapshotQuotaExceededFault（结构）
- SourceNotFoundFault（结构）
- StorageQuotaExceededFault（结构）
- StorageTypeNotSupportedFault（结构）
- SubnetAlreadyInUse（结构）
- SubscriptionAlreadyExistFault（结构）
- SubscriptionCategoryNotFoundFault（结构）
- SubscriptionNotFoundFault（结构）

Neptune 数据库集群 API

操作：

- CreateDBCluster（操作）
- DeleteDBCluster（操作）
- ModifyDBCluster（操作）
- StartDBCluster（操作）
- StopDBCluster（操作）
- AddRoleToDBCluster（操作）
- RemoveRoleFromDBCluster（操作）
- FailoverDBCluster（操作）
- PromoteReadReplicaDBCluster（操作）
- DescribeDBClusters（操作）

结构：

- DBCluster（结构）
- DBClusterMember（结构）
- DBClusterRole（结构）
- CloudwatchLogsExportConfiguration（结构）
- PendingCloudwatchLogsExports（结构）
- ClusterPendingModifiedValues（结构）

CreateDBCluster（操作）

> 此 API 的 AWS CLI 名称为：`create-db-cluster`。

创建新的 Amazon Neptune 数据库集群。

您可以使用 `ReplicationSourceIdentifier` 参数创建数据库集群，作为其他数据库集群或 Amazon Neptune 数据库实例的只读副本。

请注意，当您直接使用 CreateDBCluster 创建新的集群时，默认情况下会禁用删除保护（在控制台中创建新的生产集群时，默认情况下会启用删除保护）。仅在数据库集群的 DeletionProtection 字段设置为 false 时，才能删除数据库集群。

Request

- AvailabilityZones（在 CLI 中：--availability-zones）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  可在其中创建数据库集群的实例的 EC2 可用区列表。
- BackupRetentionPeriod（在 CLI 中：--backup-retention-period）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  自动备份的保留天数。您必须指定最小值 1。

  默认值：1

  约束：
  - 值必须介于 1 到 35 之间
- CopyTagsToSnapshot（在 CLI 中：--copy-tags-to-snapshot）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  如果设置为 *true*，则标签将复制到所创建的数据库集群的任何快照中。
- DatabaseName（在 CLI 中：--database-name）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库的名称，最多 64 个字母和数字字符。如果您不提供名称，Amazon Neptune 就不会在您正在创建的数据库集群中创建数据库。
- DBClusterIdentifier（在 CLI 中：--db-cluster-identifier）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群标识符。该参数作为一个小写字符串存储。

  约束：
  - 必须包含 1 到 63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结束或包含两个连续连字符。

  例如：my-cluster1

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与该数据库集群关联的数据库集群参数组的名称。如果省略此参数，则使用默认值。

  约束：

  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。
- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与此数据库集群关联的数据库子网组。

  约束：必须与现有 DBSubnetGroup 的名称匹配。不能是默认值。

  例如：`mySubnetgroup`
- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。默认情况下，将启用删除保护。
- EnableCloudwatchLogsExports（在 CLI 中：`--enable-cloudwatch-logs-exports`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群应导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型为：`audit`（用于发布审计日志）和 `slowquery`（用于发布慢速查询日志）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。
- EnableIAMDatabaseAuthentication（在 CLI 中：`--enable-iam-database-authentication`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则为整个数据库集群启用 Amazon Identity and Access Management (IAM) 身份验证（无法在实例级进行设置）。

  默认值：`false`。
- Engine（在 CLI 中：`--engine`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于此数据库集群的数据库引擎的名称。

  有效值：`neptune`

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于新数据库集群的数据库引擎的版本号。

  例如：`1.2.1.0`

- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  应向其中添加此新数据库集群的 Neptune 全球数据库的 ID。

- KmsKeyId（在 CLI 中：`--kms-key-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  加密数据库集群的 Amazon KMS 密钥标识符。

  KMS 密钥标识符是 KMS 加密密钥的 Amazon 资源名称 (ARN)。如果您创建数据库集群时使用的 Amazon 账户，是拥有加密新数据库集群所用 KMS 加密密钥的同一个账户，则您可以使用 KMS 密钥别名来取代 KMS 加密密钥的 ARN。

  未在 `KmsKeyId` 中指定加密密钥时：

  - 如果 `ReplicationSourceIdentifier` 标识了加密的源，则 Amazon Neptune 将使用加密源所用的加密密钥。否则，Amazon Neptune 将使用您的默认加密密钥。
  - 如果 `StorageEncrypted` 参数为 true，并且未指定 `ReplicationSourceIdentifier`，则 Amazon Neptune 将使用您的默认加密密钥。

  Amazon KMS 将为您的 Amazon 账户创建默认加密密钥。您的 Amazon 账户在每个 Amazon 区域都有一个不同的默认加密密钥。

  如果您在其它 Amazon 区域中创建加密数据库集群的只读副本，则必须将 `KmsKeyId` 设置为在目标 Amazon 区域中有效的 KMS 密钥 ID。此密钥用于在该 Amazon 区域中加密只读副本。

- Port（在 CLI 中：`--port`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  数据库集群中的实例接受连接的端口号。

  默认值：8182

- PreferredBackupWindow（在 CLI 中：`--preferred-backup-window`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

使用 BackupRetentionPeriod 参数启用了自动备份时，自动执行备份的日常时间范围。

默认值为从每个 Amazon 区域的 8 小时时段中随机选择的 30 分钟时间。要查看可用的时段，请参阅《Amazon Neptune 用户指南》中的 Neptune 维护时段。

约束：

- 必须采用 hh24:mi-hh24:mi 格式。
- 必须采用通用协调时间（UTC）。
- 不得与首选维护时段冲突。
- 必须至少为 30 分钟。

- PreferredMaintenanceWindow（在 CLI 中：--preferred-maintenance-window）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  可进行系统维护的每周时间范围（采用通用协调时间（UTC））。

  格式：ddd:hh24:mi-ddd:hh24:mi

  默认值为每个 Amazon 区域 8 小时的时段中随机选择的 30 分钟时段（随机选取周中的某天进行）。要查看可用的时段，请参阅《Amazon Neptune 用户指南》中的 Neptune 维护时段。

  有效值：Mon、Tue、Wed、Thu、Fri、Sat、Sun。

  约束：至少为 30 分钟的时段。

- PreSignedUrl（在 CLI 中：--pre-signed-url）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  当前不支持此参数。

- ReplicationSourceIdentifier（在 CLI 中：--replication-source-identifier）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  源数据库实例或数据库集群（如果该数据库集群是作为只读副本创建的）的 Amazon 资源名称 (ARN)。

- ServerlessV2ScalingConfiguration（在 CLI 中：--serverless-v2-scaling-configuration）– ServerlessV2ScalingConfiguration 对象。

  包含 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- StorageEncrypted（在 CLI 中：`--storage-encrypted`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。

- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新数据库集群的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序配置经济实惠的数据库存储。设置为 `standard` 时，响应中不返回存储类型。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  分配到新数据库集群的标签。

- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与此数据库集群关联的 EC2 VPC 安全组的列表。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。
- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。
- ReaderEndpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。
- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。
- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

有效值：

- **`standard`** - （默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterAlreadyExistsFault
- InsufficientStorageClusterCapacityFault
- DBClusterQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- InvalidVPCNetworkStateFault
- InvalidDBClusterStateFault
- InvalidDBSubnetGroupStateFault
- InvalidSubnet
- InvalidDBInstanceStateFault
- DBClusterParameterGroupNotFoundFault
- KMSKeyNotAccessibleFault
- DBClusterNotFoundFault
- DBInstanceNotFoundFault
- DBSubnetGroupDoesNotCoverEnoughAZs
- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault

DeleteDBCluster（操作）

此 API 的 AWS CLI 名称为：`delete-db-cluster`。

DeleteDBCluster 操作将删除先前预置的数据库集群。在您删除数据库集群时，会删除该数据库集群的所有自动备份，且无法恢复。系统不会删除指定数据库集群的手动数据库集群快照。

请注意，如果启用了删除保护，则无法删除数据库集群。要删除数据库集群，您必须先将其 `DeletionProtection` 字段设置为 `False`。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的数据库集群的数据库集群标识符。该参数不区分大小写。

  约束：

  - 必须匹配现有 DBClusterIdentifier。
- FinalDBSnapshotIdentifier（在 CLI 中：`--final-db-snapshot-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  `SkipFinalSnapshot` 设置为 `false` 时，新创建数据库集群快照的数据库集群快照标识符。

  > **Note**
  >
  > 指定此参数并且将 `SkipFinalShapshot` 参数设置为 true 会导致错误。

  约束：

  - 必须为 1 到 255 个字母、数字或连字符。
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符
- SkipFinalSnapshot（在 CLI 中：`--skip-final-snapshot`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  确定删除数据库集群之前是否创建最终数据库集群快照。如果指定 `true`，则不创建数据库集群快照。如果指定 `false`，则删除数据库集群之前创建一个数据库集群快照。

> **Note**
>
> 如果 SkipFinalSnapshot 为 false，则必须指定 FinalDBSnapshotIdentifier。

默认值：false

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  AllocatedStorage 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：long（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：long（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：audit（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：string（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：[A-Za-z][0-9A-Za-z-:._]*。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 iopt1 存储类型。

- KmsKeyId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果 StorageEncrypted 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 ModifyDBCluster 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 BackupRetentionPeriod 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。
- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault

- DBClusterSnapshotAlreadyExistsFault
- SnapshotQuotaExceededFault
- InvalidDBClusterSnapshotStateFault

ModifyDBCluster（操作）

此 API 的 AWS CLI 名称为：`modify-db-cluster`。

修改数据库集群的设置。您可通过在请求中指定这些参数以及新值，更改一个或多个数据库配置参数。

Request

- AllowMajorVersionUpgrade（在 CLI 中：`--allow-major-version-upgrade`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个指示是否允许在不同的主要版本之间进行升级的值。

  约束：当提供使用与数据库集群当前版本不同的主要版本的 `EngineVersion` 参数时，必须设置 allow-major-version-upgrade 标志。
- ApplyImmediately（在 CLI 中：`--apply-immediately`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定应尽快异步应用此请求中修改及任何待处理修改的值，无论数据库集群的 `PreferredMaintenanceWindow` 设置如何。如果此参数设置为 `false`，则在下一个维护时段中应用对数据库集群的更改。

  `ApplyImmediately` 参数仅影响 `NewDBClusterIdentifier` 值。如果将 `ApplyImmediately` 参数值设置为 false，则对 `NewDBClusterIdentifier` 值的更改在下一维护时段中应用。所有其他更改会立即应用，而不管 `ApplyImmediately` 参数的值如何。

  默认值：`false`
- BackupRetentionPeriod（在 CLI 中：`--backup-retention-period`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  自动备份的保留天数。您必须指定最小值 1。

  默认值：1

  约束：

  - 值必须介于 1 到 35 之间
- CloudwatchLogsExportConfiguration (在 CLI 中：`--cloudwatch-logs-export-configuration`) – CloudwatchLogsExportConfiguration 对象。

  要启用日志类型的配置设置，以便针对特定数据库集群导出到 CloudWatch Logs。请参阅使用 CLI 将 Neptune 审计日志发布到 CloudWatch Logs。
- CopyTagsToSnapshot (在 CLI 中：`--copy-tags-to-snapshot`) – BooleanOptional，类型为：`boolean` [布尔值 (true 或 false)]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。
- DBClusterIdentifier (在 CLI 中：`--db-cluster-identifier`) – 必需：一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  要修改的集群的数据库集群标识符。此参数不区分大小写。

  约束：

  - 必须匹配现有 DBCluster 的标识符。
- DBClusterParameterGroupName (在 CLI 中：`--db-cluster-parameter-group-name`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  用于数据库集群的数据库集群参数组名称。
- DBInstanceParameterGroupName (在 CLI 中：`--db-instance-parameter-group-name`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  应用于数据库集群所有实例的数据库参数组的名称。

  > **Note**
  >
  > 使用 `DBInstanceParameterGroupName` 应用参数组时，参数更改不会在下一个维护时段内应用，而是会立即应用。

  默认：现有名称设置

  约束：

  - 数据库参数组必须与目标数据库集群版本位于同一个数据库参数组系列中。

  - DBInstanceParameterGroupName 参数仅在与 AllowMajorVersionUpgrade 参数组合使用时才有效。
- DeletionProtection（在 CLI 中：--deletion-protection）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  一个值，指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。默认情况下，将禁用删除保护。
- EnableIAMDatabaseAuthentication（在 CLI 中：--enable-iam-database-authentication）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  如果启用 Amazon Identity and Access Management (IAM) 账户与数据库账户之间的映射，则为 true；否则为 false。

  默认值：false
- EngineVersion（在 CLI 中：--engine-version）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要升级到的数据库引擎的版本号。更改此参数会导致中断。除非 ApplyImmediately 参数设置为 true，否则会在下个维护时段内应用更改。

  有关有效引擎版本的列表，请参阅 Amazon Neptune 的引擎版本，或调用 the section called “DescribeDBEngineVersions”。
- NewDBClusterIdentifier（在 CLI 中：--new-db-cluster-identifier）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  重命名数据库集群时数据库集群的新标识符。此值以一个小写字符串存储。

  约束：
  - 必须包含 1 到 63 个字母、数字或连字符
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符

  例如：my-cluster2
- Port（在 CLI 中：--port）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  数据库集群用于接受连接的端口号。

  约束：值必须为 1150-65535

默认值：与原始数据库集群相同的端口。

- PreferredBackupWindow（在 CLI 中：`--preferred-backup-window`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  使用 `BackupRetentionPeriod` 参数启用了自动备份时，自动执行备份的日常时间范围。

  默认值为从每个 Amazon 区域的 8 小时时段中随机选择的 30 分钟时间。

  约束：

  - 必须采用 `hh24:mi-hh24:mi` 格式。
  - 必须采用通用协调时间（UTC）。
  - 不得与首选维护时段冲突。
  - 必须至少为 30 分钟。
- PreferredMaintenanceWindow（在 CLI 中：`--preferred-maintenance-window`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  可进行系统维护的每周时间范围（采用通用协调时间（UTC））。

  格式：`ddd:hh24:mi-ddd:hh24:mi`

  默认值为每个 Amazon 区域 8 小时的时段中随机选择的 30 分钟时段（随机选取周中的某天进行）。

  有效值：Mon、Tue、Wed、Thu、Fri、Sat、Sun。

  约束：至少为 30 分钟的时段。
- ServerlessV2ScalingConfiguration（在 CLI 中：`--serverless-v2-scaling-configuration`）– ServerlessV2ScalingConfiguration 对象。

  包含 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与数据库集群关联的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序配置经济实惠的数据库存储。

- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所属的 VPC 安全组的列表。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。
- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。
- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。
- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。
- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。
- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。
- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：audit（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：string（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：[A-Za-z][0-9A-Za-z-:._]*。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 iopt1 存储类型。

- KmsKeyId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果 StorageEncrypted 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 ModifyDBCluster 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 BackupRetentionPeriod 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。
- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault

- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- InvalidVPCNetworkStateFault
- InvalidDBSubnetGroupStateFault
- InvalidSubnet
- DBClusterParameterGroupNotFoundFault
- InvalidDBSecurityGroupStateFault
- InvalidDBInstanceStateFault
- DBClusterAlreadyExistsFault
- StorageTypeNotSupportedFault

StartDBCluster（操作）

此 API 的 AWS CLI 名称为：`start-db-cluster`。

启动已通过 Amazon 控制台、Amazon CLI stop-db-cluster 命令或 StopDBCluster API 停止的 Amazon Neptune 数据库集群。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要启动的 Neptune 数据库集群的数据库集群标识符。该参数作为一个小写字符串存储。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。
- AssociatedRoles – DBClusterRole 对象的数组。

提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。

- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

指定数据库集群是否已加密。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

StopDBCluster（操作）

此 API 的 AWS CLI 名称为：`stop-db-cluster`。

停止 Amazon Neptune 数据库集群。在停止数据库集群时，Neptune 将保留数据库集群的元数据，包括其终端节点和数据库参数组。

此外，Neptune 将保留事务日志，以便您可以在必要时执行时间点还原。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要停止的 Neptune 数据库集群的数据库集群标识符。该参数作为一个小写字符串存储。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。
- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。
- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。
- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。
- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。
- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。
- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。
- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。
- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。
- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。
- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 ModifyDBCluster 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 BackupRetentionPeriod 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。

- ReplicationSourceIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ReplicationType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：
  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

AddRoleToDBCluster（操作）

此 API 的 AWS CLI 名称为：`add-role-to-db-cluster`。

将 Identity and Access Management (IAM) 角色与 Neptune 数据库集群关联。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  IAM 角色与之关联的数据库集群的名称。
- FeatureName（在 CLI 中：`--feature-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  IAM 角色要关联的 Neptune 数据库集群的特征的名称。有关支持的功能名称的列表，请参阅the section called “DBEngineVersion”。
- RoleArn（在 CLI 中：`--role-arn`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与 Neptune 数据库集群关联的 IAM 角色的 Amazon 资源名称 (ARN)，例如 `arn:aws:iam::123456789012:role/NeptuneAccessRole`。

响应

- 无响应参数。

错误

- DBClusterNotFoundFault
- DBClusterRoleAlreadyExistsFault
- InvalidDBClusterStateFault
- DBClusterRoleQuotaExceededFault

RemoveRoleFromDBCluster（操作）

此 API 的 AWS CLI 名称为：`remove-role-from-db-cluster`。

取消数据库集群与 Identity and Access Management (IAM) 角色的关联。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  从中取消与 IAM 角色关联的数据库集群的名称。

- FeatureName（在 CLI 中：`--feature-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要取消与 IAM 角色关联的数据库集群的特征的名称。有关支持的功能名称的列表，请参阅the section called “DescribeDBEngineVersions”。
- RoleArn（在 CLI 中：`--role-arn`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要取消与数据库集群关联的 IAM 角色的 Amazon 资源名称 (ARN)，例如 `arn:aws:iam::123456789012:role/NeptuneAccessRole`。

响应

- 无响应参数。

错误

- DBClusterNotFoundFault
- DBClusterRoleNotFoundFault
- InvalidDBClusterStateFault

FailoverDBCluster（操作）

此 API 的 AWS CLI 名称为：`failover-db-cluster`。

强制数据库集群失效转移。

数据库集群的故障转移会将数据库集群中的只读副本之一（只读实例）提升为主实例（集群写入器）。

Amazon Neptune 在主实例发生故障时自动故障转移到只读副本（如果存在）。当您要模拟主实例的故障以进行测试时，可以强制进行故障转移。由于数据库集群中的每个实例都有自己的终端节点地址，您需要在故障转移完成后清理并重新建立使用这些终端节点地址的任何现有连接。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于强制进行故障转移的数据库集群标识符。此参数不区分大小写。

约束：

- 必须匹配现有 DBCluster 的标识符。

- TargetDBInstanceIdentifier（在 CLI 中：`--target-db-instance-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提升为主实例的实例的名称。

  您必须指定数据库集群中只读副本的实例标识符。例如，`mydbcluster-replica1`。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。

- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBInstanceStateFault

PromoteReadReplicaDBCluster（操作）

此 API 的 AWS CLI 名称为：`promote-read-replica-db-cluster`。

不支持。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。

- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。

- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterNotFoundFault
- InvalidDBClusterStateFault

DescribeDBClusters（操作）

此 API 的 AWS CLI 名称为：`describe-db-clusters`。

返回有关预配置的数据库集群的信息，并支持分页。

> **Note**
>
> 此操作还可以返回 Amazon RDS 集群和 Amazon DocDB 集群的信息。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用户提供的数据库集群标识符。如果指定了此参数，则只返回特定数据库集群的信息。该参数不区分大小写。

  约束：
  - 如果提供，则必须匹配现有 DBClusterIdentifier。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  筛选条件指定要描述的一个或多个数据库集群。

  支持的筛选条件：

- db-cluster-id - 接受数据库集群标识符和数据库集群 Amazon 资源名称 (ARN)。结果列表中仅包含由这些 ARN 确定的数据库集群的相关信息。
- engine - 接受引擎名称（例如 neptune），并将结果列表限制为由该引擎创建的数据库集群。

例如，要从 Amazon CLI 调用此 API 并进行筛选，以便仅返回 Neptune 数据库集群，您可以使用以下命令：

Example

```
aws neptune describe-db-clusters \
            --filters  Name=engine,Values=neptune
```

- Marker（在 CLI 中：--marker）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  由之前的 the section called “DescribeDBClusters” 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 MaxRecords 指定的值。
- MaxRecords（在 CLI 中：--max-records）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的MaxRecords 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

- DBClusters – DBCluster 对象的数组。

  包含用户的数据库集群列表。
- Marker – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  分页标记，您可在后续的 DescribeDBClusters 请求中使用。

错误

- DBClusterNotFoundFault

结构：

DBCluster（结构）

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

字段

- AllocatedStorage – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。
- AssociatedRoles – 这是 DBClusterRole 对象数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。
- AutomaticRestartTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。
- AvailabilityZones – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。
- BacktrackConsumedChangeRecords – 这是 longOptional，类型为：`long`（有符号的 64 位整数）。

  不受 Neptune 支持。
- BacktrackWindow – 这是 longOptional，类型为：`long`（有符号的 64 位整数）。

  不受 Neptune 支持。
- BackupRetentionPeriod – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。
- Capacity – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。
- CloneGroupId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

标识数据库集群与之关联的克隆组。

- ClusterCreateTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – 这是 DBClusterMember 对象数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – 这是 GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。
- LatestRestorableTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。
- MultiAZ - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。
- PendingModifiedValues –这是一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。
- PercentProgress – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。
- Port – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。
- PreferredBackupWindow – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。
- PreferredMaintenanceWindow – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。
- ReaderEndpoint – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。
- ReadReplicaIdentifiers – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。
- ReplicationSourceIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

不受 Neptune 支持。

- ReplicationType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration –这是一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。

- StorageEncrypted - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。

- StorageType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
  - **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

    Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

- VpcSecurityGroups – 这是 VpcSecurityGroupMembership 对象数组。

  提供数据库集群所属的 VPC 安全组的列表。

`DBCluster` 用作下列对象的响应元素：

- CreateDBCluster
- DeleteDBCluster
- FailoverDBCluster
- ModifyDBCluster
- PromoteReadReplicaDBCluster

- RestoreDBClusterFromSnapshot
- RestoreDBClusterToPointInTime
- StartDBCluster
- StopDBCluster

DBClusterMember（结构）

包含有关属于数据库集群的实例的信息。

字段

- DBClusterParameterGroupStatus – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的此成员的数据库集群参数组状态。
- DBInstanceIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的此成员的实例标识符。
- IsClusterWriter - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果集群成员是数据库集群的主实例，值为 `true`，否则为 `false`。
- PromotionTier – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

DBClusterRole（结构）

描述与数据库集群关联的 Amazon Identity and Access Management（IAM）角色。

字段

- FeatureName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与 Amazon Identity and Access Management（IAM）角色关联的功能的名称。有关支持的功能名称的列表，请参阅the section called “DescribeDBEngineVersions”。
- RoleArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与数据库集群关联的 IAM 角色的 Amazon 资源名称（ARN）。
- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

描述 IAM 角色与数据库集群之间关联的状态。Status 属性返回以下值之一：

- ACTIVE - 该 IAM 角色 ARN 已与数据库集群关联，可用于代表您访问其它 Amazon 服务。
- PENDING - 与数据库集群关联的 IAM 角色 ARN。
- INVALID - 与数据库集群关联的 IAM 角色 ARN，但数据库集群无法代入 IAM 角色以代表您访问其它 Amazon 服务。

CloudwatchLogsExportConfiguration（结构）

要启用日志类型的配置设置，以便针对特定数据库实例或数据库集群导出到 CloudWatch Logs。

EnableLogTypes 和 DisableLogTypes 数组确定要将哪些日志导出（或不导出）到 CloudWatch Logs。

有效的日志类型为：audit（用于发布审计日志）和 slowquery（用于发布慢速查询日志）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

字段

- DisableLogTypes – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  要禁用的日志类型列表。
- EnableLogTypes – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  要启用的日志类型的列表。

PendingCloudwatchLogsExports（结构）

其配置仍处于待处理状态的日志类型的列表。换句话说，这些日志类型正处于激活或停用过程中。

有效的日志类型为：audit（用于发布审计日志）和 slowquery（用于发布慢速查询日志）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

字段

- LogTypesToDisable – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  处于启用过程中的日志类型。启用之后，这些日志类型会导出到 CloudWatch Logs。
- LogTypesToEnable – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

处于停用过程中的日志类型。停用之后，这些日志类型不会导出到 CloudWatch Logs。

ClusterPendingModifiedValues（结构）

此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

字段

- AllocatedStorage – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  分配给数据库引擎的存储空间大小（以 GiB 为单位）。对于 Neptune，`AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。
- BackupRetentionPeriod – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  自动拍摄的数据库快照的保留天数。
- DBClusterIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 DBClusterIdentifier 值。
- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎版本。
- IAMDatabaseAuthenticationEnabled – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示是否启用 AWS Identity and Access Management (IAM) 账户与数据库账户之间的映射。
- Iops – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定预调配 IOPS（每秒 I/O 操作数）值。此设置仅适用于多可用区数据库集群。
- PendingCloudwatchLogsExports –这是一个 PendingCloudwatchLogsExports 对象。

  此 `PendingCloudwatchLogsExports` 结构指定了已启用和已禁用 CloudWatch Logs 的待处理更改。
- StorageType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群存储类型的待定更改。 有效值：

  - **`standard`** -（默认）为 I/O 使用量适中到较小的应用程序配置经济实惠的数据库存储。

- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

Neptune 全球数据库 API

操作:

- CreateGlobalCluster（操作）
- DeleteGlobalCluster（操作）
- ModifyGlobalCluster（操作）
- DescribeGlobalClusters（操作）
- FailoverGlobalCluster（操作）
- RemoveFromGlobalCluster（操作）

结构：

- GlobalCluster（结构）
- GlobalClusterMember（结构）

CreateGlobalCluster（操作）

此 API 的 AWS CLI 名称为：`create-global-cluster`。

创建分布在多个 Amazon 区域中的 Neptune 全球数据库。全球数据库包含单个具有读写功能的主集群和若干只读辅助集群，这些辅助集群通过 Neptune 存储子系统执行的高速复制从主集群接收数据。

您可以创建最初为空的全球数据库，然后向其中添加主集群和辅助集群，也可以在创建操作期间指定现有 Neptune 集群以成为全球数据库的主集群。

请求

- DatabaseName（在 CLI 中：`--database-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新全球数据库的名称（最多 64 个字母数字字符）。

- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  新全局数据库的删除保护设置。在启用删除保护时，无法删除全局数据库。
- Engine（在 CLI 中：`--engine`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于全球数据库中的数据库引擎的名称。

  有效值：`neptune`
- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库要使用的 Neptune 引擎版本。

  有效值为：`1.2.0.0` 或更高版本。
- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– 必需：GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  新全局数据库集群的集群标识符。
- SourceDBClusterIdentifier（在 CLI 中：`--source-db-cluster-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  （可选）要用作新全球数据库的主集群的现有 Neptune 数据库集群的 Amazon 资源名称 (ARN)。
- StorageEncrypted（在 CLI 中：`--storage-encrypted`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  新全球数据库集群的存储加密设置。

响应

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

全球数据库的删除保护设置。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 数据库引擎（`"neptune"`）。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。

- GlobalClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- GlobalClusterMembers – GlobalClusterMember 对象的数组。

  属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。

- GlobalClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。

- StorageEncrypted – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

错误

- GlobalClusterAlreadyExistsFault
- GlobalClusterQuotaExceededFault
- InvalidDBClusterStateFault
- DBClusterNotFoundFault

DeleteGlobalCluster（操作）

此 API 的 AWS CLI 名称为：`delete-global-cluster`。

删除全球数据库。必须首先已分离或删除主集群和所有辅助集群。

请求

- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– 必需：GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  正删除的全球数据库集群的集群标识符。

响应

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的删除保护设置。
- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 数据库引擎（`"neptune"`）。
- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。
- GlobalClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。
- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。
- GlobalClusterMembers – GlobalClusterMember 对象的数组。

属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。

- GlobalClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。
- StorageEncrypted – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

错误

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault

ModifyGlobalCluster（操作）

此 API 的 AWS CLI 名称为：`modify-global-cluster`。

修改 Amazon Neptune 全球集群的设置。您可通过在请求中指定这些参数及其新值，更改一个或多个数据库配置参数。

请求

- AllowMajorVersionUpgrade（在 CLI 中：`--allow-major-version-upgrade`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示是否允许主要版本升级的值。

  约束：如果为 `EngineVersion` 参数指定的主要版本值与数据库集群的当前版本不同，则必须允许主要版本升级。

  如果您升级全球数据库的主要版本，则集群和数据库实例参数组将设置为新版本的默认参数组，因此您需要在完成升级后应用任何自定义参数组。
- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

指示全球数据库是否启用了删除保护。在启用删除保护时，无法删除全球数据库。

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要升级到的数据库引擎的版本号。更改此参数将导致中断。除非启用 `ApplyImmediately`，否则会在下个维护时段内应用更改。

  要列出所有可用的 Neptune 引擎版本，请使用以下命令：

  Example

  ```
  aws neptune describe-db-engine-versions \
            --engine neptune \
            --query '*[]|[?SupportsGlobalDatabases == 'true'].[EngineVersion]'
  ```

- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– 必需：GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  要修改的全球集群的数据库集群标识符。此参数不区分大小写。

  约束：必须与现有全球数据库集群的标识符匹配。

- NewGlobalClusterIdentifier（在 CLI 中：`--new-global-cluster-identifier`）– GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  要分配给全球数据库的新集群标识符。此值以一个小写字符串存储。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结尾，也不能包含两个连续连字符

  示例：`my-cluster2`

响应

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的删除保护设置。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 数据库引擎（`"neptune"`）。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。

- GlobalClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- GlobalClusterMembers – GlobalClusterMember 对象的数组。

  属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。

- GlobalClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。

- StorageEncrypted – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

错误

- GlobalClusterNotFoundFault

- InvalidGlobalClusterStateFault

DescribeGlobalClusters（操作）

此 API 的 AWS CLI 名称为：`describe-global-clusters`。

返回有关 Neptune 全球数据库集群的信息。此 API 支持分页。

请求

- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  用户提供的数据库集群标识符。如果指定了此参数，则只返回有关指定的数据库集群的信息。此参数不区分大小写。

  约束：如果提供，则必须与现有的数据库集群标识符匹配。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  （可选）由上一次调用 `DescribeGlobalClusters` 返回的分页令牌。如果指定此参数，则响应将仅包含标记之外的记录，直至 `MaxRecords` 指定的数量。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数多于指定的 `MaxRecords` 值，则响应中会包含一个分页标记令牌，您可以使用它来检索剩余的结果。

  默认值：`100`

  约束：最低为 20，最高为 100。

响应

- GlobalClusters – GlobalCluster 对象的数组。

  此请求返回的全球集群和实例列表。
- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

分页令牌。如果在响应中返回此参数，则会有更多记录可用，这些记录可通过对 `DescribeGlobalClusters` 的一个或多个额外调用进行检索。

错误

- GlobalClusterNotFoundFault

FailoverGlobalCluster（操作）

此 API 的 AWS CLI 名称为：`failover-global-cluster`。

启动 Neptune 全球数据库的失效转移过程。

Neptune 全球数据库的失效转移会将一个辅助只读数据库集群提升为主数据库集群，并将主数据库集群降级为辅助（只读）数据库集群。换句话说，可以切换当前主数据库集群和选定的目标辅助数据库集群的角色。所选的辅助数据库集群假设 Neptune 全球数据库具有完全读/写能力。

> **Note**
>
> 此操作仅适用于 Neptune 全球数据库。此操作仅适用于运行状况良好、Neptune 数据库集群正常运行且没有区域性中断的 Neptune 全球数据库，用于测试灾难恢复方案或重新配置全球数据库拓扑。

请求

- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– 必需：GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  应进行失效转移的 Neptune 全球数据库的标识符。标识符是用户在创建 Neptune 全球数据库时分配的唯一密钥。换句话说，这是要进行失效转移的全球数据库的名称。

  约束：必须与现有 Neptune 全球数据库的标识符匹配。
- TargetDbClusterIdentifier（在 CLI 中：`--target-db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要提升为全球数据库的主数据库集群的辅助 Neptune 数据库集群的 Amazon 资源名称 (ARN)。

响应

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的删除保护设置。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 数据库引擎（`"neptune"`）。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。

- GlobalClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- GlobalClusterMembers – GlobalClusterMember 对象的数组。

  属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。

- GlobalClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。

- StorageEncrypted – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

错误

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault
- InvalidDBClusterStateFault
- DBClusterNotFoundFault

RemoveFromGlobalCluster（操作）

此 API 的 AWS CLI 名称为：`remove-from-global-cluster`。

将 Neptune 数据库集群与 Neptune 全球数据库分离。辅助集群变成具有读写功能而不是只读的普通独立集群，并且不再接收来自主集群的数据。

请求

- DbClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识要与 Neptune 全球数据库集群分离的集群的 Amazon 资源名称 (ARN)。
- GlobalClusterIdentifier（在 CLI 中：`--global-cluster-identifier`）– 必需：GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  要从中分离指定的 Neptune 数据库集群的 Neptune 全球数据库的标识符。

响应

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的删除保护设置。
- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

全球数据库使用的 Neptune 数据库引擎（"neptune"）。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。

- GlobalClusterArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：string（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：[A-Za-z][0-9A-Za-z-:._]*。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- GlobalClusterMembers – GlobalClusterMember 对象的数组。

  属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。

- GlobalClusterResourceId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。

- Status – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。

- StorageEncrypted – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

错误

- GlobalClusterNotFoundFault
- InvalidGlobalClusterStateFault
- DBClusterNotFoundFault

结构：

GlobalCluster（结构）

包含 Amazon Neptune 全球数据库的详细信息。

此数据类型用作 the section called “CreateGlobalCluster”、the section called “DescribeGlobalClusters”、the section called “ModifyGlobalCluster”、the section called “DeleteGlobalCluster”、the section called “FailoverGlobalCluster” 和 the section called “RemoveFromGlobalCluster” 操作的响应元素。

字段

- DeletionProtection – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的删除保护设置。
- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 数据库引擎（`"neptune"`）。
- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库使用的 Neptune 引擎版本。
- GlobalClusterArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的 Amazon 资源名称 (ARN)。
- GlobalClusterIdentifier – 这是 GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。
- GlobalClusterMembers – 这是 GlobalClusterMember 对象数组。

  属于全球数据库的所有数据库集群的集群 ARN 和实例 ARN 的列表。
- GlobalClusterResourceId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  全球数据库的不可变标识符，在所有区域内都是唯一的。只要访问了用户数据库集群的 KMS 密钥，就可在 CloudTrail 日志条目中找到此标识符。
- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此全球数据库的当前状态。
- StorageEncrypted – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  全球数据库的存储加密设置。

`GlobalCluster` 用作下列对象的响应元素：

- CreateGlobalCluster
- ModifyGlobalCluster
- DeleteGlobalCluster
- RemoveFromGlobalCluster
- FailoverGlobalCluster

GlobalClusterMember（结构）

一种数据结构，其中包含与 Neptune 全球数据库关联的任何主集群和辅助集群的相关信息。

字段

- DBClusterArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个 Neptune 集群的 Amazon 资源名称 (ARN)。
- IsWriter - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定 Neptune 集群是否为与其关联的 Neptune 全球数据库的主集群（即具有读写功能）。
- Readers – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与 Neptune 全球数据库关联的每个只读辅助集群的 Amazon 资源名称 (ARN)。

Neptune 实例 API

操作：

- CreateDBInstance（操作）
- DeleteDBInstance（操作）
- ModifyDBInstance（操作）
- RebootDBInstance（操作）
- DescribeDBInstances（操作）
- DescribeOrderableDBInstanceOptions（操作）
- DescribeValidDBInstanceModifications（操作）

结构：

- DBInstance（结构）
- DBInstanceStatusInfo（结构）
- OrderableDBInstanceOption（结构）
- PendingModifiedValues（结构）
- ValidStorageOptions（结构）
- ValidDBInstanceModificationsMessage（结构）

CreateDBInstance（操作）

此 API 的 AWS CLI 名称为：`create-db-instance`。

创建新的数据库实例。

Request

- AutoMinorVersionUpgrade（在 CLI 中：`--auto-minor-version-upgrade`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示在维护时段期间，将对该数据库实例自动应用次要引擎升级。

  默认值：`true`
- AvailabilityZone（在 CLI 中：`--availability-zone`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  在其中创建了数据库实例的 EC2 可用区

  默认值：端点的 Amazon 区域中系统选择的随机可用区。

  例如：`us-east-1d`

  约束：如果 MultiAZ 设置为 `true`，则不能指定 AvailabilityZone 参数。指定的可用区必须与当前端点位于相同 Amazon 区域中。
- BackupRetentionPeriod（在 CLI 中：`--backup-retention-period`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  自动备份的保留天数。

  不适用。自动备份保留期由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

默认值：1

约束：

- 值必须介于 0 到 35 之间
- 如果数据库实例是只读副本的源，则不能设置为 0。

- CopyTagsToSnapshot（在 CLI 中：`--copy-tags-to-snapshot`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果为 true，则将数据库实例的所有标签复制到数据库实例的快照中，否则为 false。默认值为 false。

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  实例所属的数据库集群的标识符。

  有关创建数据库集群的信息，请参阅the section called “CreateDBCluster”。

  类型：字符串

- DBInstanceClass（在 CLI 中：`--db-instance-class`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的计算和内存容量，例如，`db.m4.large`。并非所有数据库实例类在所有 Amazon 区域中都可用。

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例标识符。该参数作为一个小写字符串存储。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结束或包含两个连续连字符。

  例如：`mydbinstance`

- DBName（在 CLI 中：`--db-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。

- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与此数据库实例关联的数据库参数组的名称。如果省略此参数，则使用指定引擎的默认 DBParameterGroup。

  约束：

  - 必须为 1 到 255 个字母、数字或连字符。
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符
- DBSecurityGroups（在 CLI 中：`--db-security-groups`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与此数据库实例关联的数据库安全组的列表。

  默认值：数据库引擎的默认数据库安全组。
- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与此数据库实例关联的数据库子网组。

  如果不存在数据库子网组，则该实例为非 VPC 数据库实例。
- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护的值。在启用删除保护时，无法删除数据库。默认情况下，将禁用删除保护。请参阅删除数据库实例。

  可以删除数据库集群中的数据库实例，即使在其父数据库集群中启用了删除保护也是如此。
- Domain（在 CLI 中：`--domain`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在其中创建实例的 Active Directory 域。
- DomainIAMRoleName（在 CLI 中：`--domain-iam-role-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在对 Directory Service 进行 API 调用时使用的 IAM 角色的名称。
- EnableCloudwatchLogsExports（在 CLI 中：`--enable-cloudwatch-logs-exports`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

需要启用以导出到 CloudWatch Logs 的日志类型的列表。

- EnableIAMDatabaseAuthentication（在 CLI 中：`--enable-iam-database-authentication`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  不受 Neptune 支持（忽略）。

- Engine（在 CLI 中：`--engine`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于此实例的数据库引擎的名称。

  有效值：`neptune`

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要使用的数据库引擎的版本号。目前，设置此参数不起作用。

- Iops（在 CLI 中：`--iops`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  最初为数据库实例分配的预配置 IOPS（每秒输入/输出操作数）量。

- KmsKeyId（在 CLI 中：`--kms-key-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  加密数据库实例的 Amazon KMS 密钥标识符。

  KMS 密钥标识符是 KMS 加密密钥的 Amazon 资源名称 (ARN)。如果您创建数据库实例时使用的 Amazon 账户，是拥有加密新数据库实例所用 KMS 加密密钥的同一个账户，则您可以使用 KMS 密钥别名，而不用 KMS 加密密钥的 ARN。

  不适用。KMS 密钥标识符由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

  如果 `StorageEncrypted` 参数为 true，并且您没有为 `KmsKeyId` 参数指定值，则 Amazon Neptune 将使用您的默认加密密钥。Amazon KMS 将为您的 Amazon 账户创建默认加密密钥。您的 Amazon 账户在每个 Amazon 区域都有一个不同的默认加密密钥。

- LicenseModel（在 CLI 中：`--license-model`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。

有效值：license-included |bring-your-own-license |general-public-license

- MonitoringInterval（在 CLI 中：--monitoring-interval）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。要禁用收集增强监控指标，请指定 0。默认值是 0。

  如果指定 MonitoringRoleArn，则您还必须将 MonitoringInterval 设置为 0 以外的值。

  有效值：0, 1, 5, 10, 15, 30, 60

- MonitoringRoleArn（在 CLI 中：--monitoring-role-arn）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。例如，arn:aws:iam:123456789012:role/emaccess。

  如果 MonitoringInterval 设置为 0 以外的值，则您必须提供 MonitoringRoleArn 值。

- MultiAZ（在 CLI 中：--multi-az）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。如果 MultiAZ 参数设为 true，则不能设置 AvailabilityZone 参数。

- Port（在 CLI 中：--port）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  数据库实例接受连接的端口编号。

  不适用。端口由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

  默认值：8182

  类型：整数

- PreferredBackupWindow（在 CLI 中：--preferred-backup-window）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  创建自动备份的每日时间范围。

  不适用。每天创建自动备份的时间范围由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

- PreferredMaintenanceWindow（在 CLI 中：`--preferred-maintenance-window`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每周可以进行系统维护的时间范围（采用通用协调时间 (UTC)）。

  格式：`ddd:hh24:mi-ddd:hh24:mi`

  默认值为每个 Amazon 区域 8 小时的时段中随机选择的 30 分钟时段（随机选取周中的某天进行）。

  有效值：Mon、Tue、Wed、Thu、Fri、Sat、Sun。

  约束：至少为 30 分钟的时段。
- PromotionTier（在 CLI 中：`--promotion-tier`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

  默认值：1

  有效值：0 - 15
- PubliclyAccessible（在 CLI 中：`--publicly-accessible`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。
- StorageEncrypted（在 CLI 中：`--storage-encrypted`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否已对数据库实例加密。

  不适用。数据库实例的加密由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

  默认值：false
- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。
- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  分配到新实例的标签。

- TdeCredentialArn（在 CLI 中：`--tde-credential-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。

- TdeCredentialPassword（在 CLI 中：`--tde-credential-password`）– 一个敏感字符串，类型为：`string`（UTF-8 编码的字符串）。

  密钥存储中指定 ARN 的密码，以便访问设备。

- Timezone（在 CLI 中：`--timezone`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的时区。

- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与此数据库实例关联的 EC2 VPC 安全组的列表。

  不适用。关联的 EC2 VPC 安全组列表由数据库集群管理。有关更多信息，请参阅the section called “CreateDBCluster”。

  默认值：数据库子网组的 VPC 的默认 EC2 VPC 安全组。

响应

包含 Amazon Neptune 数据库实例的详细信息。

此数据类型用作 the section called “DescribeDBInstances” 操作中的响应元素。

- AutoMinorVersionUpgrade - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示自动应用次要版本补丁。

- AvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库实例所在可用区域的名称。

- BackupRetentionPeriod – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- CACertificateIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例的 CA 证书的标识符。

- CopyTagsToSnapshot - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定是否将标签从数据库实例复制到数据库实例的快照。

- DBClusterIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果数据库实例属于某个数据库集群，则包含数据库实例所属的数据库集群的名称。

- DBInstanceArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 资源名称 (ARN)。

- DBInstanceClass – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含数据库实例的计算和内存容量级别名称。

- DBInstanceIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含用户提供的数据库标识符。此标识符是标识数据库实例的唯一密钥。

- DbInstancePort – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定数据库实例侦听的端口。如果数据库实例属于某个数据库集群，则此端口可能不同于数据库集群的端口。

- DBInstanceStatus – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此数据库的当前状态。

- DbiResourceId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 区域唯一且不变的标识符。只要访问数据库实例的 Amazon KMS 密钥，就可以在 Amazon CloudTrail 日志条目中找到此标识符。

- DBName – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库名称。

- DBParameterGroups – DBParameterGroupStatus 对象的数组。

  提供应用到此数据库实例的数据库参数组的列表。

- DBSecurityGroups – DBSecurityGroupMembership 对象的数组。

  提供仅包含 DBSecurityGroup.Name 和 DBSecurityGroup.Status 子元素的数据库安全组元素的列表。

- DBSubnetGroup – 一个 DBSubnetGroup 对象。

指定与数据库实例关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护。在启用删除保护时，无法删除实例。请参阅删除数据库实例。

- DomainMemberships – DomainMembership 对象的数组。

  不支持

- EnabledCloudwatchLogsExports – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库实例配置为导出到 CloudWatch Logs 的日志类型的列表。

- Endpoint – 一个 端点 对象。

  指定连接端点。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供要用于此数据库实例的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- EnhancedMonitoringResourceArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  接收数据库实例的增强监控指标数据的 Amazon CloudWatch Logs 日志流的 Amazon 资源名称 (ARN)。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果 Amazon Identity and Access Management (IAM) 身份验证已启用，则为 true；否则为 false。

- InstanceCreateTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供创建数据库实例的日期和时间。

- Iops – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定预配置的 IOPS（每秒 I/O 操作数）值。

- KmsKeyId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不支持：数据库实例的加密由数据库集群管理。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。

- MonitoringInterval – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。

- MonitoringRoleArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。

- PendingModifiedValues – 一个 PendingModifiedValues 对象。

  指定对数据库实例的更改待处理。仅在更改待处理时包含此元素。特定更改由子元素标识。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- PromotionTier – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

- PubliclyAccessible - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。

- ReadReplicaDBClusterIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个数据集集群（作为此数据库实例只读副本的集群）的标识符。

- ReadReplicaDBInstanceIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库实例关联的只读副本的标识符。

- ReadReplicaSourceDBInstanceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含源数据库实例（在此数据库实例是只读副本时）的标识符。
- SecondaryAvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果存在，则指定带多可用区支持的数据库实例辅助可用区的名称。
- StatusInfos – DBInstanceStatusInfo 对象的数组。

  只读副本的状态。如果实例不是只读副本，则此项空白。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  不支持：数据库实例的加密由数据库集群管理。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库实例关联的存储类型。
- TdeCredentialArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。
- Timezone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库实例所属的 VPC 安全组元素的列表。

错误

- DBInstanceAlreadyExistsFault
- InsufficientDBInstanceCapacityFault
- DBParameterGroupNotFoundFault
- DBSecurityGroupNotFoundFault
- InstanceQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- DBSubnetGroupDoesNotCoverEnoughAZs

- InvalidDBClusterStateFault
- InvalidSubnet
- InvalidVPCNetworkStateFault
- ProvisionedIopsNotAvailableInAZFault
- OptionGroupNotFoundFault
- DBClusterNotFoundFault
- StorageTypeNotSupportedFault
- AuthorizationNotFoundFault
- KMSKeyNotAccessibleFault
- DomainNotFoundFault

DeleteDBInstance（操作）

此 API 的 AWS CLI 名称为：`delete-db-instance`。

DeleteDBInstance 操作将删除先前预置的数据库实例。删除数据库实例时，会删除该实例的所有自动备份，且无法恢复。使用 `DeleteDBInstance` 删除数据库实例时，不删除其手动数据库快照。

如果您请求最终数据库快照，则在创建数据库快照之前，Amazon Neptune 数据库实例的状态为 `deleting`。API 操作 `DescribeDBInstance` 可用于监控此操作的状态。操作一旦提交就无法取消或恢复。

请注意，当数据库实例处于 `failed`、`incompatible-restore` 或 `incompatible-network` 的失败状态时，只有在将 `SkipFinalSnapshot` 参数设置为 `true` 时才能删除它。

如果数据库实例是数据库集群中的唯一实例，或者启用了删除保护，则无法删除该实例。

Request

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的数据库实例的数据库实例标识符。该参数不区分大小写。

  约束：

  - 必须与现有数据库实例的名称匹配。

- FinalDBSnapshotIdentifier（在 CLI 中：`--final-db-snapshot-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  SkipFinalSnapshot 设置为 `false` 时，所创建新 DBSnapshot 的 DBSnapshotIdentifier。

  > **Note**
  >
  > 指定此参数并且将 SkipFinalShapshot 参数设置为 true 会导致错误。

  约束：

  - 必须为 1 到 255 个字母或数字。
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符
  - 删除只读副本时无法指定。
- SkipFinalSnapshot（在 CLI 中：`--skip-final-snapshot`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  确定删除数据库实例之前是否创建最终数据库快照。如果指定 `true`，则不创建 DBSnapshot。如果指定 `false`，则删除数据库实例之前创建一个数据库快照。

  请注意，当数据库实例处于“failed”、“incompatible-restore”或“incompatible-network”的失败状态时，只能在 SkipFinalSnapshot 参数设置为“true”时删除该实例。

  删除只读副本时指定 `true`。

  > **Note**
  >
  > 如果 SkipFinalSnapshot 为 `false`，则必须指定 FinalDBSnapshotIdentifier 参数。

  默认值：`false`

响应

包含 Amazon Neptune 数据库实例的详细信息。

此数据类型用作 the section called “DescribeDBInstances” 操作中的响应元素。

- AutoMinorVersionUpgrade - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指示自动应用次要版本补丁。

- AvailabilityZone – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库实例所在可用区域的名称。

- BackupRetentionPeriod – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- CACertificateIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库实例的 CA 证书的标识符。

- CopyTagsToSnapshot - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定是否将标签从数据库实例复制到数据库实例的快照。

- DBClusterIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果数据库实例属于某个数据库集群，则包含数据库实例所属的数据库集群的名称。

- DBInstanceArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 资源名称 (ARN)。

- DBInstanceClass – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含数据库实例的计算和内存容量级别名称。

- DBInstanceIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含用户提供的数据库标识符。此标识符是标识数据库实例的唯一密钥。

- DbInstancePort – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定数据库实例侦听的端口。如果数据库实例属于某个数据库集群，则此端口可能不同于数据库集群的端口。

- DBInstanceStatus – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此数据库的当前状态。

- DbiResourceId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 区域唯一且不变的标识符。只要访问数据库实例的 Amazon KMS 密钥，就可以在 Amazon CloudTrail 日志条目中找到此标识符。

- DBName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库名称。

- DBParameterGroups – DBParameterGroupStatus 对象的数组。

  提供应用到此数据库实例的数据库参数组的列表。

- DBSecurityGroups – DBSecurityGroupMembership 对象的数组。

  提供仅包含 `DBSecurityGroup.Name` 和 `DBSecurityGroup.Status` 子元素的数据库安全组元素的列表。

- DBSubnetGroup – 一个 DBSubnetGroup 对象。

  指定与数据库实例关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护。在启用删除保护时，无法删除实例。请参阅删除数据库实例。

- DomainMemberships – DomainMembership 对象的数组。

  不支持

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例配置为导出到 CloudWatch Logs 的日志类型的列表。

- Endpoint – 一个 端点 对象。

  指定连接端点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库实例的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- EnhancedMonitoringResourceArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  接收数据库实例的增强监控指标数据的 Amazon CloudWatch Logs 日志流的 Amazon 资源名称 (ARN)。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果 Amazon Identity and Access Management (IAM) 身份验证已启用，则为 true；否则为 false。

- InstanceCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供创建数据库实例的日期和时间。

- Iops – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定预配置的 IOPS（每秒 I/O 操作数）值。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持：数据库实例的加密由数据库集群管理。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。

- MonitoringInterval – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。

- MonitoringRoleArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。

- PendingModifiedValues – 一个 PendingModifiedValues 对象。

  指定对数据库实例的更改待处理。仅在更改待处理时包含此元素。特定更改由子元素标识。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- PromotionTier – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

- PubliclyAccessible - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。
- ReadReplicaDBClusterIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个数据集集群（作为此数据库实例只读副本的集群）的标识符。
- ReadReplicaDBInstanceIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库实例关联的只读副本的标识符。
- ReadReplicaSourceDBInstanceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含源数据库实例（在此数据库实例是只读副本时）的标识符。
- SecondaryAvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果存在，则指定带多可用区支持的数据库实例辅助可用区的名称。
- StatusInfos – DBInstanceStatusInfo 对象的数组。

  只读副本的状态。如果实例不是只读副本，则此项空白。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  不支持：数据库实例的加密由数据库集群管理。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库实例关联的存储类型。
- TdeCredentialArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。
- Timezone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库实例所属的 VPC 安全组元素的列表。

错误

- DBInstanceNotFoundFault

- InvalidDBInstanceStateFault
- DBSnapshotAlreadyExistsFault
- SnapshotQuotaExceededFault
- InvalidDBClusterStateFault

ModifyDBInstance（操作）

此 API 的 AWS CLI 名称为：`modify-db-instance`。

修改数据库实例的设置。您可通过在请求中指定这些参数以及新值，更改一个或多个数据库配置参数。要了解您可对数据库实例进行哪些修改，请在调用the section called “ModifyDBInstance”之前调用the section called “DescribeValidDBInstanceModifications”。

Request

- AllowMajorVersionUpgrade（在 CLI 中：`--allow-major-version-upgrade`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示允许主要版本升级。更改此参数不会导致中断，所做更改会尽快以异步方式应用。

- ApplyImmediately（在 CLI 中：`--apply-immediately`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否应尽快异步应用此请求中修改及任何待处理修改，无论数据库实例的 `PreferredMaintenanceWindow` 设置如何。

  如果此参数设置为 `false`，则在下一个维护时段中应用对数据库实例的更改。某些参数更改会导致中断，在下次调用 the section called “RebootDBInstance” 时或者下次故障重启时应用。

  默认值：`false`

- AutoMinorVersionUpgrade（在 CLI 中：`--auto-minor-version-upgrade`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示在维护时段期间，将对该数据库实例自动应用次要版本升级。更改此参数不会导致中断（除非在下列情况中），所做更改会尽快以异步方式应用。如果在维护时段内将此参数设置为 `true`，有较新的次要版本可用，并且 Neptune 已启用对该引擎版本的自动修补，则将导致服务中断。

- BackupRetentionPeriod（在 CLI 中：`--backup-retention-period`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

不适用。自动备份保留期由数据库集群管理。有关更多信息，请参阅the section called "ModifyDBCluster"。

默认值：使用现有设置

- CACertificateIdentifier（在 CLI 中：`--ca-certificate-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示需要与实例相关联的证书。

- CloudwatchLogsExportConfiguration（在 CLI 中：`--cloudwatch-logs-export-configuration`）– CloudwatchLogsExportConfiguration 对象。

  要启用日志类型的配置设置，以便针对特定数据库实例或数据库集群导出到 CloudWatch Logs。

- CopyTagsToSnapshot（在 CLI 中：`--copy-tags-to-snapshot`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果为 true，则将数据库实例的所有标签复制到数据库实例的快照中，否则为 false。默认值为 false。

- DBInstanceClass（在 CLI 中：`--db-instance-class`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的新计算和内存容量，例如，`db.m4.large`。并非所有数据库实例类在所有 Amazon 区域中都可用。

  如果您修改数据库实例类，则在更改期间会发生中断。更改在下一个维护时段内应用，除非此请求的 `ApplyImmediately` 指定为 `true`。

  默认值：使用现有设置

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例标识符。此值以一个小写字符串存储。

  约束：

  - 必须匹配现有 DBInstance 的标识符。

- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

要应用到数据库实例的数据库参数组的名称。更改此设置不会导致中断。参数组名称本身将立即发生更改，但在您重新启动实例而不进行故障转移之前，不会应用实际参数更改。在下一个维护时段内，将不自动重启数据库实例，并且不应用参数更改。

默认值：使用现有设置

约束：数据库参数组必须与此数据库实例位于同一个数据库参数组系列中。

- DBPortNumber（在 CLI 中：`--db-port-number`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  数据库实例接受连接的端口编号。

  `DBPortNumber` 参数的值不能与为数据库实例选项组中的选项指定的任何端口值相同。

  不论 `ApplyImmediately` 参数的值如何，在您更改 `DBPortNumber` 值时，数据库将重启。

  默认值：8182

- DBSecurityGroups（在 CLI 中：`--db-security-groups`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  在此数据库实例上授权的数据库安全组的列表。更改此设置不会导致中断，所做更改会尽快以异步方式应用。

  约束：

  - 如果提供，则必须匹配现有 DBSecurityGroups。

- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的新数据库子网组。您可以使用此参数将数据库实例移动到其他 VPC。

  更改子网组将导致更改期间发生中断。更改在下一个维护时段内应用，除非您为 `ApplyImmediately` 参数指定 `true`。

  约束：如果提供，则必须与现有 DBSubnetGroup 的名称匹配。

  例如：`mySubnetGroup`

- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

指示数据库实例是否已启用删除保护的值。在启用删除保护时，无法删除数据库。默认情况下，将禁用删除保护。请参阅删除数据库实例。

- Domain（在 CLI 中：`--domain`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。

- DomainIAMRoleName（在 CLI 中：`--domain-iam-role-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持

- EnableIAMDatabaseAuthentication（在 CLI 中：`--enable-iam-database-authentication`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用 Amazon Identity and Access Management (IAM) 账户与数据库账户之间的映射，则为 true；否则为 false。

  您可以为以下数据库引擎启用 IAM 数据库身份验证

  不适用。将 Amazon IAM 账户映射到数据库账户是由数据库集群管理的。有关更多信息，请参阅the section called “ModifyDBCluster”。

  默认值：`false`

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎要升级到的版本号。目前，设置此参数不起作用。要将数据库引擎升级到最新版本，请使用 the section called “ApplyPendingMaintenanceAction” API。

- Iops（在 CLI 中：`--iops`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  实例的新预配置 IOPS（每秒 I/O 操作数）值。

  更改此设置不会导致中断，除非此请求的 `ApplyImmediately` 参数设置为 `true`，否则将在下个维护时段应用更改。

  默认值：使用现有设置

- MonitoringInterval（在 CLI 中：`--monitoring-interval`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。要禁用收集增强监控指标，请指定 0。默认值是 0。

如果指定 MonitoringRoleArn，则您还必须将 MonitoringInterval 设置为 0 以外的值。

有效值：0, 1, 5, 10, 15, 30, 60

- MonitoringRoleArn（在 CLI 中：--monitoring-role-arn）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。例如，arn:aws:iam:123456789012:role/emaccess。

  如果 MonitoringInterval 设置为 0 以外的值，则您必须提供 MonitoringRoleArn 值。
- MultiAZ（在 CLI 中：--multi-az）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。更改此参数不会导致中断，除非此请求的 ApplyImmediately 参数设置为 true，否则将在下个维护时段应用更改。
- NewDBInstanceIdentifier（在 CLI 中：--new-db-instance-identifier）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  重命名数据库实例时数据库实例的新标识符。在更改数据库实例标识符的时候，若将 Apply Immediately 设置为 true，则实例立即重启；若将 Apply Immediately 设置为 false，则实例将在下一个维护时段重启。此值以一个小写字符串存储。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结束或包含两个连续连字符。

  例如：mydbinstance
- PreferredBackupWindow（在 CLI 中：--preferred-backup-window）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果启用自动备份，该属性指定创建自动备份的日常时间范围。

  不适用。每天创建自动备份的时间范围由数据库集群管理。有关更多信息，请参阅the section called “ModifyDBCluster”。

约束：

- 必须采用 hh24:mi-hh24:mi 格式
- 必须采用协调世界时 (UTC)
- 不得与首选维护时段冲突
- 必须至少为 30 分钟

- PreferredMaintenanceWindow（在 CLI 中：`--preferred-maintenance-window`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  可进行系统维护的每周时间范围（采用 UTC），这可能导致中断。更改此参数不会导致中断（除非在下列情况中），所做更改会尽快以异步方式应用。如果有待处理的操作导致服务重启，并且维护时段经过更改，加入了当前时间，则更改此参数将导致数据库实例重启。如果将此时段移动到当前时间，则当前时间与时段结束之间必须相隔至少 30 分钟以确保应用待处理的更改。

  默认值：使用现有设置

  格式：ddd:hh24:mi-ddd:hh24:mi

  有效值：Mon | Tue | Wed | Thu | Fri | Sat | Sun

  约束：必须至少为 30 分钟

- PromotionTier（在 CLI 中：`--promotion-tier`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

  默认值：1

  有效值：0 - 15

- PubliclyAccessible（在 CLI 中：`--publicly-accessible`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。

- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。

- TdeCredentialArn（在 CLI 中：`--tde-credential-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。

- TdeCredentialPassword（在 CLI 中：`--tde-credential-password`）– 一个敏感字符串，类型为：`string`（UTF-8 编码的字符串）。

  密钥存储中指定 ARN 的密码，以便访问设备。

- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  在此数据库实例上授权的 EC2 VPC 安全组的列表。此更改会尽快异步应用。

  不适用。关联的 EC2 VPC 安全组列表由数据库集群管理。有关更多信息，请参阅the section called “ModifyDBCluster”。

  约束：

  - 如果提供，则必须匹配现有 VpcSecurityGroupIds。

响应

包含 Amazon Neptune 数据库实例的详细信息。

此数据类型用作 the section called “DescribeDBInstances” 操作中的响应元素。

- AutoMinorVersionUpgrade - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示自动应用次要版本补丁。

- AvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库实例所在可用区域的名称。

- BackupRetentionPeriod – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- CACertificateIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例的 CA 证书的标识符。

- CopyTagsToSnapshot - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

指定是否将标签从数据库实例复制到数据库实例的快照。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果数据库实例属于某个数据库集群，则包含数据库实例所属的数据库集群的名称。

- DBInstanceArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的 Amazon 资源名称 (ARN)。

- DBInstanceClass – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含数据库实例的计算和内存容量级别名称。

- DBInstanceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库标识符。此标识符是标识数据库实例的唯一密钥。

- DbInstancePort – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定数据库实例侦听的端口。如果数据库实例属于某个数据库集群，则此端口可能不同于数据库集群的端口。

- DBInstanceStatus – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库的当前状态。

- DbiResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的 Amazon 区域唯一且不变的标识符。只要访问数据库实例的 Amazon KMS 密钥，就可以在 Amazon CloudTrail 日志条目中找到此标识符。

- DBName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库名称。

- DBParameterGroups – DBParameterGroupStatus 对象的数组。

  提供应用到此数据库实例的数据库参数组的列表。

- DBSecurityGroups – DBSecurityGroupMembership 对象的数组。

  提供仅包含 `DBSecurityGroup.Name` 和 `DBSecurityGroup.Status` 子元素的数据库安全组元素的列表。

- DBSubnetGroup – 一个 DBSubnetGroup 对象。

  指定与数据库实例关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护。在启用删除保护时，无法删除实例。请参阅删除数据库实例。

- DomainMemberships – DomainMembership 对象的数组。

  不支持

- EnabledCloudwatchLogsExports – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库实例配置为导出到 CloudWatch Logs 的日志类型的列表。

- Endpoint – 一个 端点 对象。

  指定连接端点。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供要用于此数据库实例的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- EnhancedMonitoringResourceArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  接收数据库实例的增强监控指标数据的 Amazon CloudWatch Logs 日志流的 Amazon 资源名称 (ARN)。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果 Amazon Identity and Access Management (IAM) 身份验证已启用，则为 true；否则为 false。

- InstanceCreateTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供创建数据库实例的日期和时间。

- Iops – IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定预配置的 IOPS（每秒 I/O 操作数）值。

- KmsKeyId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不支持：数据库实例的加密由数据库集群管理。

- LatestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

指定数据库可以使用时间点还原的最新还原时间。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。

- MonitoringInterval – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。

- MonitoringRoleArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。

- PendingModifiedValues – 一个 PendingModifiedValues 对象。

  指定对数据库实例的更改待处理。仅在更改待处理时包含此元素。特定更改由子元素标识。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- PromotionTier – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

- PubliclyAccessible - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。

- ReadReplicaDBClusterIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个数据集集群（作为此数据库实例只读副本的集群）的标识符。

- ReadReplicaDBInstanceIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库实例关联的只读副本的标识符。

- ReadReplicaSourceDBInstanceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

包含源数据库实例（在此数据库实例是只读副本时）的标识符。

- SecondaryAvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果存在，则指定带多可用区支持的数据库实例辅助可用区的名称。

- StatusInfos – DBInstanceStatusInfo 对象的数组。

  只读副本的状态。如果实例不是只读副本，则此项空白。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  不支持：数据库实例的加密由数据库集群管理。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库实例关联的存储类型。

- TdeCredentialArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。

- Timezone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。

- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库实例所属的 VPC 安全组元素的列表。

错误

- InvalidDBInstanceStateFault
- InvalidDBSecurityGroupStateFault
- DBInstanceAlreadyExistsFault
- DBInstanceNotFoundFault
- DBSecurityGroupNotFoundFault
- DBParameterGroupNotFoundFault
- InsufficientDBInstanceCapacityFault
- StorageQuotaExceededFault
- InvalidVPCNetworkStateFault
- ProvisionedIopsNotAvailableInAZFault

- OptionGroupNotFoundFault
- DBUpgradeDependencyFailureFault
- StorageTypeNotSupportedFault
- AuthorizationNotFoundFault
- CertificateNotFoundFault
- DomainNotFoundFault

RebootDBInstance（操作）

此 API 的 AWS CLI 名称为：`reboot-db-instance`。

您可能需要重启数据库实例，通常是出于维护目的。例如，如果进行某些修改或更改与数据库实例关联的数据库参数组，您必须重启该实例以使更改生效。

重启数据库实例会重新启动数据库引擎服务。重启数据库实例将导致短暂中断，在此期间，数据库实例状态将设置为正在重启。

Request

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例标识符。该参数作为一个小写字符串存储。

  约束：

  - 必须匹配现有 DBInstance 的标识符。
- ForceFailover（在 CLI 中：`--force-failover`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果为 `true`，则通过多可用区故障转移进行重启。

  约束：如果没有为多可用区配置实例，则无法指定 `true`。

响应

包含 Amazon Neptune 数据库实例的详细信息。

此数据类型用作 the section called “DescribeDBInstances” 操作中的响应元素。

- AutoMinorVersionUpgrade - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指示自动应用次要版本补丁。

- AvailabilityZone – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库实例所在可用区域的名称。

- BackupRetentionPeriod – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- CACertificateIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库实例的 CA 证书的标识符。

- CopyTagsToSnapshot - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定是否将标签从数据库实例复制到数据库实例的快照。

- DBClusterIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果数据库实例属于某个数据库集群，则包含数据库实例所属的数据库集群的名称。

- DBInstanceArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 资源名称 (ARN)。

- DBInstanceClass – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含数据库实例的计算和内存容量级别名称。

- DBInstanceIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含用户提供的数据库标识符。此标识符是标识数据库实例的唯一密钥。

- DbInstancePort – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定数据库实例侦听的端口。如果数据库实例属于某个数据库集群，则此端口可能不同于数据库集群的端口。

- DBInstanceStatus – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此数据库的当前状态。

- DbiResourceId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的 Amazon 区域唯一且不变的标识符。只要访问数据库实例的 Amazon KMS 密钥，就可以在 Amazon CloudTrail 日志条目中找到此标识符。

- DBName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库名称。

- DBParameterGroups – DBParameterGroupStatus 对象的数组。

  提供应用到此数据库实例的数据库参数组的列表。

- DBSecurityGroups – DBSecurityGroupMembership 对象的数组。

  提供仅包含 `DBSecurityGroup.Name` 和 `DBSecurityGroup.Status` 子元素的数据库安全组元素的列表。

- DBSubnetGroup – 一个 DBSubnetGroup 对象。

  指定与数据库实例关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护。在启用删除保护时，无法删除实例。请参阅删除数据库实例。

- DomainMemberships – DomainMembership 对象的数组。

  不支持

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例配置为导出到 CloudWatch Logs 的日志类型的列表。

- Endpoint – 一个 端点 对象。

  指定连接端点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库实例的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- EnhancedMonitoringResourceArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  接收数据库实例的增强监控指标数据的 Amazon CloudWatch Logs 日志流的 Amazon 资源名称 (ARN)。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果 Amazon Identity and Access Management (IAM) 身份验证已启用，则为 true；否则为 false。

- InstanceCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供创建数据库实例的日期和时间。

- Iops – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定预配置的 IOPS（每秒 I/O 操作数）值。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持：数据库实例的加密由数据库集群管理。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。

- MonitoringInterval – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。

- MonitoringRoleArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。

- PendingModifiedValues – 一个 PendingModifiedValues 对象。

  指定对数据库实例的更改待处理。仅在更改待处理时包含此元素。特定更改由子元素标识。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- PromotionTier – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

- PubliclyAccessible - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  此标记不应再使用。
- ReadReplicaDBClusterIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个数据集集群（作为此数据库实例只读副本的集群）的标识符。
- ReadReplicaDBInstanceIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库实例关联的只读副本的标识符。
- ReadReplicaSourceDBInstanceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含源数据库实例（在此数据库实例是只读副本时）的标识符。
- SecondaryAvailabilityZone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果存在，则指定带多可用区支持的数据库实例辅助可用区的名称。
- StatusInfos – DBInstanceStatusInfo 对象的数组。

  只读副本的状态。如果实例不是只读副本，则此项空白。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  不支持：数据库实例的加密由数据库集群管理。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库实例关联的存储类型。
- TdeCredentialArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。
- Timezone – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库实例所属的 VPC 安全组元素的列表。

错误

- InvalidDBInstanceStateFault

- DBInstanceNotFoundFault

DescribeDBInstances（操作）

此 API 的 AWS CLI 名称为：`describe-db-instances`。

返回有关预配置的实例的信息，并支持分页。

> **Note**
>
> 此操作还可以返回 Amazon RDS 实例和 Amazon DocDB 实例的信息。

Request

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用户提供的实例标识符。如果指定了此参数，则只返回特定数据库实例的信息。该参数不区分大小写。

  约束：

  - 如果提供，必须匹配现有 DBInstance 的标识符。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  筛选条件指定要描述的一个或多个数据库实例。

  支持的筛选条件：

  - `db-cluster-id` - 接受数据库集群标识符和数据库集群 Amazon 资源名称 (ARN)。该结果列表仅包括与这些 ARN 所标识数据库集群关联的数据库实例的相关信息。
  - `engine` - 接受引擎名称（例如 `neptune`），并将结果列表限制为由该引擎创建的数据库实例。

  例如，要从 Amazon CLI 调用此 API 并进行筛选，以便仅返回 Neptune 数据库实例，您可以使用以下命令：

  Example

```
aws neptune describe-db-instances \
            --filters  Name=engine,Values=neptune
```

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBInstances` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

- DBInstances – DBInstance 对象的数组。

  the section called “DBInstance” 实例的列表。

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

错误

- DBInstanceNotFoundFault

DescribeOrderableDBInstanceOptions（操作）

此 API 的 AWS CLI 名称为：`describe-orderable-db-instance-options`。

返回指定引擎的可订购数据库实例选项的列表。

Request

- DBInstanceClass（在 CLI 中：`--db-instance-class`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

数据库实例类筛选值。指定此参数以只显示与指定数据库实例类匹配的可用产品。

- Engine（在 CLI 中：`--engine`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为其检索数据库实例选项的引擎的名称。

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  引擎版本筛选值。指定此参数以只显示与指定引擎版本匹配的可用产品。

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。

- LicenseModel（在 CLI 中：`--license-model`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  许可模式筛选值。指定此参数以只显示与指定许可模式匹配的可用产品。

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeOrderableDBInstanceOptions 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

- Vpc（在 CLI 中：`--vpc`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  VPC 筛选值。指定此参数以只显示可用的 VPC 或非 VPC 产品。

响应

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

由之前的 OrderableDBInstanceOptions 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- OrderableDBInstanceOptions – OrderableDBInstanceOption 对象的数组。

  一个 the section called “OrderableDBInstanceOption” 结构，其中包含有关数据库实例可订购选项的信息。

DescribeValidDBInstanceModifications（操作）

此 API 的 AWS CLI 名称为：`describe-valid-db-instance-modifications`。

您可以调用 the section called “DescribeValidDBInstanceModifications” 来了解可以对数据库实例进行哪些修改。您可在调用 the section called “ModifyDBInstance” 时使用此信息。

Request

- DBInstanceIdentifier（在 CLI 中：`--db-instance-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  客户标识符或数据库实例的 ARN。

响应

有关您可对数据库实例进行的有效更改的信息。包含对 the section called “DescribeValidDBInstanceModifications” 操作的成功调用的结果。您可在调用 the section called “ModifyDBInstance” 时使用此信息。

- Storage – ValidStorageOptions 对象的数组。

  数据库实例的有效存储选项。

错误

- DBInstanceNotFoundFault
- InvalidDBInstanceStateFault

结构：

DBInstance（结构）

包含 Amazon Neptune 数据库实例的详细信息。

此数据类型用作 the section called “DescribeDBInstances” 操作中的响应元素。

字段

- AutoMinorVersionUpgrade - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示自动应用次要版本补丁。
- AvailabilityZone – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库实例所在可用区域的名称。
- BackupRetentionPeriod – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。
- CACertificateIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例的 CA 证书的标识符。
- CopyTagsToSnapshot - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否将标签从数据库实例复制到数据库实例的快照。
- DBClusterIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果数据库实例属于某个数据库集群，则包含数据库实例所属的数据库集群的名称。
- DBInstanceArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的 Amazon 资源名称 (ARN)。
- DBInstanceClass – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含数据库实例的计算和内存容量级别名称。
- DBInstanceIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库标识符。此标识符是标识数据库实例的唯一密钥。
- DbInstancePort – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

指定数据库实例侦听的端口。如果数据库实例属于某个数据库集群，则此端口可能不同于数据库集群的端口。

- DBInstanceStatus – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库的当前状态。

- DbiResourceId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的 Amazon 区域唯一且不变的标识符。只要访问数据库实例的 Amazon KMS 密钥，就可以在 Amazon CloudTrail 日志条目中找到此标识符。

- DBName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库名称。

- DBParameterGroups – 这是 DBParameterGroupStatus 对象数组。

  提供应用到此数据库实例的数据库参数组的列表。

- DBSecurityGroups – 这是 DBSecurityGroupMembership 对象数组。

  提供仅包含 `DBSecurityGroup.Name` 和 `DBSecurityGroup.Status` 子元素的数据库安全组元素的列表。

- DBSubnetGroup –这是一个 DBSubnetGroup 对象。

  指定与数据库实例关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否已启用删除保护。在启用删除保护时，无法删除实例。请参阅删除数据库实例。

- DomainMemberships – 这是 DomainMembership 对象数组。

  不支持

- EnabledCloudwatchLogsExports – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库实例配置为导出到 CloudWatch Logs 的日志类型的列表。

- Endpoint –这是一个 端点 对象。

  指定连接端点。

- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供要用于此数据库实例的数据库引擎的名称。

- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。
- EnhancedMonitoringResourceArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  接收数据库实例的增强监控指标数据的 Amazon CloudWatch Logs 日志流的 Amazon 资源名称 (ARN)。
- IAMDatabaseAuthenticationEnabled - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果 Amazon Identity and Access Management (IAM) 身份验证已启用，则为 true；否则为 false。
- InstanceCreateTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供创建数据库实例的日期和时间。
- Iops – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定预配置的 IOPS（每秒 I/O 操作数）值。
- KmsKeyId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持：数据库实例的加密由数据库集群管理。
- LatestRestorableTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。
- LicenseModel – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  该数据库实例的许可模式信息。
- MonitoringInterval – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  收集数据库实例的增强监控指标的时间点之间的间隔，以秒为单位。
- MonitoringRoleArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  允许 Neptune 将增强监控指标发送到 Amazon CloudWatch Logs 的 IAM 角色的 ARN。
- MultiAZ - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库实例是否为多可用区部署。
- PendingModifiedValues –这是一个 PendingModifiedValues 对象。

指定对数据库实例的更改待处理。仅在更改待处理时包含此元素。特定更改由子元素标识。

- PreferredBackupWindow – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 BackupRetentionPeriod 所规定。

- PreferredMaintenanceWindow – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- PromotionTier – 这是 IntegerOptional，类型为：integer（带符号的 32 位整数）。

  该值指定在现有主实例发生故障后将只读副本提升为主实例的顺序。

- PubliclyAccessible - 这是一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  此标记不应再使用。

- ReadReplicaDBClusterIdentifiers – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含一个或多个数据集集群（作为此数据库实例只读副本的集群）的标识符。

- ReadReplicaDBInstanceIdentifiers – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含一个或多个与此数据库实例关联的只读副本的标识符。

- ReadReplicaSourceDBInstanceIdentifier – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含源数据库实例（在此数据库实例是只读副本时）的标识符。

- SecondaryAvailabilityZone – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果存在，则指定带多可用区支持的数据库实例辅助可用区的名称。

- StatusInfos – 这是 DBInstanceStatusInfo 对象数组。

  只读副本的状态。如果实例不是只读副本，则此项空白。

- StorageEncrypted - 这是一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  不支持：数据库实例的加密由数据库集群管理。

- StorageType – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定与数据库实例关联的存储类型。

- TdeCredentialArn – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

来自密钥存储的 ARN，实例通过它进行关联用于 TDE 加密。

- Timezone – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。

- VpcSecurityGroups – 这是 VpcSecurityGroupMembership 对象数组。

  提供数据库实例所属的 VPC 安全组元素的列表。

`DBInstance` 用作下列对象的响应元素：

- CreateDBInstance
- DeleteDBInstance
- ModifyDBInstance
- RebootDBInstance

DBInstanceStatusInfo（结构）

提供数据库实例状态信息的列表。

字段

- Message – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  实例出现错误时，错误的详细信息。如果实例未处于出错状态，则此值为空白。

- Normal - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  布尔值，如果实例正常运行则为 true，如果实例处于错误状态则为 false。

- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的状态。对于只读副本的 StatusType，该值可以为 replicating、error、stopped 或 terminated。

- StatusType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此值当前是“只读复制”。

OrderableDBInstanceOption（结构）

包含数据库实例的可用选项列表。

此数据类型用作 the section called “DescribeOrderableDBInstanceOptions” 操作中的响应元素。

字段

- AvailabilityZones – 这是 AvailabilityZone 对象数组。

  数据库实例的可用区域列表。
- DBInstanceClass – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的数据库实例类。
- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的引擎类型。
- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的引擎版本。
- LicenseModel – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的许可模式
- MaxIopsPerDbInstance – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  数据库实例的最大总预配置 IOPS。
- MaxIopsPerGib – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

  数据库实例每 Gib 的最大预配置 IOPS。
- MaxStorageSize – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  数据库实例的最大存储大小。
- MinIopsPerDbInstance – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  数据库实例的最小总预配置 IOPS。
- MinIopsPerGib – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

  数据库实例每 Gib 的最小预配置 IOPS。
- MinStorageSize – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

数据库实例的最小存储大小。

- MultiAZCapable - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否有多可用区。

- ReadReplicaCapable - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否可以有只读副本。

- StorageType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。

- SupportsEnhancedMonitoring - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否支持增强监控（间隔从 1 到 60 秒）。

- SupportsGlobalDatabases - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，表示是否可以将 Neptune 全球数据库与其它数据库引擎属性的特定组合一起使用。

- SupportsIAMDatabaseAuthentication - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否支持 IAM 数据库身份验证。

- SupportsIops - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否支持预配置的 IOPS。

- SupportsStorageEncryption - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否支持加密存储。

- Vpc - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库实例是否在 VPC 中。

PendingModifiedValues（结构）

此数据类型用作 the section called “ModifyDBInstance” 操作中的响应元素。

字段

- AllocatedStorage – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

包含将要应用或当前正在应用的数据库实例的新 AllocatedStorage 大小。

- BackupRetentionPeriod – 这是 IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定保留自动备份的挂起天数。

- CACertificateIdentifier – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库实例的 CA 证书的标识符。

- DBInstanceClass – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含将要应用或当前正在应用的数据库实例的新 DBInstanceClass。

- DBInstanceIdentifier – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含将要应用或当前正在应用的数据库实例的新 DBInstanceIdentifier。

- DBSubnetGroupName – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库实例的新数据库子网组。

- EngineVersion – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- Iops – 这是 IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定将应用或当前正在应用的数据库实例的新预配置 IOPS 值。

- MultiAZ – 这是 BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示单可用区数据库实例将更改为多可用区部署。

- PendingCloudwatchLogsExports –这是一个 PendingCloudwatchLogsExports 对象。

  此 PendingCloudwatchLogsExports 结构指定了已启用和已禁用 CloudWatch Logs 的待处理更改。

- Port – 这是 IntegerOptional，类型为：integer（带符号的 32 位整数）。

  指定数据库实例的挂起端口。

- StorageType – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。

ValidStorageOptions（结构）

不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。

字段

- IopsToStorageRatio – 这是 DoubleRange 对象数组。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。
- ProvisionedIops – 这是 范围 对象数组。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。
- StorageSize – 这是 范围 对象数组。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。
- StorageType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不适用。在 Neptune 中，存储类型在数据库集群级别进行管理。

ValidDBInstanceModificationsMessage（结构）

有关您可对数据库实例进行的有效更改的信息。包含对 the section called “DescribeValidDBInstanceModifications” 操作的成功调用的结果。您可在调用 the section called “ModifyDBInstance” 时使用此信息。

字段

- Storage – 这是 ValidStorageOptions 对象数组。

  数据库实例的有效存储选项。

`ValidDBInstanceModificationsMessage` 用作下列对象的响应元素：

- DescribeValidDBInstanceModifications

Neptune 参数 API

操作:

- CopyDBParameterGroup（操作）
- CopyDBClusterParameterGroup（操作）
- CreateDBParameterGroup（操作）
- CreateDBClusterParameterGroup（操作）
- DeleteDBParameterGroup（操作）
- DeleteDBClusterParameterGroup（操作）
- ModifyDBParameterGroup（操作）
- ModifyDBClusterParameterGroup（操作）
- ResetDBParameterGroup（操作）
- ResetDBClusterParameterGroup（操作）
- DescribeDBParameters（操作）
- DescribeDBParameterGroups（操作）
- DescribeDBClusterParameters（操作）
- DescribeDBClusterParameterGroups（操作）
- DescribeEngineDefaultParameters（操作）
- DescribeEngineDefaultClusterParameters（操作）

结构：

- 参数（结构）
- DBParameterGroup（结构）
- DBClusterParameterGroup（结构）
- DBParameterGroupStatus（结构）

CopyDBParameterGroup（操作）

此 API 的 AWS CLI 名称为：`copy-db-parameter-group`。

复制指定的数据库参数组。

请求

- SourceDBParameterGroupIdentifier（在 CLI 中：`--source-db-parameter-group-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

源数据库参数组的标识符或 ARN。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。

约束：

- 必须指定有效的数据库参数组。
- 必须指定有效的数据库参数组标识符，例如 `my-db-param-group`，或者有效的 ARN。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给复制数据库参数组的标签。

- TargetDBParameterGroupDescription（在 CLI 中：`--target-db-parameter-group-description`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  复制的数据库参数组的描述。

- TargetDBParameterGroupIdentifier（在 CLI 中：`--target-db-parameter-group-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  复制的数据库参数组的标识符。

  约束：

  - 不能为 null 或空。
  - 必须包含 1 到 255 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结束或包含两个连续连字符。

  示例：`my-db-parameter-group`

响应

包含 Amazon Neptune 数据库参数组的详细信息。

此数据类型用作 the section called “DescribeDBParameterGroups” 操作中的响应元素。

- DBParameterGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的 Amazon 资源名称 (ARN)。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库参数组兼容的数据库参数组系列的名称。

- DBParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库参数组的名称。

- Description – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库参数组提供客户指定的描述。

错误

- DBParameterGroupNotFoundFault
- DBParameterGroupAlreadyExistsFault
- DBParameterGroupQuotaExceededFault

CopyDBClusterParameterGroup（操作）

此 API 的 AWS CLI 名称为：`copy-db-cluster-parameter-group`。

复制指定的数据库集群参数组。

请求

- SourceDBClusterParameterGroupIdentifier（在 CLI 中：`--source-db-cluster-parameter-group-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  源数据库集群参数组的标识符或 Amazon 资源名称 (ARN)。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。

  约束：

  - 必须指定有效的数据库集群参数组。
  - 如果源数据库集群参数组与副本位于相同的 Amazon 区域中，请指定有效的数据库参数组标识符（例如 `my-db-cluster-param-group`）或有效的 ARN。
  - 如果源数据库参数组与副本位于不同的 Amazon 区域中，请指定有效的数据库集群参数组 ARN，例如 `arn:aws:rds:us-east-1:123456789012:cluster-pg:custom-cluster-group1`。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给复制数据库集群参数组的标签。

- TargetDBClusterParameterGroupDescription（在 CLI 中：`--target-db-cluster-parameter-group-description`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  复制的数据库集群参数组的描述。

- TargetDBClusterParameterGroupIdentifier（在 CLI 中：`--target-db-cluster-parameter-group-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  复制的数据库集群参数组的标识符。

  约束：

  - 不能为 null 或空
  - 必须包含 1 到 255 个字母、数字或连字符
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符

  示例：`my-cluster-param-group1`

响应

包含 Amazon Neptune 数据库集群参数组的详细信息。

此数据类型用作 the section called “DescribeDBClusterParameterGroups” 操作中的响应元素。

- DBClusterParameterGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的 Amazon 资源名称 (ARN)。

- DBClusterParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群参数组的名称。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群参数组兼容的数据库参数组系列的名称。

- Description – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库集群参数组提供客户指定的描述。

错误

- DBParameterGroupNotFoundFault
- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

CreateDBParameterGroup（操作）

此 API 的 AWS CLI 名称为：`create-db-parameter-group`。

创建新的数据库参数组。

最初创建数据库参数组时，使用的是数据库实例所用数据库引擎的默认参数。要向任何参数提供自定义值，您必须在创建之后使用 ModifyDBParameterGroup 对其进行修改。创建数据库参数组后，您需要使用 ModifyDBInstance 将它关联到数据库实例。将新数据库参数组关联到正在运行的数据库实例时，您需要重新启动数据库实例而不进行故障转移，以使新数据库参数组和关联设置生效。

> **Important**
>
> 创建数据库参数组之后，您应至少等待 5 分钟，再创建使用该数据库参数组作为默认参数组的第一个数据库实例。这样，在将参数组用作新数据库实例的默认设置之前，Amazon Neptune 可以完成全部创建操作。这对于在为数据库实例创建默认数据库时十分关键的参数 (例如，由 `character_set_database` 参数定义的默认数据库字符集) 非常重要。您可以使用 Amazon Neptune 控制台的 Parameter Groups (参数组) 选项或使用 DescribeDBParameters 命令来验证是否已创建或修改数据库参数组。

请求

- DBParameterGroupFamily（在 CLI 中：`--db-parameter-group-family`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组系列名称。数据库参数组能且仅能与一个数据库参数组系列关联，只能应用到运行数据库引擎且引擎版本与该数据库参数组系列兼容的数据库实例上。
- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的名称。

约束：

- 必须为 1 到 255 个字母、数字或连字符。
- 第一个字符必须是字母
- 不能以连字符结束或包含两个连续连字符

> **Note**
>
> 此值以一个小写字符串存储。

- Description（在 CLI 中：`--description`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的描述。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给新数据库参数组的标签。

响应

包含 Amazon Neptune 数据库参数组的详细信息。

此数据类型用作 the section called “DescribeDBParameterGroups” 操作中的响应元素。

- DBParameterGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的 Amazon 资源名称 (ARN)。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库参数组兼容的数据库参数组系列的名称。

- DBParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库参数组的名称。

- Description – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库参数组提供客户指定的描述。

错误

- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

CreateDBClusterParameterGroup（操作）

此 API 的 AWS CLI 名称为：`create-db-cluster-parameter-group`。

创建新的数据库集群参数组。

数据库集群参数组中的参数应用到数据库集群中的所有实例。

最初创建数据库集群参数组时，使用的是数据库集群中实例所用数据库引擎的默认参数。为了提供针对任何参数的自定义值，您必须在创建缓存参数组之后使用the section called “ModifyDBClusterParameterGroup”对其进行修改。创建数据库集群参数组后，您需要使用the section called “ModifyDBCluster”将它关联到数据库集群。将新数据库集群参数组关联到正在运行的数据库集群时，您需要重新启动数据库集群中的数据库实例而不进行故障转移，以使新数据库集群参数组和关联设置生效。

> **Important**
>
> 创建数据库集群参数组之后，您应至少等待 5 分钟，再创建使用该数据库集群参数组作为默认参数组的第一个数据库集群。这样，在将数据库集群参数组用作新数据库集群的默认设置之前，Amazon Neptune 可以完成全部创建操作。这对于在为数据库集群创建默认数据库时十分关键的参数（例如，由 `character_set_database` 参数定义的默认数据库字符集）非常重要。您可以使用 Amazon Neptune 控制台的参数组选项或 the section called “DescribeDBClusterParameters” 命令来验证是否已创建或修改数据库集群参数组。

请求

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的名称。

  约束：

  - 必须与现有 DBClusterParameterGroup 的名称匹配。

> **Note**
>
> 此值以一个小写字符串存储。

- DBParameterGroupFamily（在 CLI 中：`--db-parameter-group-family`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组系列名称。数据库集群参数组能且仅能与一个数据库集群参数组系列关联，只能应用到运行数据库引擎且引擎版本与该数据库集群参数组系列兼容的数据库集群上。

- Description（在 CLI 中：`--description`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的描述。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给新数据库集群参数组的标签。

响应

包含 Amazon Neptune 数据库集群参数组的详细信息。

此数据类型用作 the section called “DescribeDBClusterParameterGroups” 操作中的响应元素。

- DBClusterParameterGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的 Amazon 资源名称 (ARN)。

- DBClusterParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群参数组的名称。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群参数组兼容的数据库参数组系列的名称。

- Description – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库集群参数组提供客户指定的描述。

错误

- DBParameterGroupQuotaExceededFault
- DBParameterGroupAlreadyExistsFault

DeleteDBParameterGroup（操作）

此 API 的 AWS CLI 名称为：`delete-db-parameter-group`。

删除指定的 DBParameterGroup。要添加的 DBParameterGroup 不能与任何数据库实例关联。

请求

- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的名称。

  约束：

  - 必须是现有数据库参数组的名称
  - 您无法删除默认数据库参数组
  - 不能与任何数据库实例关联

响应

- 无响应参数。

错误

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

DeleteDBClusterParameterGroup（操作）

此 API 的 AWS CLI 名称为：`delete-db-cluster-parameter-group`。

删除指定的数据库集群参数组。要添加的数据库集群参数组不能与任何数据库集群关联。

请求

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的名称。

  约束：

  - 必须是现有数据库集群参数组的名称。
  - 您无法删除默认的数据库集群参数组。
  - 不能与任何数据库集群关联。

响应

- 无响应参数。

错误

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

ModifyDBParameterGroup（操作）

此 API 的 AWS CLI 名称为：`modify-db-parameter-group`。

修改数据库参数组的参数。要修改多个参数，请提交以下对象的列表：`ParameterName`、`ParameterValue` 和 `ApplyMethod`。在单个请求中，最多可以修改 20 个参数。

> **Note**
>
> 对动态参数所做的更改将立即应用。对静态参数所做的更改需要在重启（不执行故障转移）与参数组关联的数据库实例后才生效。

> **Important**
>
> 修改数据库参数组之后，您应至少等待 5 分钟，再创建使用该数据库参数组作为默认参数组的第一个数据库实例。这样，在将参数组用作新数据库实例的默认设置之前，Amazon Neptune 可以完成全部修改操作。这对于在为数据库实例创建默认数据库时十分关键的参数 (例如，由 `character_set_database` 参数定义的默认数据库字符集) 非常重要。您可以使用 Amazon Neptune 控制台的 Parameter Groups (参数组) 选项或使用 DescribeDBParameters 命令来验证是否已创建或修改数据库参数组。

请求

- DBParameterGroupName (在 CLI 中：`--db-parameter-group-name`) – 必需：一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  数据库参数组的名称。

  约束：

  - 如果提供，必须与现有 DBParameterGroup 的名称匹配。
- Parameters (在 CLI 中：`--parameters`) - 必需：参数 对象的数组。

  参数名称、值以及参数更新应用方法的数组。必须至少提供一个参数名称、值和应用方法；后续参数是可选的。在单个请求中，最多可以修改 20 个参数。

  有效值 (对于应用方法) ：`immediate | pending-reboot`

  > **Note**
  >
  > 您只能使用具有动态参数的即时值。您可以为动态参数和静态参数使用等待重启值，在您重启数据库实例而不进行故障转移时应用更改。

响应

- DBParameterGroupName – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  提供数据库参数组的名称。

错误

- DBParameterGroupNotFoundFault
- InvalidDBParameterGroupStateFault

ModifyDBClusterParameterGroup（操作）

此 API 的 AWS CLI 名称为：`modify-db-cluster-parameter-group`。

修改数据库集群参数组的参数。要修改多个参数，请提交以下对象的列表：`ParameterName`、`ParameterValue` 和 `ApplyMethod`。在单个请求中，最多可以修改 20 个参数。

> **Note**
>
> 对动态参数所做的更改将立即应用。对静态参数所做的更改需要在重启（不执行故障转移）与参数组关联的数据库集群后才生效。

> **Important**
>
> 创建数据库集群参数组之后，您应至少等待 5 分钟，再创建使用该数据库集群参数组作为默认参数组的第一个数据库集群。这样，在将参数组用作新数据库集群的默认设置之前，Amazon Neptune 可以完成全部创建操作。这对于在为数据库集群创建默认数据库时十分关键的参数（例如，由 `character_set_database` 参数定义的默认数据库字符集）非常重要。您可以使用 Amazon Neptune 控制台的 Parameter Groups (参数组) 选项或the section called “DescribeDBClusterParameters”命令来验证是否已创建或修改数据库集群参数组。

请求

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要修改的数据库集群参数组的名称。
- Parameters（在 CLI 中：`--parameters`）- 必需：参数 对象的数组。

  要修改的数据库集群参数组中的参数列表。

响应

- DBClusterParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的名称。

  约束：

  - 必须为 1 到 255 个字母或数字。
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符

  > **Note**
  >
  > 此值以一个小写字符串存储。

错误

- DBParameterGroupNotFoundFault
- InvalidDBParameterGroupStateFault

ResetDBParameterGroup（操作）

此 API 的 AWS CLI 名称为：`reset-db-parameter-group`。

将数据库参数组的参数修改为引擎/系统默认值。要重置特定参数，请提供以下内容的列表：`ParameterName` 和 `ApplyMethod`。要重置整个数据库参数组，请指定 `DBParameterGroup` 名称和 `ResetAllParameters` 参数。重置整个组时，动态参数立即更新，静态参数设置为 `pending-reboot` 以便在下次数据库实例重新启动或 `RebootDBInstance` 请求时生效。

请求

- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的名称。

  约束：

  - 必须与现有 DBParameterGroup 的名称匹配。

- Parameters：（在 CLI 中：`--parameters`）参数 对象的数组。

  要重置整个数据库参数组，请指定 `DBParameterGroup` 名称和 `ResetAllParameters` 参数。要重置特定参数，请提供以下内容的列表：`ParameterName` 和 `ApplyMethod`。在单个请求中，最多可以修改 20 个参数。

  有效值（对于应用方法）：`pending-reboot`

- ResetAllParameters（在 CLI 中：`--reset-all-parameters`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定将数据库参数组中的所有参数重置为默认值 (`true`) 还是不重置 (`false`)。

  默认值：`true`

响应

- DBParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库参数组的名称。

错误

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

ResetDBClusterParameterGroup（操作）

此 API 的 AWS CLI 名称为：`reset-db-cluster-parameter-group`。

将数据库集群参数组的参数修改为默认值。要重置特定参数，请提交以下内容的列表：`ParameterName` 和 `ApplyMethod`。要重置整个数据库集群参数组，请指定 `DBClusterParameterGroupName` 和 `ResetAllParameters` 参数。

重置整个组时，动态参数立即更新，静态参数设置为 `pending-reboot` 以便在下次数据库实例重新启动或 the section called “RebootDBInstance” 请求时生效。对于数据库集群中您希望将更新的静态参数应用到的每个数据库实例，您必须调用 the section called “RebootDBInstance”。

请求

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要重置的数据库集群参数组的名称。

- Parameters：（在 CLI 中：`--parameters`）参数 对象的数组。

  数据库集群参数组中要重置为默认值的参数名称列表。如果 `ResetAllParameters` 参数设置为 `true`，则无法使用此参数。

- ResetAllParameters（在 CLI 中：`--reset-all-parameters`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  值设置为 `true` 可将数据库集群参数组中的所有参数重置为其默认值，否则为 `false`。如果为 `Parameters` 参数指定了参数名称列表，您无法使用此参数。

响应

- DBClusterParameterGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的名称。

  约束：

  - 必须为 1 到 255 个字母或数字。
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符

> **Note**
>
> 此值以一个小写字符串存储。

错误

- InvalidDBParameterGroupStateFault
- DBParameterGroupNotFoundFault

DescribeDBParameters（操作）

此 API 的 AWS CLI 名称为：`describe-db-parameters`。

返回特定数据库参数组的详细参数列表。

请求

- DBParameterGroupName（在 CLI 中：`--db-parameter-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要返回其详细信息的特定数据库参数组的名称。

  约束：

  - 如果提供，必须与现有 DBParameterGroup 的名称匹配。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBParameters` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。
- Source（在 CLI 中：`--source`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要返回的参数类型。

  默认值：返回所有参数类型

  有效值：`user | system | engine-default`

响应

- Marker – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 MaxRecords 指定的值。

- Parameters – 参数 对象的数组。

  the section called “参数” 值的列表。

错误

- DBParameterGroupNotFoundFault

DescribeDBParameterGroups（操作）

此 API 的 AWS CLI 名称为：describe-db-parameter-groups。

返回 DBParameterGroup 描述的列表。如果指定了 DBParameterGroupName，则列表中只包含指定数据库参数组的描述。

请求

- DBParameterGroupName（在 CLI 中：--db-parameter-group-name）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要返回其详细信息的特定数据库参数组的名称。

  约束：

  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。
- Filters：（在 CLI 中：--filters）筛选条件 对象的数组。

  当前不支持此参数。

- Marker（在 CLI 中：--marker）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  由之前的 DescribeDBParameterGroups 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 MaxRecords 指定的值。

- MaxRecords（在 CLI 中：--max-records）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

包括在响应中的最大记录数。如果存在的记录数超过了指定的MaxRecords 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

默认值：100

约束：最低为 20，最高为 100。

响应

- DBParameterGroups – DBParameterGroup 对象的数组。

  the section called “DBParameterGroup” 实例的列表。
- Marker – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 MaxRecords 指定的值。

错误

- DBParameterGroupNotFoundFault

DescribeDBClusterParameters（操作）

此 API 的 AWS CLI 名称为：describe-db-cluster-parameters。

返回特定数据库集群参数组的详细参数列表。

请求

- DBClusterParameterGroupName（在 CLI 中：--db-cluster-parameter-group-name）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  要返回其参数详细信息的特定数据库集群参数组的名称。

  约束：
  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。
- Filters：（在 CLI 中：--filters）筛选条件 对象的数组。

  当前不支持此参数。

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBClusterParameters` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。
- Source（在 CLI 中：`--source`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示仅返回特定源的参数的值。参数源可以是 `engine`、`service` 或 `customer`。

响应

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeDBClusterParameters 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- Parameters – 参数 对象的数组。

  提供数据库集群参数组的参数列表。

错误

- DBParameterGroupNotFoundFault

DescribeDBClusterParameterGroups（操作）

此 API 的 AWS CLI 名称为：`describe-db-cluster-parameter-groups`。

返回 `DBClusterParameterGroup` 描述的列表。如果指定了 `DBClusterParameterGroupName` 参数，则列表中只包含指定数据库集群参数组的描述。

请求

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要返回其详细信息的特定数据库集群参数组的名称。

  约束：

  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBClusterParameterGroups` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

- DBClusterParameterGroups – DBClusterParameterGroup 对象的数组。

  数据库集群参数组的列表。
- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBClusterParameterGroups` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

错误

- DBParameterGroupNotFoundFault

DescribeEngineDefaultParameters（操作）

此 API 的 AWS CLI 名称为：`describe-engine-default-parameters`。

返回指定数据库引擎的默认引擎和系统参数信息。

请求

- DBParameterGroupFamily（在 CLI 中：`--db-parameter-group-family`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组系列的名称。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeEngineDefaultParameters` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

包含 the section called “DescribeEngineDefaultParameters” 操作的成功调用的结果。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定引擎默认参数应用到的数据库参数组系列的名称。
- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 EngineDefaults 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- Parameters – 参数 对象的数组。

  包含引擎默认参数的列表。

DescribeEngineDefaultClusterParameters（操作）

此 API 的 AWS CLI 名称为：`describe-engine-default-cluster-parameters`。

返回集群数据库引擎的默认引擎和系统参数信息。

请求

- DBParameterGroupFamily（在 CLI 中：`--db-parameter-group-family`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要返回其引擎参数信息的数据库集群参数组系列的名称。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeEngineDefaultClusterParameters` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

包含 the section called “DescribeEngineDefaultParameters” 操作的成功调用的结果。

- DBParameterGroupFamily – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定引擎默认参数应用到的数据库参数组系列的名称。

- Marker – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  由之前的 EngineDefaults 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 MaxRecords 指定的值。
- Parameters – 参数 对象的数组。

  包含引擎默认参数的列表。

结构：

参数（结构）

指定参数。

字段

- AllowedValues – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定参数值的有效范围。
- ApplyMethod – 这是 ApplyMethod，类型为：string（UTF-8 编码的字符串）。

  指示何时应用参数更新。
- ApplyType – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定引擎特定的参数类型。
- DataType – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定参数的有效数据类型。
- Description – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供参数的说明。
- IsModifiable - 这是一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指示参数可以修改 (true) 还是不能修改 (false)。一些参数具有安全或操作影响，会阻止更改这些参数。
- MinimumEngineVersion – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  参数可以应用到的最早引擎版本。
- ParameterName – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

指定参数的名称。

- ParameterValue – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定参数的值。
- Source – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示参数值的源。

DBParameterGroup（结构）

包含 Amazon Neptune 数据库参数组的详细信息。

此数据类型用作 the section called “DescribeDBParameterGroups” 操作中的响应元素。

字段

- DBParameterGroupArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的 Amazon 资源名称 (ARN)。
- DBParameterGroupFamily – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库参数组兼容的数据库参数组系列的名称。
- DBParameterGroupName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库参数组的名称。
- Description – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库参数组提供客户指定的描述。

`DBParameterGroup` 用作下列对象的响应元素：

- CopyDBParameterGroup
- CreateDBParameterGroup

DBClusterParameterGroup（结构）

包含 Amazon Neptune 数据库集群参数组的详细信息。

此数据类型用作 the section called “DescribeDBClusterParameterGroups” 操作中的响应元素。

字段

- DBClusterParameterGroupArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群参数组的 Amazon 资源名称 (ARN)。
- DBClusterParameterGroupName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群参数组的名称。
- DBParameterGroupFamily – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群参数组兼容的数据库参数组系列的名称。
- Description – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为此数据库集群参数组提供客户指定的描述。

`DBClusterParameterGroup` 用作下列对象的响应元素：

- CopyDBClusterParameterGroup
- CreateDBClusterParameterGroup

DBParameterGroupStatus（结构）

数据库参数组的状态。

此数据类型用作下列操作中的响应元素：

- the section called “CreateDBInstance”
- the section called “DeleteDBInstance”
- the section called “ModifyDBInstance”
- the section called “RebootDBInstance”

字段

- DBParameterGroupName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库参数组的名称。

- ParameterApplyStatus – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  参数更新的状态。

Neptune 子网 API

操作:

- CreateDBSubnetGroup（操作）
- DeleteDBSubnetGroup（操作）
- ModifyDBSubnetGroup（操作）
- DescribeDBSubnetGroups（操作）

结构：

- 子网（结构）
- DBSubnetGroup（结构）

CreateDBSubnetGroup（操作）

此 API 的 AWS CLI 名称为：`create-db-subnet-group`。

创建新的数据库子网组。数据库子网组必须在 Amazon 区域的至少两个可用区中包含至少一个子网。

请求

- DBSubnetGroupDescription（在 CLI 中：`--db-subnet-group-description`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的描述。
- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的名称。此值以一个小写字符串存储。

  约束：必须包含不超过 255 个字母、数字、句点、下划线、空格或连字符。不能是默认值。

  示例：`mySubnetgroup`

- SubnetIds（在 CLI 中：`--subnet-ids`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的 EC2 子网 ID。
- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给新数据库子网组的标签。

响应

包含 Amazon Neptune 数据库子网组的详细信息。

此数据类型用作 the section called “DescribeDBSubnetGroups” 操作中的响应元素。

- DBSubnetGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的 Amazon 资源名称 (ARN)。
- DBSubnetGroupDescription – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库子网组的描述。
- DBSubnetGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的名称。
- SubnetGroupStatus – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库子网组的状态。
- Subnets – 子网 对象的数组。

  包含 the section called “子网” 元素的列表。
- VpcId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库子网组的 VpcId。

错误

- DBSubnetGroupAlreadyExistsFault
- DBSubnetGroupQuotaExceededFault
- DBSubnetQuotaExceededFault

- DBSubnetGroupDoesNotCoverEnoughAZs
- InvalidSubnet

DeleteDBSubnetGroup（操作）

此 API 的 AWS CLI 名称为：`delete-db-subnet-group`。

删除数据库子网组。

> **Note**
>
> 指定的数据库子网组不得与任何数据库实例关联。

请求

- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的数据库子网组的名称。

  > **Note**
  >
  > 您不能删除默认子网组。

  约束：

  约束：必须与现有 DBSubnetGroup 的名称匹配。不能是默认值。

  示例：`mySubnetgroup`

响应

- 无响应参数。

错误

- InvalidDBSubnetGroupStateFault

- InvalidDBSubnetStateFault
- DBSubnetGroupNotFoundFault

ModifyDBSubnetGroup（操作）

此 API 的 AWS CLI 名称为：`modify-db-subnet-group`。

修改现有的数据库子网组。数据库子网组必须在 Amazon 区域的至少两个可用区中包含至少一个子网。

请求

- DBSubnetGroupDescription（在 CLI 中：`--db-subnet-group-description`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的描述。
- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的名称。此值以一个小写字符串存储。您不能修改默认子网组。

  约束：必须与现有 DBSubnetGroup 的名称匹配。不能是默认值。

  示例：`mySubnetgroup`
- SubnetIds（在 CLI 中：`--subnet-ids`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的 EC2 子网 ID。

响应

包含 Amazon Neptune 数据库子网组的详细信息。

此数据类型用作 the section called “DescribeDBSubnetGroups” 操作中的响应元素。

- DBSubnetGroupArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的 Amazon 资源名称 (ARN)。
- DBSubnetGroupDescription – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

提供数据库子网组的描述。

- DBSubnetGroupName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库子网组的名称。

- SubnetGroupStatus – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库子网组的状态。

- Subnets – 子网 对象的数组。

  包含 the section called “子网” 元素的列表。

- VpcId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库子网组的 VpcId。

错误

- DBSubnetGroupNotFoundFault
- DBSubnetQuotaExceededFault
- SubnetAlreadyInUse
- DBSubnetGroupDoesNotCoverEnoughAZs
- InvalidSubnet

DescribeDBSubnetGroups（操作）

此 API 的 AWS CLI 名称为：`describe-db-subnet-groups`。

返回 DBSubnetGroup 描述的列表。如果指定了 DBSubnetGroupName，则列表中只包含指定 DBSubnetGroup 的描述。

要查看 CIDR 范围概览，请访问 Wikipedia Tutorial。

请求

- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要为其返回详细信息的数据库子网组的名称。

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeDBSubnetGroups 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

- DBSubnetGroups – DBSubnetGroup 对象的数组。

  the section called “DBSubnetGroup” 实例的列表。

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

错误

- DBSubnetGroupNotFoundFault

结构：

子网（结构）

指定一个子网。

此数据类型用作 the section called “DescribeDBSubnetGroups” 操作中的响应元素。

字段

- SubnetAvailabilityZone –这是一个 AvailabilityZone 对象。

  指定子网所在的 EC2 可用区。
- SubnetIdentifier – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定子网的标识符。
- SubnetStatus – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定子网的状态。

DBSubnetGroup（结构）

包含 Amazon Neptune 数据库子网组的详细信息。

此数据类型用作 the section called “DescribeDBSubnetGroups” 操作中的响应元素。

字段

- DBSubnetGroupArn – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库子网组的 Amazon 资源名称 (ARN)。
- DBSubnetGroupDescription – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库子网组的描述。
- DBSubnetGroupName – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库子网组的名称。
- SubnetGroupStatus – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库子网组的状态。
- Subnets – 这是 子网 对象数组。

  包含 the section called “子网” 元素的列表。
- VpcId – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库子网组的 VpcId。

DBSubnetGroup 用作下列对象的响应元素：

- CreateDBSubnetGroup
- ModifyDBSubnetGroup

Neptune 快照 API

操作：

- CreateDBClusterSnapshot（操作）
- DeleteDBClusterSnapshot（操作）
- CopyDBClusterSnapshot（操作）
- ModifyDBClusterSnapshotAttribute（操作）
- RestoreDBClusterFromSnapshot（操作）
- RestoreDBClusterToPointInTime（操作）
- DescribeDBClusterSnapshots（操作）
- DescribeDBClusterSnapshotAttributes（操作）

结构：

- DBClusterSnapshot（结构）
- DBClusterSnapshotAttribute（结构）
- DBClusterSnapshotAttributesResult（结构）

CreateDBClusterSnapshot（操作）

> 此 API 的 AWS CLI 名称为：`create-db-cluster-snapshot`。

创建数据库集群的快照。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

要为其创建快照的数据库集群的标识符。此参数不区分大小写。

约束：

- 必须匹配现有 DBCluster 的标识符。

例如：`my-cluster1`

- DBClusterSnapshotIdentifier（在 CLI 中：`--db-cluster-snapshot-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群快照的标识符。该参数作为一个小写字符串存储。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符。
  - 第一个字符必须是字母。
  - 不能以连字符结束或包含两个连续连字符。

  例如：`my-cluster1-snapshot1`
- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给数据库集群快照的标签。

响应

包含 Amazon Neptune 数据库集群快照的详细信息

此数据类型用作 the section called “DescribeDBClusterSnapshots” 操作中的响应元素。

- AllocatedStorage – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定分配的存储大小（以 GiB 为单位）。
- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群快照中的实例可以还原到的 EC2 可用区列表。
- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。
- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

指定从中创建了此数据库集群快照的数据库集群的标识符。

- DBClusterSnapshotArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群快照的 Amazon 资源名称 (ARN)。

- DBClusterSnapshotIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群快照的标识符。必须与现有快照的标识符匹配。

  在使用 `DBClusterSnapshotIdentifier` 还原数据库集群后，您必须为该数据库集群的任何将来更新指定相同的 `DBClusterSnapshotIdentifier`。当您为更新指定此属性时，不会再次从快照还原数据库群集，并且数据库中的数据也不会更改。

  不过，如果您不指定 `DBClusterSnapshotIdentifier`，则会创建空数据库集群，并删除原始数据库集群。如果指定的属性与以前的快照还原属性不同，则从 `DBClusterSnapshotIdentifier` 指定的快照中还原数据库集群，并删除原始数据库集群。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群快照的数据库引擎的版本。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群快照的 Amazon KMS 密钥标识符。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群快照的许可模式信息。

- PercentProgress – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定估计的已传输数据百分比。

- Port – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定获取快照时数据库集群侦听的端口。

- SnapshotCreateTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供获取快照的时间，以通用协调时 (UTC) 表示。
- SnapshotType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库集群快照的类型。
- SourceDBClusterSnapshotArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果数据库集群快照复制自源数据库集群快照，则为源数据库集群快照的 Amazon 资源名称 (ARN)，否则为 null 值。
- Status – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此数据库集群快照的状态。
- StorageEncrypted - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定是否加密数据库集群快照。
- StorageType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  与数据库集群快照关联的存储类型。
- VpcId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供与数据库集群快照关联的 VPC ID。

错误

- DBClusterSnapshotAlreadyExistsFault
- InvalidDBClusterStateFault
- DBClusterNotFoundFault
- SnapshotQuotaExceededFault
- InvalidDBClusterSnapshotStateFault

DeleteDBClusterSnapshot（操作）

此 API 的 AWS CLI 名称为：delete-db-cluster-snapshot。

删除数据库集群快照。如果正在复制快照，则复制操作将会终止。

> **Note**
>
> 只有处于 `available` 状态的数据库集群快照才能删除。

Request

- DBClusterSnapshotIdentifier（在 CLI 中：`--db-cluster-snapshot-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的数据库集群快照的标识符。

  约束：必须为处于 `available` 状态的现有数据库集群快照的名称。

响应

包含 Amazon Neptune 数据库集群快照的详细信息

此数据类型用作 the section called “DescribeDBClusterSnapshots” 操作中的响应元素。

- AllocatedStorage – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定分配的存储大小（以 GiB 为单位）。
- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群快照中的实例可以还原到的 EC2 可用区列表。
- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。
- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定从中创建了此数据库集群快照的数据库集群的标识符。
- DBClusterSnapshotArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群快照的 Amazon 资源名称 (ARN)。
- DBClusterSnapshotIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群快照的标识符。必须与现有快照的标识符匹配。

在使用 DBClusterSnapshotIdentifier 还原数据库集群后，您必须为该数据库集群的任何将来更新指定相同的 DBClusterSnapshotIdentifier。当您为更新指定此属性时，不会再次从快照还原数据库群集，并且数据库中的数据也不会更改。

不过，如果您不指定 DBClusterSnapshotIdentifier，则会创建空数据库集群，并删除原始数据库集群。如果指定的属性与以前的快照还原属性不同，则从 DBClusterSnapshotIdentifier 指定的快照中还原数据库集群，并删除原始数据库集群。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供此数据库集群快照的数据库引擎的版本。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- KmsKeyId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果 StorageEncrypted 为 true，则为加密数据库集群快照的 Amazon KMS 密钥标识符。

- LicenseModel – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供此数据库集群快照的许可模式信息。

- PercentProgress – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定估计的已传输数据百分比。

- Port – 一个整数，类型为：integer（带符号的 32 位整数）。

  指定获取快照时数据库集群侦听的端口。

- SnapshotCreateTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供获取快照的时间，以通用协调时 (UTC) 表示。

- SnapshotType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库集群快照的类型。

- SourceDBClusterSnapshotArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

如果数据库集群快照复制自源数据库集群快照，则为源数据库集群快照的 Amazon 资源名称 (ARN)，否则为 null 值。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群快照的状态。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否加密数据库集群快照。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与数据库集群快照关联的存储类型。

- VpcId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供与数据库集群快照关联的 VPC ID。

错误

- InvalidDBClusterSnapshotStateFault
- DBClusterSnapshotNotFoundFault

CopyDBClusterSnapshot（操作）

此 API 的 AWS CLI 名称为：`copy-db-cluster-snapshot`。

复制数据库集群的快照。

要从共享的手动数据库集群快照复制数据库集群快照，`SourceDBClusterSnapshotIdentifier` 必须为共享数据库集群快照的 Amazon 资源名称 (ARN)。

Request

- CopyTags（在 CLI 中：`--copy-tags`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果为 true，则将源数据库集群快照的所有标签复制到目标数据库集群快照；否则为 false。默认值为 false。

- KmsKeyId（在 CLI 中：`--kms-key-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

加密数据库集群快照的 Amazon KMS 密钥 ID。KMS 密钥 ID 是 Amazon 资源名称 (ARN)、KMS 密钥标识符或 KMS 加密密钥的 KMS 密钥别名。

如果您从 Amazon 账户复制加密的数据库集群快照，则可以为 KmsKeyId 指定值来使用新的 KMS 加密密钥加密副本。如果您不为 KmsKeyId 指定值，则使用与源数据库集群快照相同的 KMS 密钥来加密数据库集群快照的副本。

如果您复制从其它 Amazon 账户共享的加密数据库集群快照，则必须为 KmsKeyId 指定值。

KMS 加密密钥是特定于创建它们的 Amazon 区域的，您无法将一个 Amazon 区域的加密密钥用于另一个 Amazon 区域。

您无法在复制时加密未加密的数据库集群快照。如果您尝试复制未加密的数据库集群快照并为 KmsKeyId 参数指定值，则会返回错误。

- PreSignedUrl（在 CLI 中：--pre-signed-url）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  当前不支持。

- SourceDBClusterSnapshotIdentifier（在 CLI 中：--source-db-cluster-snapshot-identifier）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  要复制的数据库集群快照的标识符。此参数不区分大小写。

  约束：

  - 必须指定处于“available”状态的有效系统快照。
  - 指定有效的数据库快照标识符。

  例如：my-cluster-snapshot1

- Tags：（在 CLI 中：--tags）标签 对象的数组。

  分配到新数据库集群快照副本的标签。

- TargetDBClusterSnapshotIdentifier（在 CLI 中：--target-db-cluster-snapshot-identifier）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  要从源数据库集群快照创建的新数据库集群快照标识符。此参数不区分大小写。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符。

- 第一个字符必须是字母。
- 不能以连字符结束或包含两个连续连字符。

例如：`my-cluster-snapshot2`

响应

包含 Amazon Neptune 数据库集群快照的详细信息

此数据类型用作 the section called “DescribeDBClusterSnapshots” 操作中的响应元素。

- AllocatedStorage – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定分配的存储大小（以 GiB 为单位）。
- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群快照中的实例可以还原到的 EC2 可用区列表。
- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。
- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定从中创建了此数据库集群快照的数据库集群的标识符。
- DBClusterSnapshotArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群快照的 Amazon 资源名称 (ARN)。
- DBClusterSnapshotIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群快照的标识符。必须与现有快照的标识符匹配。

  在使用 `DBClusterSnapshotIdentifier` 还原数据库集群后，您必须为该数据库集群的任何将来更新指定相同的 `DBClusterSnapshotIdentifier`。当您为更新指定此属性时，不会再次从快照还原数据库群集，并且数据库中的数据也不会更改。

  不过，如果您不指定 `DBClusterSnapshotIdentifier`，则会创建空数据库集群，并删除原始数据库集群。如果指定的属性与以前的快照还原属性不同，则从 `DBClusterSnapshotIdentifier` 指定的快照中还原数据库集群，并删除原始数据库集群。
- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

指定数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群快照的数据库引擎的版本。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群快照的 Amazon KMS 密钥标识符。

- LicenseModel – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供此数据库集群快照的许可模式信息。

- PercentProgress – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定估计的已传输数据百分比。

- Port – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定获取快照时数据库集群侦听的端口。

- SnapshotCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供获取快照的时间，以通用协调时 (UTC) 表示。

- SnapshotType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群快照的类型。

- SourceDBClusterSnapshotArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果数据库集群快照复制自源数据库集群快照，则为源数据库集群快照的 Amazon 资源名称 (ARN)，否则为 null 值。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群快照的状态。

- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否加密数据库集群快照。

- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与数据库集群快照关联的存储类型。
- VpcId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供与数据库集群快照关联的 VPC ID。

错误

- DBClusterSnapshotAlreadyExistsFault
- DBClusterSnapshotNotFoundFault
- InvalidDBClusterStateFault
- InvalidDBClusterSnapshotStateFault
- SnapshotQuotaExceededFault
- KMSKeyNotAccessibleFault

ModifyDBClusterSnapshotAttribute（操作）

此 API 的 AWS CLI 名称为：`modify-db-cluster-snapshot-attribute`。

向手动数据库集群快照添加属性和值，或者从中删除属性和值。

要与其它 Amazon 账户共享手动数据库集群快照，请指定 `restore` 作为 `AttributeName`，并使用 `ValuesToAdd` 参数添加已获得授权可还原手动数据库集群快照的 Amazon 账户的 ID 列表。使用值 `all` 以公开手动数据库集群快照，这意味着所有 Amazon 账户都可以复制或还原它。若任何手动数据库集群快照包含您不想向所有 Amazon 账户公开的私有信息，则不要添加 `all` 值。如果手动数据库集群快照已加密，则可以共享它，但只能通过为 `ValuesToAdd` 参数指定已授权 Amazon 账户 ID 列表来共享。在这种情况下，您不能使用 `all` 作为该参数的值。

要查看哪些 Amazon 账户有权复制或还原手动数据库集群快照，或者手动数据库集群快照为公有还是私有，请使用the section called “DescribeDBClusterSnapshotAttributes” API 操作。

Request

- AttributeName（在 CLI 中：`--attribute-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要修改的数据库集群快照属性的名称。

要管理其它 Amazon 账户复制或还原数据库集群快照的授权，请将此值设置为 `restore`。

- DBClusterSnapshotIdentifier（在 CLI 中：`--db-cluster-snapshot-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要修改其属性的数据库集群快照的标识符。

- ValuesToAdd（在 CLI 中：`--values-to-add`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要添加到 `AttributeName` 所指定属性中的数据库集群快照属性的列表。

  要授权其它 Amazon 账户复制或还原手动数据库集群快照，请设置此列表，在其中包含一个或多个 Amazon 账户 ID，或者设置为 `all` 以使手动数据库集群快照可由所有 Amazon 账户还原。若任何手动数据库集群快照包含您不想向所有 Amazon 账户公开的私有信息，则不要添加 `all` 值。

- ValuesToRemove（在 CLI 中：`--values-to-remove`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要从 `AttributeName` 所指定属性中删除的数据库集群快照属性的列表。

  要删除其它 Amazon 账户复制或还原手动数据库集群快照的授权，请设置此列表，在其中包含一个或多个 Amazon 账户标识符，或者设置为 `all` 以删除所有 Amazon 账户复制或还原数据库集群快照的授权。如果您指定 `all`，则其账户 ID 显式添加到 `restore` 属性的 Amazon 账户仍可以复制或还原手动数据库集群快照。

响应

包含成功调用the section called “DescribeDBClusterSnapshotAttributes” API 操作的结果。

手动数据库集群快照属性用于授权其它 Amazon 账户复制或还原手动数据库集群快照。有关更多信息，请参阅 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

- DBClusterSnapshotAttributes – DBClusterSnapshotAttribute 对象的数组。

  手动数据库集群快照的属性和值的列表。

- DBClusterSnapshotIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  属性应用到的手动数据库集群快照的标识符。

错误

- DBClusterSnapshotNotFoundFault
- InvalidDBClusterSnapshotStateFault
- SharedSnapshotQuotaExceededFault

RestoreDBClusterFromSnapshot（操作）

此 API 的 AWS CLI 名称为：`restore-db-cluster-from-snapshot`。

从数据库快照或数据库集群快照创建新的数据库集群。

如果指定数据库快照，则使用默认配置和默认安全组，从源数据库快照创建目标数据库集群。

如果指定数据库集群快照，则使用具有与原始源数据库集群相同配置的源数据库集群恢复点，创建目标数据库集群，不同之处在于新数据库集群在默认安全组中创建。

Request

- AvailabilityZones（在 CLI 中：`--availability-zones`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其中创建所还原数据库集群的实例的 EC2 可用区列表。
- CopyTagsToSnapshot（在 CLI 中：`--copy-tags-to-snapshot`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的已还原数据库集群的任何快照中。
- DatabaseName（在 CLI 中：`--database-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不支持。
- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  从数据库快照或数据库集群快照创建的数据库集群的名称。该参数不区分大小写。

  约束：
  - 必须包含 1 到 63 个字母、数字或连字符
  - 第一个字符必须是字母

- 不能以连字符结束或包含两个连续连字符

  例如：`my-snapshot-id`

- DBClusterParameterGroupName（在 CLI 中：`--db-cluster-parameter-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要与新数据库集群关联的数据库集群参数组的名称。

  约束：

  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。

- DBSubnetGroupName（在 CLI 中：`--db-subnet-group-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于新数据库集群的数据库子网组的名称。

  约束：如果提供，则必须与现有 DBSubnetGroup 的名称匹配。

  例如：`mySubnetgroup`

- DeletionProtection（在 CLI 中：`--deletion-protection`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。默认情况下，将禁用删除保护。

- EnableCloudwatchLogsExports（在 CLI 中：`--enable-cloudwatch-logs-exports`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要导出到 Amazon CloudWatch Logs 的已还原数据库集群的日志列表。

- EnableIAMDatabaseAuthentication（在 CLI 中：`--enable-iam-database-authentication`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用 Amazon Identity and Access Management (IAM) 账户与数据库账户之间的映射，则为 true；否则为 false。

  默认值：`false`

- Engine（在 CLI 中：`--engine`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于新数据库集群的数据库引擎。

默认值：与源相同

约束：必须与源的引擎兼容

- EngineVersion（在 CLI 中：`--engine-version`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于新数据库集群的数据库引擎的版本。

- KmsKeyId（在 CLI 中：`--kms-key-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  从数据库快照或数据库集群快照还原加密数据库集群时使用的 Amazon KMS 密钥标识符。

  KMS 密钥标识符是 KMS 加密密钥的 Amazon 资源名称 (ARN)。如果您还原数据库集群所用的 Amazon 的账户，与拥有用于加密新数据库集群所用 KMS 密钥的账户相同，则您可以使用 KMS 密钥别名而不是 KMS 加密密钥的 ARN。

  如果您未指定 `KmsKeyId` 参数的值，则会出现以下情况：

  - 如果 `SnapshotIdentifier` 中的数据库快照或数据库集群快照已加密，将使用用于加密数据库快照或数据库集群快照的同一 KMS 密钥加密还原的数据库集群。
  - 如果 `SnapshotIdentifier` 中的数据库快照或数据库集群快照未加密，还原的数据库集群也不会加密。

- Port（在 CLI 中：`--port`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  新数据库集群用于接受连接的端口号。

  约束：值必须为 `1150-65535`

  默认值：与原始数据库集群相同的端口。

- ServerlessV2ScalingConfiguration（在 CLI 中：`--serverless-v2-scaling-configuration`）– ServerlessV2ScalingConfiguration 对象。

  包含 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- SnapshotIdentifier（在 CLI 中：`--snapshot-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要从中进行还原的数据库快照或数据库集群快照的标识符。

您可以使用名称或 Amazon 资源名称（ARN） 指定数据库集群快照。但是，您只能使用 ARN 指定数据库快照。

约束：

- 必须与现有快照的标识符匹配。

- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定要与数据库集群关联的存储类型。

  有效值：`standard`、`iopt1`

  默认值：`standard`

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要分配给所还原数据库集群的标签。

- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新数据库集群所属的 VPC 安全组的列表。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。

- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：audit（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：string（UTF-8 编码的字符串）。

指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

指定在启用自动备份时，自动执行备份的日常时间范围，如 BackupRetentionPeriod 所规定。

- PreferredMaintenanceWindow – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。

- ReaderEndpoint – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。

- ReadReplicaIdentifiers – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。

- ReplicationSourceIdentifier – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ReplicationType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  不受 Neptune 支持。

- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- Status – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。

- StorageEncrypted - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指定数据库集群是否已加密。

- StorageType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

  有效值：

  - **standard** -（默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。

- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。
- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterAlreadyExistsFault
- DBClusterQuotaExceededFault
- StorageQuotaExceededFault
- DBSubnetGroupNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterSnapshotNotFoundFault
- InsufficientDBClusterCapacityFault
- InsufficientStorageClusterCapacityFault
- InvalidDBSnapshotStateFault
- InvalidDBClusterSnapshotStateFault
- StorageQuotaExceededFault
- InvalidVPCNetworkStateFault
- InvalidRestoreFault
- DBSubnetGroupNotFoundFault
- InvalidSubnet
- OptionGroupNotFoundFault
- KMSKeyNotAccessibleFault
- DBClusterParameterGroupNotFoundFault

RestoreDBClusterToPointInTime（操作）

此 API 的 AWS CLI 名称为：`restore-db-cluster-to-point-in-time`。

将数据库集群还原到任意时间点。用户可以还原到 LatestRestorableTime 之前最多 BackupRetentionPeriod 天的任意时间点。使用具有与原始数据库集群相同配置的源数据库集群创建目标数据库集群，不同之处在于新数据库集群在默认数据库安全组中创建。

> **Note**
>
> 此操作仅还原数据库集群，而不还原该数据库集群的数据库实例。您必须调用the section called “CreateDBInstance”操作为还原的数据库集群创建数据库实例，并在 DBClusterIdentifier 中指定还原的数据库集群的标识符。只有在完成 RestoreDBClusterToPointInTime 操作并且数据库集群可用后，您才能创建数据库实例。

Request

- DBClusterIdentifier（在 CLI 中：--db-cluster-identifier）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  要创建的新数据库集群的名称。

  约束：

  - 必须包含 1 到 63 个字母、数字或连字符
  - 第一个字符必须是字母
  - 不能以连字符结束或包含两个连续连字符
- DBClusterParameterGroupName（在 CLI 中：--db-cluster-parameter-group-name）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要与新数据库集群关联的数据库集群参数组的名称。

  约束：

  - 如果提供，必须与现有 DBClusterParameterGroup 的名称匹配。
- DBSubnetGroupName（在 CLI 中：--db-subnet-group-name）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要用于新数据库集群的数据库子网组名称。

  约束：如果提供，则必须与现有 DBSubnetGroup 的名称匹配。

例如：mySubnetgroup

- DeletionProtection（在 CLI 中：--deletion-protection）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  一个值，指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。默认情况下，将禁用删除保护。

- EnableCloudwatchLogsExports（在 CLI 中：--enable-cloudwatch-logs-exports）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要导出到 CloudWatch Logs 的已还原数据库集群的日志列表。

- EnableIAMDatabaseAuthentication（在 CLI 中：--enable-iam-database-authentication）– BooleanOptional，类型为：boolean [布尔值（true 或 false）]。

  如果启用 Amazon Identity and Access Management (IAM) 账户与数据库账户之间的映射，则为 true；否则为 false。

  默认值：false

- KmsKeyId（在 CLI 中：--kms-key-id）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  从加密数据库集群快照还原加密数据库集群时使用的 Amazon KMS 密钥标识符。

  KMS 密钥标识符是 KMS 加密密钥的 Amazon 资源名称 (ARN)。如果您还原数据库集群所用的 Amazon 的账户，与拥有用于加密新数据库集群所用 KMS 密钥的账户相同，则您可以使用 KMS 密钥别名而不是 KMS 加密密钥的 ARN。

  您可以还原到新数据库集群，并使用与加密源数据库集群所用密钥不同的 KMS 密钥，加密新数据库集群。新数据库集群使用由 KmsKeyId 参数确定的 KMS 密钥加密。

  如果您未指定 KmsKeyId 参数的值，则会出现以下情况：

  - 如果数据库集群已加密，将使用用于加密源数据库集群的同一 KMS 密钥加密还原的数据库集群。
  - 如果数据库集群未加密，还原的数据库集群也不会加密。

  如果 DBClusterIdentifier 引用未加密的数据库集群，则将拒绝恢复请求。

- Port（在 CLI 中：--port）– IntegerOptional，类型为：integer（带符号的 32 位整数）。

  新数据库集群用于接受连接的端口号。

约束：值必须为 1150-65535

默认值：与原始数据库集群相同的端口。

- RestoreToTime（在 CLI 中：`--restore-to-time`）– TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  要将数据库集群还原到的日期和时间。

  有效值：值必须为通用协调时间 (UTC) 格式的时间

  约束：

  - 必须在数据库实例的最新可还原时间之前
  - 如果未提供 `UseLatestRestorableTime` 参数，则必须指定
  - 如果 `UseLatestRestorableTime` 参数为 true，则无法指定
  - 如果 `RestoreType` 参数为 `copy-on-write`，则无法指定

  例如：`2015-03-07T23:45:00Z`

- RestoreType（在 CLI 中：`--restore-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要执行的还原类型。可以指定以下值之一：

  - `full-copy` – 新数据库集群作为源数据库集群的完整副本还原。
  - `copy-on-write` – 新数据库集群作为源数据库集群的克隆还原。

  如果您没有指定 `RestoreType` 值，则新数据库集群作为源数据库集群的完整副本还原。

- ServerlessV2ScalingConfiguration（在 CLI 中：`--serverless-v2-scaling-configuration`）– ServerlessV2ScalingConfiguration 对象。

  包含 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

- SourceDBClusterIdentifier（在 CLI 中：`--source-db-cluster-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要从中还原的源数据库集群的标识符。

  约束：

  - 必须匹配现有 DBCluster 的标识符。
- StorageType（在 CLI 中：`--storage-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定要与数据库集群关联的存储类型。

  有效值：`standard`、`iopt1`

  默认值：`standard`
- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要应用到所还原数据库集群的标签。
- UseLatestRestorableTime（在 CLI 中：`--use-latest-restorable-time`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  值设置为 `true` 时，将数据库集群还原到最新可还原的备份时间，否则为 `false`。

  默认值：`false`

  约束：如果未提供 `RestoreToTime` 参数，则无法指定。
- VpcSecurityGroupIds（在 CLI 中：`--vpc-security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新数据库集群所属的 VPC 安全组的列表。

响应

包含 Amazon Neptune 数据库集群的详细信息。

此数据类型用作 the section called “DescribeDBClusters” 中的响应元素。

- AllocatedStorage – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  `AllocatedStorage` 始终返回 1，因为 Neptune 数据库集群存储大小不固定，而是会根据需要自动调整。
- AssociatedRoles – DBClusterRole 对象的数组。

  提供与数据库集群关联的 Amazon Identity and Access Management（IAM）角色的列表。与数据库集群关联的 IAM 角色授予数据库集群代表您访问其他 Amazon 服务的权限。

- AutomaticRestartTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  数据库集群将自动重启的时间。

- AvailabilityZones – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供可在其上创建数据库集群中实例的 EC2 可用区的列表。

- BacktrackConsumedChangeRecords – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BacktrackWindow – longOptional，类型为：`long`（带符号的 64 位整数）。

  不受 Neptune 支持。

- BackupRetentionPeriod – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  指定自动数据库快照的保留天数。

- Capacity – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  不受 Neptune 支持。

- CloneGroupId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识数据库集群与之关联的克隆组。

- ClusterCreateTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。

- CopyTagsToSnapshot – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 *`true`*，则标签将复制到所创建的数据库集群的任何快照中。

- CrossAccountClone – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `true`，则可以跨账户克隆数据库集群。

- DatabaseName – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含此数据库集群在创建时提供的初始数据库的名称（如果在创建数据库集群时指定了初始数据库）。在数据库集群的使用期间会始终返回同一名称。

- DBClusterArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

数据库集群的 Amazon 资源名称 (ARN)。

- DBClusterIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含用户提供的数据库集群标识符。此标识符是识别数据库集群的唯一键。

- DBClusterMembers – DBClusterMember 对象的数组。

  提供组成数据库集群的实例的列表。

- DBClusterParameterGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的数据库集群参数组名称。

- DbClusterResourceId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群在 Amazon 区域中唯一的不可变标识符。只要访问了数据库集群的 Amazon KMS 密钥，就可在 Amazon CloudTrail 日志条目中找到此标识符。

- DBSubnetGroup – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定与数据库集群关联的子网组的信息，包括名称、描述和子网组中的子网。

- DeletionProtection – BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库集群是否已启用删除保护。在启用删除保护时，无法删除数据库。

- EarliestBacktrackTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  不受 Neptune 支持。

- EarliestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最早还原时间。

- EnabledCloudwatchLogsExports – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据库集群配置为导出到 CloudWatch Logs 的日志类型的列表。有效的日志类型有：`audit`（将审计日志发布到 CloudWatch）和慢速查询（将慢速查询日志发布到 CloudWatch）。请参阅将 Neptune 日志发布到 Amazon CloudWatch Logs。

- Endpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群的主实例的连接终端节点。

- Engine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

提供要用于此数据库集群的数据库引擎的名称。

- EngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示数据库引擎版本。

- GlobalClusterIdentifier – GlobalClusterIdentifier，类型为：`string`（UTF-8 编码的字符串），不小于 1 或大于 255，与以下正则表达式匹配：`[A-Za-z][0-9A-Za-z-:._]*`。

  包含用户提供的全球数据库集群标识符。此标识符是标识全球数据库的唯一键。

- HostedZoneId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。

- IAMDatabaseAuthenticationEnabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- IOOptimizedNextAllowedModificationTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  下次您可以修改数据库集群以使用 `iopt1` 存储类型。

- KmsKeyId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 `StorageEncrypted` 为 true，则为加密数据库集群的 Amazon KMS 密钥标识符。

- LatestRestorableTime – TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定数据库可以使用时间点还原的最新还原时间。

- MultiAZ - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否在多个可用区中有实例。

- PendingModifiedValues – 一个 ClusterPendingModifiedValues 对象。

  此数据类型用作 `ModifyDBCluster` 操作中的响应元素，包含将在下一个维护时段期间应用的更改。

- PercentProgress – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定操作的进度百分比。

- Port – IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

指定数据库引擎侦听的端口。

- PreferredBackupWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定在启用自动备份时，自动执行备份的日常时间范围，如 `BackupRetentionPeriod` 所规定。
- PreferredMaintenanceWindow – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定可进行系统维护的每周时间范围（采用通用协调时间 (UTC)）。
- ReaderEndpoint – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群的读取器端点。数据库集群的读取器终端节点跨数据库集群中可用的只读副本，实现连接的负载均衡。当客户端请求与读取器终端节点的新连接时，Neptune 将在数据库集群中的只读副本之间分配连接请求。该功能可帮助跨数据库集群中的多个只读副本平衡读取工作负载。

  如果发生了故障转移并且连接到的只读副本将提升到主实例，则将删除您的连接。要继续向集群中的其他只读副本发送读取工作负载，您可以随后重新连接到读取器终端节点。
- ReadReplicaIdentifiers – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含一个或多个与此数据库集群关联的只读副本的标识符。
- ReplicationSourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ReplicationType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不受 Neptune 支持。
- ServerlessV2ScalingConfiguration – 一个 ServerlessV2ScalingConfigurationInfo 对象。

  显示 Neptune 无服务器数据库集群的扩展配置。

  有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群的当前状态。
- StorageEncrypted - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定数据库集群是否已加密。
- StorageType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群所使用的存储类型。

有效值：

- **`standard`** - （默认）为 I/O 使用量适中到较小的应用程序提供经济实惠的数据库存储。
- **`iopt1`** - 启用 I/O 优化型存储，其符合 I/O 密集型图形工作负载的需求，且价格可预测，I/O 延迟低，I/O 吞吐量稳定。

  Neptune I/O 优化型存储仅从引擎版本 1.3.0.0 开始可用。

- VpcSecurityGroups – VpcSecurityGroupMembership 对象的数组。

  提供数据库集群所属的 VPC 安全组的列表。

错误

- DBClusterAlreadyExistsFault
- DBClusterNotFoundFault
- DBClusterQuotaExceededFault
- DBClusterSnapshotNotFoundFault
- DBSubnetGroupNotFoundFault
- InsufficientDBClusterCapacityFault
- InsufficientStorageClusterCapacityFault
- InvalidDBClusterSnapshotStateFault
- InvalidDBClusterStateFault
- InvalidDBSnapshotStateFault
- InvalidRestoreFault
- InvalidSubnet
- InvalidVPCNetworkStateFault
- KMSKeyNotAccessibleFault
- OptionGroupNotFoundFault
- StorageQuotaExceededFault
- DBClusterParameterGroupNotFoundFault

DescribeDBClusterSnapshots（操作）

此 API 的 AWS CLI 名称为：`describe-db-cluster-snapshots`。

返回有关数据库集群快照的信息。此 API 操作支持分页。

Request

- DBClusterIdentifier（在 CLI 中：`--db-cluster-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要为其检索数据库集群快照列表的数据库集群的 ID。此参数不能与 `DBClusterSnapshotIdentifier` 参数结合使用。此参数不区分大小写。

  约束：

  - 如果提供，则必须与现有 DBCluster 的标识符匹配。
- DBClusterSnapshotIdentifier（在 CLI 中：`--db-cluster-snapshot-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要描述的特定数据库集群快照标识符。此参数不能与 `DBClusterIdentifier` 参数结合使用。此值以一个小写字符串存储。

  约束：

  - 如果提供，则必须与现有 DBClusterSnapshot 的标识符匹配。
  - 如果此标识符用于自动快照，则还必须指定 `SnapshotType` 参数。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- IncludePublic（在 CLI 中：`--include-public`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果为 true，则包括可由所有 Amazon 账户复制或还原的公有手动数据库集群快照，否则为 false。默认为 `false`。默认值为 false。

  您可使用 the section called “ModifyDBClusterSnapshotAttribute” API 操作，将手动数据库集群快照作为公有来共享。
- IncludeShared（在 CLI 中：`--include-shared`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

如果为 true，则包括来自其它 Amazon 账户且此 Amazon 账户已获得授权可复制或还原的共享手动数据库集群快照，否则为 false。默认为 `false`。

您可以通过 the section called “ModifyDBClusterSnapshotAttribute” API 操作，授予 Amazon 账户权限来还原其它 Amazon 账户的手动数据库集群快照。

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribeDBClusterSnapshots` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

- SnapshotType（在 CLI 中：`--snapshot-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要返回的数据库集群快照的类型。可以指定以下值之一：

  - `automated` - 返回由 Amazon Neptune 为您的 Amazon 账户自动获取的所有数据库集群快照。
  - `manual` - 返回您的 Amazon 账户获取的所有数据库集群快照。
  - `shared` - 返回与您 Amazon 账户共享的所有手动数据库集群快照。
  - `public` – 返回已标记为公有的所有数据库集群快照。

  如果您未指定 `SnapshotType` 值，则返回自动和手动数据库集群快照。您可以通过将 `IncludeShared` 参数设置为 `true`，在这些结果中包括共享数据库集群快照。您可以通过将 `IncludePublic` 参数设置为 `true`，在这些结果中包括公有数据库集群快照。

  `IncludeShared` 和 `IncludePublic` 参数不适用于 `SnapshotType` 的值 `manual` 或 `automated`。`SnapshotType` 设置为 `shared` 时，`IncludePublic` 参数不适用。`SnapshotType` 设置为 `public` 时，`IncludeShared` 参数不适用。

响应

- DBClusterSnapshots – DBClusterSnapshot 对象的数组。

  提供用户的数据库集群快照的列表。

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 the section called “DescribeDBClusterSnapshots” 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

错误

- DBClusterSnapshotNotFoundFault

DescribeDBClusterSnapshotAttributes（操作）

此 API 的 AWS CLI 名称为：`describe-db-cluster-snapshot-attributes`。

返回手动数据库集群快照的数据库集群快照属性名称和值的列表。

与其它 Amazon 账户共享快照时，`DescribeDBClusterSnapshotAttributes` 返回 `restore` 属性以及已获授权可复制或还原手动数据库集群快照的 Amazon 账户的 ID 列表。如果 `restore` 属性的值列表中包含了 `all`，则手动数据库集群快照为公有，可以由所有 Amazon 账户复制或还原。

要为 Amazon 账户添加或删除用于复制或还原手动数据库集群快照的权限，或者使手动数据库集群快照为公有或私有，请使用 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

Request

- DBClusterSnapshotIdentifier（在 CLI 中：`--db-cluster-snapshot-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要描述其属性的数据库集群快照的标识符。

响应

包含成功调用the section called “DescribeDBClusterSnapshotAttributes” API 操作的结果。

手动数据库集群快照属性用于授权其它 Amazon 账户复制或还原手动数据库集群快照。有关更多信息，请参阅 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

- DBClusterSnapshotAttributes – DBClusterSnapshotAttribute 对象的数组。

手动数据库集群快照的属性和值的列表。

- DBClusterSnapshotIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  属性应用到的手动数据库集群快照的标识符。

错误

- DBClusterSnapshotNotFoundFault

结构：

DBClusterSnapshot（结构）

包含 Amazon Neptune 数据库集群快照的详细信息

此数据类型用作 the section called “DescribeDBClusterSnapshots” 操作中的响应元素。

字段

- AllocatedStorage – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  指定分配的存储大小（以 GiB 为单位）。
- AvailabilityZones – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供数据库集群快照中的实例可以还原到的 EC2 可用区列表。
- ClusterCreateTime – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定创建数据库集群的时间，采用通用协调时间 (UTC)。
- DBClusterIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定从中创建了此数据库集群快照的数据库集群的标识符。
- DBClusterSnapshotArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库集群快照的 Amazon 资源名称 (ARN)。
- DBClusterSnapshotIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定数据库集群快照的标识符。必须与现有快照的标识符匹配。

在使用 DBClusterSnapshotIdentifier 还原数据库集群后，您必须为该数据库集群的任何将来更新指定相同的 DBClusterSnapshotIdentifier。当您为更新指定此属性时，不会再次从快照还原数据库群集，并且数据库中的数据也不会更改。

不过，如果您不指定 DBClusterSnapshotIdentifier，则会创建空数据库集群，并删除原始数据库集群。如果指定的属性与以前的快照还原属性不同，则从 DBClusterSnapshotIdentifier 指定的快照中还原数据库集群，并删除原始数据库集群。

- Engine – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库引擎的名称。

- EngineVersion – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供此数据库集群快照的数据库引擎的版本。

- IAMDatabaseAuthenticationEnabled - 这是一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果启用了 Amazon Identity and Access Management（IAM）账户与数据库账户之间的映射，则为 true；否则为 false。

- KmsKeyId – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  如果 StorageEncrypted 为 true，则为加密数据库集群快照的 Amazon KMS 密钥标识符。

- LicenseModel – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供此数据库集群快照的许可模式信息。

- PercentProgress – 这是一个整数，类型为：integer（带符号的 32 位整数）。

  指定估计的已传输数据百分比。

- Port – 这是一个整数，类型为：integer（带符号的 32 位整数）。

  指定获取快照时数据库集群侦听的端口。

- SnapshotCreateTime – 这是 TStamp，类型为：timestamp（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  提供获取快照的时间，以通用协调时 (UTC) 表示。

- SnapshotType – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  提供数据库集群快照的类型。

- SourceDBClusterSnapshotArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果数据库集群快照复制自源数据库集群快照，则为源数据库集群快照的 Amazon 资源名称 (ARN)，否则为 null 值。
- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定此数据库集群快照的状态。
- StorageEncrypted - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指定是否加密数据库集群快照。
- StorageType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与数据库集群快照关联的存储类型。
- VpcId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供与数据库集群快照关联的 VPC ID。

`DBClusterSnapshot` 用作下列对象的响应元素：

- CreateDBClusterSnapshot
- CopyDBClusterSnapshot
- DeleteDBClusterSnapshot

DBClusterSnapshotAttribute（结构）

包含手动数据库集群快照属性的名称和值。

手动数据库集群快照属性用于授权其它 Amazon 账户还原手动数据库集群快照。有关更多信息，请参阅 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

字段

- AttributeName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  手动数据库集群快照属性的名称。

  名为 `restore` 的属性引用有权复制或还原手动数据库集群快照的 Amazon 账户列表。有关更多信息，请参阅 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

- AttributeValues – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  手动数据库集群快照属性的值。

  如果 `AttributeName` 字段设置为 `restore`，则此元素返回已获授权可复制或还原手动数据库集群快照的 Amazon 账户列表。如果列表中有值 `all`，则手动数据库集群快照为公有，可供任意 Amazon 账户复制或还原。

DBClusterSnapshotAttributesResult（结构）

包含成功调用the section called “DescribeDBClusterSnapshotAttributes” API 操作的结果。

手动数据库集群快照属性用于授权其它 Amazon 账户复制或还原手动数据库集群快照。有关更多信息，请参阅 the section called “ModifyDBClusterSnapshotAttribute” API 操作。

字段

- DBClusterSnapshotAttributes – 这是 DBClusterSnapshotAttribute 对象数组。

  手动数据库集群快照的属性和值的列表。

- DBClusterSnapshotIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  属性应用到的手动数据库集群快照的标识符。

`DBClusterSnapshotAttributesResult` 用作下列对象的响应元素：

- DescribeDBClusterSnapshotAttributes
- ModifyDBClusterSnapshotAttribute

Neptune 事件 API

操作:

- CreateEventSubscription（操作）
- DeleteEventSubscription（操作）
- ModifyEventSubscription（操作）
- DescribeEventSubscriptions（操作）

- AddSourceIdentifierToSubscription（操作）
- RemoveSourceIdentifierFromSubscription（操作）
- DescribeEvents（操作）
- DescribeEventCategories（操作）

结构：

- Event（结构）
- EventCategoriesMap（结构）
- EventSubscription（结构）

CreateEventSubscription（操作）

此 API 的 AWS CLI 名称为：`create-event-subscription`。

创建事件通知订阅。此操作需要一个通过 Neptune 控制台、SNS 控制台或 SNS API 创建的主题 ARN（Amazon 资源名称）。要使用 SNS 获取 ARN，您必须在 Amazon SNS 中创建主题并订阅该主题。ARN 显示在 SNS 控制台中。

您可以指定要获取通知的源类型 (SourceType)，提供触发事件的 Neptune 源 (SourceIds) 列表，以及提供您希望获得通知的事件的事件类别 (EventCategories) 列表。例如，您可以指定 SourceType = db-instance、SourceIds = mydbinstance1, mydbinstance2 并且 EventCategories = Availability, Backup。

如果您同时指定 SourceType 和 SourceIds，例如 SourceType = db-instance 和 SourceIdentifier = myDBInstance1，您将收到指定源的所有 db-instance 事件的通知。如果指定 SourceType，但未指定 SourceIdentifier，则将收到所有 Neptune 源的该源类型事件的通知。如果既未指定 SourceType，也未指定 SourceIdentifier，则将收到通过属于客户账户的所有 Neptune 源生成的事件的通知。

请求

- Enabled（在 CLI 中：`--enabled`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  一个布尔值；设置为 true 可激活订阅，设置为 false 可创建订阅但不激活它。
- EventCategories（在 CLI 中：`--event-categories`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

您要订阅到的 SourceType 的事件类别列表。您可以使用 DescribeEventCategories 操作，查看指定 SourceType 的类别的列表。

- SnsTopicArn（在 CLI 中：`--sns-topic-arn`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为事件通知创建的 SNS 主题的 Amazon 资源名称（ARN）。在您创建主题并订阅到该主题时，由 Amazon SNS 创建 ARN。

- SourceIds（在 CLI 中：`--source-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为其返回事件的事件源的标识符列表。如果未指定，则响应中包含所有源。标识符必须以字母开头，并且只能包含 ASCII 字母、数字和连字符，不能以连字符结尾，也不能包含两个连续的连字符。

  约束：

  - 如果提供 SourceIds，则还必须提供 SourceType。
  - 如果源类型是数据库实例，则必须提供 `DBInstanceIdentifier`。
  - 如果源类型是数据库安全组，则必须提供 `DBSecurityGroupName`。
  - 如果源类型是数据库参数组，则必须提供 `DBParameterGroupName`。
  - 如果源类型是数据库快照，则必须提供 `DBSnapshotIdentifier`。

- SourceType（在 CLI 中：`--source-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  生成事件的源的类型。例如，如果您要接收由数据库实例生成的事件的通知，则应将此参数设置为 db-instance。如果未指定此值，则将返回所有事件。

  有效值：`db-instance` | `db-cluster` | `db-parameter-group` | `db-security-group` | `db-snapshot` | `db-cluster-snapshot`

- SubscriptionName（在 CLI 中：`--subscription-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  订阅的名称。

  约束：名称必须少于 255 个字符。

- Tags：（在 CLI 中：`--tags`）标签 对象的数组。

  要应用到新事件订阅的标签。

响应

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

- CustomerAwsId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。
- CustSubscriptionId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅 ID。
- Enabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示是否启用订阅的布尔值。True 表示订阅已启用。
- EventCategoriesList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。
- EventSubscriptionArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。
- SnsTopicArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。
- SourceIdsList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。
- SourceType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源类型。
- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的状态。

  约束：

  可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

  状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。
- SubscriptionCreationTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

创建事件订阅通知的时间。

错误

- EventSubscriptionQuotaExceededFault
- SubscriptionAlreadyExistFault
- SNSInvalidTopicFault
- SNSNoAuthorizationFault
- SNSTopicArnNotFoundFault
- SubscriptionCategoryNotFoundFault
- SourceNotFoundFault

DeleteEventSubscription（操作）

此 API 的 AWS CLI 名称为：`delete-event-subscription`。

删除事件通知订阅。

请求

- SubscriptionName（在 CLI 中：`--subscription-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  您要删除的事件通知订阅的名称。

响应

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

- CustomerAwsId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。
- CustSubscriptionId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅 ID。
- Enabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

指示是否启用订阅的布尔值。True 表示订阅已启用。

- EventCategoriesList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。

- EventSubscriptionArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。

- SnsTopicArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。

- SourceIdsList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。

- SourceType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源类型。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的状态。

  约束：

  可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

  状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。

- SubscriptionCreationTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  创建事件订阅通知的时间。

错误

- SubscriptionNotFoundFault
- InvalidEventSubscriptionStateFault

ModifyEventSubscription（操作）

此 API 的 AWS CLI 名称为：`modify-event-subscription`。

修改现有的事件通知订阅。请注意，您无法使用此调用修改源标识符；要更改订阅的源标识符，请使用 the section called “AddSourceIdentifierToSubscription” 和 the section called “RemoveSourceIdentifierFromSubscription” 调用。

您可以使用 DescribeEventCategories 操作，查看指定 SourceType 的事件类别的列表。

请求

- Enabled（在 CLI 中：`--enabled`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  布尔值；设置为 true 可激活订阅。
- EventCategories（在 CLI 中：`--event-categories`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  您要订阅到的 SourceType 的事件类别列表。您可以使用 DescribeEventCategories 操作，查看指定 SourceType 的类别的列表。
- SnsTopicArn（在 CLI 中：`--sns-topic-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为事件通知创建的 SNS 主题的 Amazon 资源名称（ARN）。在您创建主题并订阅到该主题时，由 Amazon SNS 创建 ARN。
- SourceType（在 CLI 中：`--source-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  生成事件的源的类型。例如，如果您要接收由数据库实例生成的事件的通知，则应将此参数设置为 db-instance。如果未指定此值，则将返回所有事件。

  有效值：db-instance | db-parameter-group | db-security-group | db-snapshot
- SubscriptionName（在 CLI 中：`--subscription-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的名称。

响应

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

- CustomerAwsId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。

- CustSubscriptionId – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅 ID。

- Enabled - 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指示是否启用订阅的布尔值。True 表示订阅已启用。

- EventCategoriesList – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。

- EventSubscriptionArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。

- SnsTopicArn – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。

- SourceIdsList – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。

- SourceType – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅的源类型。

- Status – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  事件通知订阅的状态。

  约束：

  可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

  状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。

- SubscriptionCreationTime – 一个字符串，类型为：string（UTF-8 编码的字符串）。

  创建事件订阅通知的时间。

错误

- EventSubscriptionQuotaExceededFault
- SubscriptionNotFoundFault
- SNSInvalidTopicFault
- SNSNoAuthorizationFault
- SNSTopicArnNotFoundFault
- SubscriptionCategoryNotFoundFault

DescribeEventSubscriptions（操作）

> 此 API 的 AWS CLI 名称为：`describe-event-subscriptions`。

列出客户账户的所有订阅描述。订阅的描述包括 SubscriptionName、SNSTopicARN、CustomerID、SourceType、SourceID、CreationTime 和 Status。

如果您指定了 SubscriptionName，则列出该订阅的描述。

请求

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeOrderableDBInstanceOptions 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。

- SubscriptionName（在 CLI 中：`--subscription-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  您要描述的事件通知订阅的名称。

响应

- EventSubscriptionsList – EventSubscription 对象的数组。

  EventSubscriptions 数据类型的列表。
- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeOrderableDBInstanceOptions 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

错误

- SubscriptionNotFoundFault

AddSourceIdentifierToSubscription（操作）

此 API 的 AWS CLI 名称为：`add-source-identifier-to-subscription`。

将源标识符添加到现有事件通知订阅。

请求

- SourceIdentifier（在 CLI 中：`--source-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要添加的事件源的标识符。

  约束：
  - 如果源类型是数据库实例，则必须提供 `DBInstanceIdentifier`。
  - 如果源类型是数据库安全组，则必须提供 `DBSecurityGroupName`。
  - 如果源类型是数据库参数组，则必须提供 `DBParameterGroupName`。
  - 如果源类型是数据库快照，则必须提供 `DBSnapshotIdentifier`。

- SubscriptionName（在 CLI 中：`--subscription-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要源标识符添加到的事件通知订阅的名称。

响应

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

- CustomerAwsId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。

- CustSubscriptionId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅 ID。

- Enabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示是否启用订阅的布尔值。True 表示订阅已启用。

- EventCategoriesList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。

- EventSubscriptionArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。

- SnsTopicArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。

- SourceIdsList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。

- SourceType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源类型。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的状态。

约束：

可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。

- SubscriptionCreationTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  创建事件订阅通知的时间。

错误

- SubscriptionNotFoundFault
- SourceNotFoundFault

RemoveSourceIdentifierFromSubscription（操作）

此 API 的 AWS CLI 名称为：`remove-source-identifier-from-subscription`。

从现有事件通知订阅中删除源标识符。

请求

- SourceIdentifier（在 CLI 中：`--source-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要从订阅中删除的源标识符，例如数据库实例的数据库实例标识符或安全组的名称。
- SubscriptionName（在 CLI 中：`--subscription-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  从中删除源标识符的事件通知订阅的名称。

响应

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

- CustomerAwsId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。
- CustSubscriptionId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

事件通知订阅 ID。

- Enabled - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示是否启用订阅的布尔值。True 表示订阅已启用。

- EventCategoriesList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。

- EventSubscriptionArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。

- SnsTopicArn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。

- SourceIdsList – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。

- SourceType – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源类型。

- Status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的状态。

  约束：

  可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

  状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。

- SubscriptionCreationTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  创建事件订阅通知的时间。

错误

- SubscriptionNotFoundFault
- SourceNotFoundFault

DescribeEvents（操作）

此 API 的 AWS CLI 名称为：`describe-events`。

返回过去 14 天与数据库实例、数据库安全组、数据库快照和数据库参数组相关的事件。对于特定的数据库实例、数据库安全组、数据库快照或数据库参数组，特定于它们的事件可以通过提供名称作为参数来获取。默认情况下，返回过去一小时的事件。

请求

- Duration（在 CLI 中：`--duration`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  从中检索事件的分钟数。

  默认值：60
- EndTime（在 CLI 中：`--end-time`）– TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  要检索事件的时间段的结束，以 ISO 8601 格式指定。有关 ISO 8601 的更多信息，请转至 ISO8601 Wikipedia 页面。

  示例：2009-07-08T18:00Z
- EventCategories（在 CLI 中：`--event-categories`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  触发事件通知订阅的通知的事件类别列表。
- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 DescribeEvents 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

默认值：100

约束：最低为 20，最高为 100。

- SourceIdentifier（在 CLI 中：`--source-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  为其返回事件的事件源的标识符。如果未指定，则响应中包含所有源。

  约束：

  - 如果提供 SourceIdentifier，则还必须提供 SourceType。
  - 如果源类型是 `DBInstance`，则必须提供 `DBInstanceIdentifier`。
  - 如果源类型是 `DBSecurityGroup`，则必须提供 `DBSecurityGroupName`。
  - 如果源类型是 `DBParameterGroup`，则必须提供 `DBParameterGroupName`。
  - 如果源类型是 `DBSnapshot`，则必须提供 `DBSnapshotIdentifier`。
  - 不能以连字符结束或包含两个连续连字符。
- SourceType（在 CLI 中：`--source-type`）– SourceType，类型为：`string`（UTF-8 编码的字符串）。

  要从中检索事件的事件源。如果未指定值，则返回所有事件。
- StartTime（在 CLI 中：`--start-time`）– TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  要检索事件的时间段的开始，以 ISO 8601 格式指定。有关 ISO 8601 的更多信息，请转至 ISO8601 Wikipedia 页面。

  示例：2009-07-08T18:00Z

响应

- Events – 事件 对象的数组。

  the section called “事件” 实例的列表。
- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 Events 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

DescribeEventCategories（操作）

此 API 的 AWS CLI 名称为：`describe-event-categories`。

显示所有事件源类型的类别列表；或如果指定，则显示指定源类型的类别列表。

请求

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。
- SourceType（在 CLI 中：`--source-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  生成事件的源的类型。

  有效值：db-instance | db-parameter-group | db-security-group | db-snapshot

响应

- EventCategoriesMapList – EventCategoriesMap 对象的数组。

  EventCategoriesMap 数据类型的列表。

结构：

Event（结构）

此数据类型用作 the section called “DescribeEvents” 操作中的响应元素。

字段

- Date – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  指定事件的日期和时间。
- EventCategories – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定事件的类别。
- Message – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

提供此事件的文本。

- SourceArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件的 Amazon 资源名称 (ARN)。

- SourceIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供事件的源的标识符。

- SourceType – 这是 SourceType，类型为：`string`（UTF-8 编码的字符串）。

  为此事件指定源类型。

EventCategoriesMap（结构）

包含成功调用 the section called “DescribeEventCategories” 操作的结果。

字段

- EventCategories – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定源类型的事件类别

- SourceType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回的类别所属的源类型

EventSubscription（结构）

包含成功调用 the section called “DescribeEventSubscriptions” 操作的结果。

字段

- CustomerAwsId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  与事件通知订阅关联的 Amazon 客户账户。

- CustSubscriptionId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅 ID。

- Enabled - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

指示是否启用订阅的布尔值。True 表示订阅已启用。

- EventCategoriesList – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的事件类别列表。

- EventSubscriptionArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件订阅的 Amazon 资源名称 (ARN)。

- SnsTopicArn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的主题 ARN。

- SourceIdsList – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源 ID 列表。

- SourceType – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的源类型。

- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  事件通知订阅的状态。

  约束：

  可以为以下值之一：creating | modifying | deleting | active | no-permission | topic-not-exist

  状态“no-permission”指示 Neptune 不再有权发布到此 SNS 主题。状态“topic-not-exist”指示主题在创建订阅之后删除。

- SubscriptionCreationTime – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  创建事件订阅通知的时间。

`EventSubscription` 用作下列对象的响应元素：

- CreateEventSubscription
- ModifyEventSubscription
- AddSourceIdentifierToSubscription
- RemoveSourceIdentifierFromSubscription
- DeleteEventSubscription

其他 Neptune API

操作:

- AddTagsToResource（操作）
- ListTagsForResource（操作）
- RemoveTagsFromResource（操作）
- ApplyPendingMaintenanceAction（操作）
- DescribePendingMaintenanceActions（操作）
- DescribeDBEngineVersions（操作）

结构：

- DBEngineVersion（结构）
- EngineDefaults（结构）
- PendingMaintenanceAction（结构）
- ResourcePendingMaintenanceActions（结构）
- UpgradeTarget（结构）
- Tag（结构）

AddTagsToResource（操作）

> 此 API 的 AWS CLI 名称为：`add-tags-to-resource`。

向 Amazon Neptune 资源中添加元数据标签。这些标签还可以与成本分配报告一起使用，以跟踪与 Amazon Neptune 资源相关的成本，或者在 Amazon Neptune 的 IAM 策略中的条件语句内使用。

请求

- ResourceName（在 CLI 中：`--resource-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  将标签添加到的 Amazon Neptune 资源。此值是 Amazon 资源名称 (ARN)。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。
- Tags（在 CLI 中：`--tags`）- 必需：标签 对象的数组。

要分配给 Amazon Neptune 资源的标签。

响应

- 无响应参数。

错误

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterNotFoundFault

ListTagsForResource（操作）

此 API 的 AWS CLI 名称为：`list-tags-for-resource`。

列出 Amazon Neptune 资源上的所有标签。

请求

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持此参数。

- ResourceName（在 CLI 中：`--resource-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  带有要列出的标签的 Amazon Neptune 资源。此值是 Amazon 资源名称 (ARN)。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。

响应

- TagList – 标签 对象的数组。

  ListTagsForResource 操作返回的标签列表。

错误

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterNotFoundFault

RemoveTagsFromResource（操作）

此 API 的 AWS CLI 名称为：`remove-tags-from-resource`。

从 Amazon Neptune 资源中删除元数据标签。

请求

- ResourceName（在 CLI 中：`--resource-name`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  从其中删除标签的 Amazon Neptune 资源。此值是 Amazon 资源名称 (ARN)。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。
- TagKeys（在 CLI 中：`--tag-keys`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的标签的标签键（名称）。

响应

- 无响应参数。

错误

- DBInstanceNotFoundFault
- DBSnapshotNotFoundFault
- DBClusterNotFoundFault

ApplyPendingMaintenanceAction（操作）

此 API 的 AWS CLI 名称为：`apply-pending-maintenance-action`。

将待处理的维护操作应用于资源（例如，应用于数据库实例）。

请求

- ApplyAction（在 CLI 中：`--apply-action`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  应用于此资源的待处理的维护操作。

  有效值：`system-update`、`db-upgrade`
- OptInType（在 CLI 中：`--opt-in-type`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于指定加入请求类型或撤消加入请求的值。不能撤消 `immediate` 类型的加入请求。

  有效值：
  - `immediate` - 立即应用维护操作。
  - `next-maintenance` - 在资源的下一个维护时段内应用维护操作。
  - `undo-opt-in` - 取消任何现有的 `next-maintenance` 加入请求。
- ResourceIdentifier（在 CLI 中：`--resource-identifier`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  待处理的维护操作应用于的资源的 Amazon 资源名称 (ARN)。有关创建 ARN 的信息，请参阅构造 Amazon 资源名称 (ARN)。

响应

描述资源的待处理维护操作。

- PendingMaintenanceActionDetails – PendingMaintenanceAction 对象的数组。

  一个列表，提供有关资源的待处理维护操作的详细信息。
- ResourceIdentifier – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  具有待处理维护操作的资源的 ARN。

错误

- ResourceNotFoundFault

DescribePendingMaintenanceActions（操作）

此 API 的 AWS CLI 名称为：`describe-pending-maintenance-actions`。

返回至少具有一个待处理的维护操作的资源（例如，数据库实例）的列表。

请求

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  一个筛选条件，用于指定一个或多个资源以返回其待处理的维护操作。

  支持的筛选条件：

  - `db-cluster-id` - 接受数据库集群标识符和数据库集群 Amazon 资源名称 (ARN)。结果列表将仅包括由这些 ARN 标识的数据库集群的待处理维护操作。
  - `db-instance-id` - 接受数据库实例标识符和数据库实例 ARN。结果列表将仅包括由这些 ARN 标识的数据库实例的待处理维护操作。
- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 `DescribePendingMaintenanceActions` 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数为 `MaxRecords` 指定的记录数。
- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果存在的记录数超过了指定的`MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索剩余的结果。

  默认值：100

  约束：最低为 20，最高为 100。
- ResourceIdentifier（在 CLI 中：`--resource-identifier`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于返回其待处理维护操作的资源的 ARN。

响应

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

由之前的 DescribePendingMaintenanceActions 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数为 MaxRecords 指定的记录数。

- PendingMaintenanceActions – ResourcePendingMaintenanceActions 对象的数组。

  资源的待处理维护操作的列表。

错误

- ResourceNotFoundFault

DescribeDBEngineVersions（操作）

此 API 的 AWS CLI 名称为：describe-db-engine-versions。

返回可用数据库引擎的列表。

请求

- DBParameterGroupFamily（在 CLI 中：--db-parameter-group-family）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要返回其详细信息的特定数据库参数组系列的名称。

  约束：

  - 如果提供，则必须匹配现有 DBParameterGroupFamily。
- DefaultOnly（在 CLI 中：--default-only）– 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  指示仅返回指定引擎的默认版本还是返回引擎与主要版本的组合。
- Engine（在 CLI 中：--engine）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要返回的数据库引擎。
- EngineVersion（在 CLI 中：--engine-version）– 一个字符串，类型为：string（UTF-8 编码的字符串）。

  要返回的数据库引擎版本。

  示例：5.1.49

- Filters：（在 CLI 中：`--filters`）筛选条件 对象的数组。

  当前不支持。

- ListSupportedCharacterSets（在 CLI 中：`--list-supported-character-sets`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果指定了此参数且请求的引擎支持 `CreateDBInstance` 的 `CharacterSetName` 参数，则响应包括每个引擎版本支持的字符集列表。

- ListSupportedTimezones（在 CLI 中：`--list-supported-timezones`）– BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  如果指定了此参数且请求的引擎支持 `CreateDBInstance` 的 `TimeZone` 参数，则响应包括每个引擎版本支持的时区列表。

- Marker（在 CLI 中：`--marker`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

- MaxRecords（在 CLI 中：`--max-records`）– IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  包括在响应中的最大记录数。如果允许超过 `MaxRecords` 值，则在响应中包含称为标记的分页记号，以便检索以下结果。

  默认值：100

  约束：最低为 20，最高为 100。

响应

- DBEngineVersions – DBEngineVersion 对象的数组。

  `DBEngineVersion` 元素的列表。

- Marker – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。

结构：

DBEngineVersion（结构）

此数据类型用作 the section called “DescribeDBEngineVersions” 操作中的响应元素。

字段

- DBEngineDescription – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎的描述。
- DBEngineVersionDescription – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎版本的描述。
- DBParameterGroupFamily – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要用于数据库引擎的数据库参数组系列的名称。
- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎的名称。
- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎的版本号。
- ExportableLogTypes – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库引擎可用于导出到 CloudWatch Logs 的日志的类型。
- SupportedTimezones – 这是 时区 对象数组。

  用于 `CreateDBInstance` 操作的 `Timezone` 参数且受此引擎支持的时区列表。
- SupportsGlobalDatabases - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示您是否可以将 Aurora 全球数据库与特定数据库引擎版本一起使用。
- SupportsLogExportsToCloudwatchLogs - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示引擎版本是否支持将 ExportableLogTypes 指定的日志类型导出到 CloudWatch Logs。
- SupportsReadReplica - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示数据库引擎版本是否支持只读副本。

- ValidUpgradeTarget – 这是 UpgradeTarget 对象数组。

  此数据库引擎版本可以升级到的引擎版本列表。

EngineDefaults（结构）

包含 the section called “DescribeEngineDefaultParameters” 操作的成功调用的结果。

字段

- DBParameterGroupFamily – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指定引擎默认参数应用到的数据库参数组系列的名称。
- Marker – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由之前的 EngineDefaults 请求提供的可选分页标记。如果指定此参数，则响应仅包含标记之外的记录，最大数量为 `MaxRecords` 指定的值。
- Parameters – 这是 参数 对象数组。

  包含引擎默认参数的列表。

`EngineDefaults` 用作下列对象的响应元素：

- DescribeEngineDefaultParameters
- DescribeEngineDefaultClusterParameters

PendingMaintenanceAction（结构）

提供有关资源的待处理维护操作的信息。

字段

- Action – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  可用于资源的待处理维护操作的类型。
- AutoAppliedAfterDate – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

应用操作时维护时段的日期。维护操作在此日期之后的第一个维护时段期间应用于资源。如果指定了此日期，则忽略任何 `next-maintenance` 加入请求。

- CurrentApplyDate – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  待处理维护操作应用于资源的生效日期。此日期考虑了从 the section called "ApplyPendingMaintenanceAction" API、`AutoAppliedAfterDate` 和 `ForcedApplyDate` 收到的加入请求。如果未收到加入请求且未将任何内容指定为 `AutoAppliedAfterDate` 或 `ForcedApplyDate`，则此值为空。

- Description – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  提供有关维护操作的更多详细信息的描述。

- ForcedApplyDate – 这是 TStamp，类型为：`timestamp`（一个时间点，通常定义为与 1970 年 1 月 1 日午夜的偏移量）。

  自动应用维护操作时的日期。无论资源的维护时段如何，维护操作都将在此日期应用于资源。如果指定了此日期，则忽略任何 `immediate` 加入请求。

- OptInStatus – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  指示已为资源接收的加入请求的类型。

ResourcePendingMaintenanceActions（结构）

描述资源的待处理维护操作。

字段

- PendingMaintenanceActionDetails – 这是 PendingMaintenanceAction 对象数组。

  一个列表，提供有关资源的待处理维护操作的详细信息。

- ResourceIdentifier – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  具有待处理维护操作的资源的 ARN。

`ResourcePendingMaintenanceActions` 用作下列对象的响应元素：

- ApplyPendingMaintenanceAction

UpgradeTarget（结构）

数据库实例可以升级到的数据库引擎的版本。

字段

- AutoUpgrade - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，指示目标版本是否应用于 AutoMinorVersionUpgrade 设置为 true 的任何源数据库实例。
- Description – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例可以升级到的数据库引擎的版本。
- Engine – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  升级目标数据库引擎的名称。
- EngineVersion – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  升级目标数据库引擎的版本号。
- IsMajorVersionUpgrade - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，用于指示数据库引擎是否已升级到主要版本。
- SupportsGlobalDatabases – 这是 BooleanOptional，类型为：`boolean` [布尔值（true 或 false）]。

  一个值，用于指示是否可以将 Neptune 全球数据库与目标引擎版本一起使用。

Tag（结构）

分配给包含键值对的 Amazon Neptune 资源的元数据。

字段

- Key – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  键是标签必需的名称。该字符串值的长度可以在 1 到 128 个 Unicode 字符之间，并且不能带有前缀 `aws:` 或 `rds:`。该字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。
- Value – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

值是标签的可选内容。该字符串值的长度可以在 1 到 256 个 Unicode 字符之间，并且不能带有前缀 `aws:` 或 `rds:`。该字符串只能包含 Unicode 字母、数字、空格、“_”、“.”、“/”、“=”、“+”、“-”的集合（Java 正则表达式：“^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$”）。

常见的 Neptune 数据类型

结构：

- AvailabilityZone（结构）
- DBSecurityGroupMembership（结构）
- DomainMembership（结构）
- DoubleRange（结构）
- Endpoint（结构）
- Filter（结构）
- Range（结构）
- ServerlessV2ScalingConfiguration（结构）
- ServerlessV2ScalingConfigurationInfo（结构）
- Timezone（结构）
- VpcSecurityGroupMembership（结构）

AvailabilityZone（结构）

指定可用区。

字段

- Name – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  可用区的名称。

DBSecurityGroupMembership（结构）

指定所指定数据库安全组中的成员资格。

字段

- DBSecurityGroupName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库安全组的名称。
- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库安全组的状态。

DomainMembership（结构）

与数据库实例关联的 Active Directory 域成员资格记录。

字段

- Domain – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Active Directory 域的标识符。
- FQDN – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Active Directory 域的完全限定域名。
- IAMRoleName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  在对 Directory Service 进行 API 调用时要使用的 IAM 角色的名称。
- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据库实例的 Active Directory 域成员资格的状态，例如已加入、待加入、失败等。

DoubleRange（结构）

双精度值的范围。

字段

- From – 这是双精度，类型为：`double`（双精度 IEEE 754 浮点数）。

  范围中的最小值。
- To – 这是双精度，类型为：`double`（双精度 IEEE 754 浮点数）。

  范围中的最大值。

Endpoint（结构）

指定连接终端节点。

有关表示 Amazon Neptune 数据库集群端点的数据结构，请参阅DBClusterEndpoint。

字段

- Address – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定数据库实例的 DNS 地址。
- HostedZoneId – 这是一个字符串，类型为：string（UTF-8 编码的字符串）。

  指定在您创建托管区域时 Amazon Route 53 分配的 ID。
- Port – 这是一个整数，类型为：integer（带符号的 32 位整数）。

  指定数据库引擎侦听的端口。

Filter（结构）

当前不支持此类型。

字段

- Name – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  当前不支持此参数。
- Values – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  当前不支持此参数。

Range（结构）

整数值的范围。

字段

- From – 这是一个整数，类型为：integer（带符号的 32 位整数）。

  范围中的最小值。

- Step – 这是 IntegerOptional，类型为：`integer`（带符号的 32 位整数）。

  范围的步长值。例如，如果您的范围为 5,000 到 10,000，步长值为 1,000，则有效值从 5,000 开始，每次增加 1,000。尽管 7,500 位于该范围内，但它不是该范围的有效值。有效值为 5,000、6,000、7,000、8,000...
- To – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  范围中的最大值。

ServerlessV2ScalingConfiguration（结构）

包含 Neptune 无服务器数据库集群的扩展配置。

有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

字段

- MaxCapacity – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

  Neptune 无服务器集群中数据库实例的最大 Neptune 容量单元 (NCU) 数。您能够以半步增量指定 NCU 值，如 40、40.5、41，以此类推。
- MinCapacity – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

  Neptune 无服务器集群中数据库实例的最小 Neptune 容量单元 (NCU) 数。您能够以半步增量指定 NCU 值，如 8、8.5、9，以此类推。

ServerlessV2ScalingConfigurationInfo（结构）

显示 Neptune 无服务器数据库集群的扩展配置。

有关更多信息，请参阅《Amazon Neptune 用户指南》中的使用 Amazon Neptune 无服务器。

字段

- MaxCapacity – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

  Neptune 无服务器集群中数据库实例的最大 Neptune 容量单元 (NCU) 数。您能够以半步增量指定 NCU 值，如 40、40.5、41，以此类推。
- MinCapacity – 这是 DoubleOptional，类型为：`double`（双精度 IEEE 754 浮点数）。

Neptune 无服务器集群中数据库实例的最小 Neptune 容量单元 (NCU) 数。您能够以半步增量指定 NCU 值，如 8、8.5、9，以此类推。

Timezone（结构）

与 the section called “DBInstance” 关联的时区。

字段

- TimezoneName – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  时区的名称。

VpcSecurityGroupMembership（结构）

此数据类型用作 VPC 安全组成员资格查询的响应元素。

字段

- Status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  VPC 安全组的状态。
- VpcSecurityGroupId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  VPC 安全组的名称。

个别 API 特定的 Neptune 例外

例外：

- AuthorizationAlreadyExistsFault（结构）
- AuthorizationNotFoundFault（结构）
- AuthorizationQuotaExceededFault（结构）
- CertificateNotFoundFault（结构）
- DBClusterAlreadyExistsFault（结构）
- DBClusterNotFoundFault（结构）
- DBClusterParameterGroupNotFoundFault（结构）

- DBClusterQuotaExceededFault（结构）
- DBClusterRoleAlreadyExistsFault（结构）
- DBClusterRoleNotFoundFault（结构）
- DBClusterRoleQuotaExceededFault（结构）
- DBClusterSnapshotAlreadyExistsFault（结构）
- DBClusterSnapshotNotFoundFault（结构）
- DBInstanceAlreadyExistsFault（结构）
- DBInstanceNotFoundFault（结构）
- DBLogFileNotFoundFault（结构）
- DBParameterGroupAlreadyExistsFault（结构）
- DBParameterGroupNotFoundFault（结构）
- DBParameterGroupQuotaExceededFault（结构）
- DBSecurityGroupAlreadyExistsFault（结构）
- DBSecurityGroupNotFoundFault（结构）
- DBSecurityGroupNotSupportedFault（结构）
- DBSecurityGroupQuotaExceededFault（结构）
- DBSnapshotAlreadyExistsFault（结构）
- DBSnapshotNotFoundFault（结构）
- DBSubnetGroupAlreadyExistsFault（结构）
- DBSubnetGroupDoesNotCoverEnoughAZs（结构）
- DBSubnetGroupNotAllowedFault（结构）
- DBSubnetGroupNotFoundFault（结构）
- DBSubnetGroupQuotaExceededFault（结构）
- DBSubnetQuotaExceededFault（结构）
- DBUpgradeDependencyFailureFault（结构）
- DomainNotFoundFault（结构）
- EventSubscriptionQuotaExceededFault（结构）
- GlobalClusterAlreadyExistsFault（结构）
- GlobalClusterNotFoundFault（结构）
- GlobalClusterQuotaExceededFault（结构）

- InstanceQuotaExceededFault（结构）
- InsufficientDBClusterCapacityFault（结构）
- InsufficientDBInstanceCapacityFault（结构）
- InsufficientStorageClusterCapacityFault（结构）
- InvalidDBClusterEndpointStateFault（结构）
- InvalidDBClusterSnapshotStateFault（结构）
- InvalidDBClusterStateFault（结构）
- InvalidDBInstanceStateFault（结构）
- InvalidDBParameterGroupStateFault（结构）
- InvalidDBSecurityGroupStateFault（结构）
- InvalidDBSnapshotStateFault（结构）
- InvalidDBSubnetGroupFault（结构）
- InvalidDBSubnetGroupStateFault（结构）
- InvalidDBSubnetStateFault（结构）
- InvalidEventSubscriptionStateFault（结构）
- InvalidGlobalClusterStateFault（结构）
- InvalidOptionGroupStateFault（结构）
- InvalidRestoreFault（结构）
- InvalidSubnet（结构）
- InvalidVPCNetworkStateFault（结构）
- KMSKeyNotAccessibleFault（结构）
- OptionGroupNotFoundFault（结构）
- PointInTimeRestoreNotEnabledFault（结构）
- ProvisionedIopsNotAvailableInAZFault（结构）
- ResourceNotFoundFault（结构）
- SNSInvalidTopicFault（结构）
- SNSNoAuthorizationFault（结构）
- SNSTopicArnNotFoundFault（结构）
- SharedSnapshotQuotaExceededFault（结构）
- SnapshotQuotaExceededFault（结构）

- SourceNotFoundFault（结构）
- StorageQuotaExceededFault（结构）
- StorageTypeNotSupportedFault（结构）
- SubnetAlreadyInUse（结构）
- SubscriptionAlreadyExistFault（结构）
- SubscriptionCategoryNotFoundFault（结构）
- SubscriptionNotFoundFault（结构）

AuthorizationAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

已指定的 CIDRIP 或 EC2 安全组已获得指定的数据库安全组的授权。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

AuthorizationNotFoundFault（结构）

返回的 HTTP 状态代码：404。

已指定的 CIDRIP 或 EC2 安全组未获得指定的数据库安全组的授权。

Neptune 也可能无法通过 IAM 获得代表您执行必要操作的授权。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

AuthorizationQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

已达到数据库安全组授权配额。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

CertificateNotFoundFault（结构）

返回的 HTTP 状态代码：404。

CertificateIdentifier 不引用现有证书。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

用户已拥有具有给定标识符的数据库集群。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBClusterIdentifier 不引用现有数据库集群。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterParameterGroupNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBClusterParameterGroupName 不引用现有数据库集群参数组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterQuotaExceededFault（结构）

返回的 HTTP 状态代码：403。

用户尝试创建新的数据库集群，并且用户已达到允许的最大数据库集群配额。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterRoleAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

指定的 IAM 角色 Amazon 资源名称 (ARN) 已与指定的数据库集群关联。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterRoleNotFoundFault（结构）

返回的 HTTP 状态代码：404。

指定的 IAM 角色 Amazon 资源名称 (ARN) 未与指定的数据库集群关联。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterRoleQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

您已超过可与指定数据库集群关联的最大 IAM 角色数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterSnapshotAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

用户已拥有具有给定标识符的数据库集群快照。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBClusterSnapshotNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBClusterSnapshotIdentifier 不引用现有数据库集群快照。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBInstanceAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

用户已拥有具有给定标识符的数据库实例。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBInstanceNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBInstanceIdentifier 不引用现有数据库实例。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBLogFileNotFoundFault（结构）

返回的 HTTP 状态代码：404。

LogFileName 不引用现有数据库日志文件。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBParameterGroupAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

存在同名的数据库参数组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBParameterGroupNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBParameterGroupName 不引用现有数据库参数组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBParameterGroupQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过允许的数据库参数组数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSecurityGroupAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

已存在名为 DBSecurityGroupName 的数据库安全组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSecurityGroupNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBSecurityGroupName 不引用现有数据库安全组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSecurityGroupNotSupportedFault（结构）

返回的 HTTP 状态代码：400。

此操作不允许使用数据库安全组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSecurityGroupQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过允许的数据库安全组数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSnapshotAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

DBSnapshotIdentifier 已被现有快照使用。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSnapshotNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBSnapshotIdentifier 不引用现有数据库快照。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetGroupAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

DBSubnetGroupName 已由现有数据库子网组使用。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetGroupDoesNotCoverEnoughAZs（结构）

返回的 HTTP 状态代码：400。

除非只有一个可用区，否则数据库子网组中的子网应至少包含两个可用区。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetGroupNotAllowedFault（结构）

返回的 HTTP 状态代码：400。

指示在创建与源实例位于同一区域中的只读副本时，不应指定 DBSubnetGroup。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetGroupNotFoundFault（结构）

返回的 HTTP 状态代码：404。

DBSubnetGroupName 不引用现有数据库子网组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetGroupQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过允许的数据库子网组数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBSubnetQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过数据库子网组中允许的子网数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DBUpgradeDependencyFailureFault（结构）

返回的 HTTP 状态代码：400。

数据库升级失败，因为无法修改数据库所依赖的资源。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

DomainNotFoundFault（结构）

返回的 HTTP 状态代码：404。

域 不引用现有的 Active Directory 域。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

EventSubscriptionQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

您已超过可订阅的事件数量。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

GlobalClusterAlreadyExistsFault（结构）

返回的 HTTP 状态代码：400。

GlobalClusterIdentifier 已存在。选择新的全球数据库标识符（唯一名称）来创建新的全球数据库集群。

字段

- message – 这是 ExceptionMessage，类型为：string（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

GlobalClusterNotFoundFault（结构）

返回的 HTTP 状态代码：404。

GlobalClusterIdentifier 不指现有的全球数据库集群。

字段

- message – 这是 ExceptionMessage，类型为：string（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

GlobalClusterQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

该账户的全球数据库集群数量已达到允许的最大值。

字段

- message – 这是 ExceptionMessage，类型为：string（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InstanceQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过允许的数据库实例数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InsufficientDBClusterCapacityFault（结构）

返回的 HTTP 状态代码：403。

数据库集群没有足够的容量用于当前操作。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InsufficientDBInstanceCapacityFault（结构）

返回的 HTTP 状态代码：400。

指定的数据库实例类在指定的可用区中不可用。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InsufficientStorageClusterCapacityFault（结构）

返回的 HTTP 状态代码：400。

当前操作没有足够的可用存储空间。通过更新子网组来使用具有更多可用存储空间的不同可用区，可以解决此错误。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBClusterEndpointStateFault（结构）

返回的 HTTP 状态代码：400。

当端点处于这种状态时，无法在端点上执行所请求的操作。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBClusterSnapshotStateFault（结构）

返回的 HTTP 状态代码：400。

提供的值不是有效的数据库集群快照状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBClusterStateFault（结构）

返回的 HTTP 状态代码：400。

数据库集群未处于有效状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBInstanceStateFault（结构）

返回的 HTTP 状态代码：400。

指定的数据库实例未处于可用 状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBParameterGroupStateFault（结构）

返回的 HTTP 状态代码：400。

数据库参数组正在使用或处于无效状态。如果您尝试删除参数组，则在参数组处于此状态时无法将其删除。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBSecurityGroupStateFault（结构）

返回的 HTTP 状态代码：400。

数据库安全组的状态不允许执行删除。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBSnapshotStateFault（结构）

返回的 HTTP 状态代码：400。

数据库快照的状态不允许执行删除。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBSubnetGroupFault（结构）

返回的 HTTP 状态代码：400。

指示 DBSubnetGroup 与同一源实例的现有跨区域只读副本的 VPC 不属于同一 VPC。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBSubnetGroupStateFault（结构）

返回的 HTTP 状态代码：400。

无法删除数据库子网组，因为它正在使用中。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidDBSubnetStateFault（结构）

返回的 HTTP 状态代码：400。

数据库子网未处于可用 状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

描述问题详细信息的消息。

InvalidEventSubscriptionStateFault（结构）

返回的 HTTP 状态代码：400。

事件订阅处于无效状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidGlobalClusterStateFault（结构）

返回的 HTTP 状态代码：400。

全球集群处于无效状态，无法执行请求的操作。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidOptionGroupStateFault（结构）

返回的 HTTP 状态代码：400。

选项组未处于可用 状态。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidRestoreFault（结构）

返回的 HTTP 状态代码：400。

无法从 vpc 备份还原到非 vpc 数据库实例。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidSubnet（结构）

返回的 HTTP 状态代码：400。

请求的子网无效，或者请求的多个子网并非全部位于同一个常见 VPC 中。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

InvalidVPCNetworkStateFault（结构）

返回的 HTTP 状态代码：400。

由于用户的更改，数据库子网组在创建后并不会覆盖所有可用区。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

KMSKeyNotAccessibleFault（结构）

返回的 HTTP 状态代码：400。

访问 KMS 密钥时出错。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

OptionGroupNotFoundFault（结构）

返回的 HTTP 状态代码：404。

无法找到指定的选项组。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

PointInTimeRestoreNotEnabledFault（结构）

返回的 HTTP 状态代码：400。

SourceDBInstanceIdentifier 引用 BackupRetentionPeriod 等于 0 的数据库实例。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

ProvisionedIopsNotAvailableInAZFault（结构）

返回的 HTTP 状态代码：400。

预配置 IOPS 在指定的可用区中不可用。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

ResourceNotFoundFault（结构）

返回的 HTTP 状态代码：404。

找不到指定的资源 ID。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SNSInvalidTopicFault（结构）

返回的 HTTP 状态代码：400。

SNS 主题无效。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SNSNoAuthorizationFault（结构）

返回的 HTTP 状态代码：400。

没有 SNS 授权。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SNSTopicArnNotFoundFault（结构）

返回的 HTTP 状态代码：404。

找不到 SNS 主题的 ARN。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SharedSnapshotQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

您已超过您可与其共享手动数据库快照的最大账户数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SnapshotQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过允许的数据库快照数。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SourceNotFoundFault（结构）

返回的 HTTP 状态代码：404。

找不到源。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

StorageQuotaExceededFault（结构）

返回的 HTTP 状态代码：400。

请求将导致用户超过所有数据库实例中可用的允许存储量。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

StorageTypeNotSupportedFault（结构）

返回的 HTTP 状态代码：400。

指定的 StorageType 无法与数据库实例关联。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SubnetAlreadyInUse（结构）

返回的 HTTP 状态代码：400。

数据库子网已在可用区中使用。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SubscriptionAlreadyExistFault（结构）

返回的 HTTP 状态代码：400。

此订阅已存在。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SubscriptionCategoryNotFoundFault（结构）

返回的 HTTP 状态代码：404。

找不到指定的订阅类别。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

SubscriptionNotFoundFault（结构）

返回的 HTTP 状态代码：404。

找不到指定的订阅。

字段

- message – 这是 ExceptionMessage，类型为：`string`（UTF-8 编码的字符串）。

  描述问题详细信息的消息。

Amazon Neptune 数据 API 参考

本章记录了 Neptune 数据 API 操作，您可以使用这些操作来加载、访问和维护 Neptune 图形中的数据。

Neptune 数据 API 为加载数据、运行查询、获取有关数据的信息以及运行机器学习操作提供 SDK 支持。它支持 Neptune 中的 Gremlin 和 openCypher 查询语言，并可用于所有 SDK 语言中。它自动签署 API 请求，极大地简化了将 Neptune 集成到应用程序中的过程。

目录

- ClientTimeoutException（结构）
- ConcurrentModificationException（结构）
- ConstraintViolationException（结构）
- ExpiredStreamException（结构）
- FailureByQueryException（结构）
- IllegalArgumentException（结构）
- InternalFailureException（结构）
- InvalidArgumentException（结构）
- InvalidNumericDataException（结构）
- InvalidParameterException（结构）
- LoadUrlAccessDeniedException（结构）
- MalformedQueryException（结构）
- MemoryLimitExceededException（结构）
- MethodNotAllowedException（结构）
- MissingParameterException（结构）
- MLResourceNotFoundException（结构）
- ParsingException（结构）
- PreconditionsFailedException（结构）
- QueryLimitExceededException（结构）
- QueryLimitException（结构）
- QueryTooLargeException（结构）
- ReadOnlyViolationException（结构）
- S3Exception（结构）
- ServerShutdownException（结构）
- StatisticsNotAvailableException（结构）
- StreamRecordsNotFoundException（结构）
- ThrottlingException（结构）
- TimeLimitExceededException（结构）
- TooManyRequestsException（结构）
- UnsupportedOperationException（结构）

- UnloadUrlAccessDeniedException（结构）

Neptune 数据平面引擎、快速重置和通用结构 API

引擎操作：

- GetEngineStatus（操作）
- ExecuteFastReset（操作）

引擎操作结构：

- QueryLanguageVersion（结构）
- FastResetToken（结构）

GetEngineStatus（操作）

此 API 的 AWS CLI 名称为：`get-engine-status`。

检索主机上图形数据库的状态。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetEngineStatus IAM 操作的策略。

请求

- 无请求参数。

响应

- dbEngineVersion – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  设置为在数据库集群上运行的 Neptune 引擎版本。如果此引擎版本发布以来已手动安装补丁程序，则版本号添加前缀 `Patch-`。
- dfeQueryEngine – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果 DFE 引擎已完全启用，则设置为 `enabled`；或者如果 DFE 引擎仅用于 `useDFE` 查询提示设置为 `true` 的查询，则设置为 `viaQueryHint`（默认值）。

- features – 它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  包含有关在您的数据库集群上启用的特征的状态信息。
- gremlin – 一个 QueryLanguageVersion 对象。

  包含有关您的集群上可用的 Gremlin 查询语言的信息。具体而言，它包含一个版本字段，用于指定引擎正在使用的当前 TinkerPop 版本。
- labMode – 它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含引擎正在使用的实验室模式设置。
- opencypher – 一个 QueryLanguageVersion 对象。

  包含有关您的集群上可用的 openCypher 查询语言的信息。具体而言，它包含一个版本字段，用于指定引擎正在使用的当前 OperCypher 版本。
- role – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果实例为只读副本，则设置为 `reader`；如果实例为主实例，则设置为 `writer`。
- rollingBackTrxCount – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  如果有事务正在回滚，则此字段将设置为此类事务的数量。如果没有，则该字段根本不会出现。
- rollingBackTrxEarliestStartTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  设置为正在回滚的最早事务的开始时间。如果没有回滚任何事务，则该字段根本不会出现。
- settings – 它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含有关数据库集群上的当前设置的信息。例如，包含当前集群查询超时设置（`clusterQueryTimeoutInMs`）。

- sparql – 一个 QueryLanguageVersion 对象。

  包含有关您的集群上可用的 SPARQL 查询语言的信息。具体而言，它包含一个版本字段，用于指定引擎正在使用的当前 SPARQL 版本。

- startTime – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  设置为当前服务器进程启动的 UTC 时间。

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果实例没有遇到问题，则设置为 `healthy`。如果实例从崩溃中恢复或者进行了重启，并且最近一次服务器关闭时仍有活动的事务在运行，则状态设置为 `recovery`。

错误

- UnsupportedOperationException
- InternalFailureException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- IllegalArgumentException
- TooManyRequestsException

ExecuteFastReset（操作）

此 API 的 AWS CLI 名称为：`execute-fast-reset`。

快速重置 REST API 允许您快速轻松地重置 Neptune 图形，同时移除其所有数据。

Neptune 快速重置是一个两步过程。首先，在 `action` 设置为 `initiateDatabaseReset` 的情况下调用 `ExecuteFastReset`。这将返回一个 UUID 令牌，然后在 `action` 设置为 `performDatabaseReset` 的情况下再次调用 `ExecuteFastReset` 时包含该令牌。请参阅使用快速重置 API 清空 Amazon Neptune 数据库集群。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ResetDatabase IAM 操作的策略。

请求

- action（在 CLI 中：`--action`）– 必需：一个操作，类型为：`string`（UTF-8 编码的字符串）。

  快速重置操作。下列值之一：

  - **`initiateDatabaseReset`** – 此操作会生成实际执行快速重置所需的唯一令牌。
  - **`performDatabaseReset`** – 此操作使用 `initiateDatabaseReset` 操作生成的令牌来实际执行快速重置。

- token（在 CLI 中：`--token`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于启动重置的快速重置令牌。

响应

- payload – 一个 FastResetToken 对象。

  `payload` 仅由 `initiateDatabaseReset` 操作返回，并包含用于 `performDatabaseReset` 操作以实现重置的唯一令牌。
- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  `status` 仅针对 `performDatabaseReset` 操作返回，表示快速重置请求是否被接受。

错误

- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- ServerShutdownException
- PreconditionsFailedException
- MethodNotAllowedException
- ReadOnlyViolationException
- ConstraintViolationException

- InvalidArgumentException
- MissingParameterException

引擎操作结构：

QueryLanguageVersion（结构）

用于表达查询语言版本的结构。

字段

- version – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  查询语言的版本。

FastResetToken（结构）

包含用于启动快速重置的快速重置令牌的结构。

字段

- token – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由数据库在 `initiateDatabaseReset` 操作中生成的 UUID，然后 `performDatabaseReset` 使用它来重置数据库。

Neptune 查询 API

Gremlin 查询操作：

- ExecuteGremlinQuery（操作）
- ExecuteGremlinExplainQuery（操作）
- ExecuteGremlinProfileQuery（操作）
- ListGremlinQueries（操作）
- GetGremlinQueryStatus（操作）
- CancelGremlinQuery（操作）

openCypher 查询操作：

- ExecuteOpenCypherQuery（操作）
- ExecuteOpenCypherExplainQuery（操作）
- ListOpenCypherQueries（操作）
- GetOpenCypherQueryStatus（操作）
- CancelOpenCypherQuery（操作）

查询结构：

- QueryEvalStats（结构）
- GremlinQueryStatus（结构）
- GremlinQueryStatusAttributes（结构）

ExecuteGremlinQuery（操作）

此 API 的 AWS CLI 名称为：`execute-gremlin-query`。

此命令执行一个 Gremlin 查询。Amazon Neptune 与 Apache TinkerPop3 和 Gremlin 兼容，因此，您可以使用 Gremlin 遍历语言来查询图形，正如 Apache TinkerPop3 文档的图形中所述。更多详细信息也可以在使用 Gremlin 访问 Neptune 图形中找到。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加一个策略，该策略允许在该集群中执行以下 IAM 操作之一，具体取决于查询：

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:Gremlin IAM 条件键来限制 Gremlin 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- gremlinQuery（在 CLI 中：`--gremlin-query`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

使用此 API，您可以像使用 HTTP 端点一样以字符串格式运行 Gremlin 查询。该接口与您的数据库集群使用的任何 Gremlin 版本兼容（请参阅 Tinkerpop 客户端部分以确定您的引擎版本支持哪些 Gremlin 版本）。

- serializer（在 CLI 中：`--serializer`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果不为 null，则在以该参数指定的格式序列化的响应消息中返回查询结果。有关当前支持的格式列表，请参阅 TinkerPop 文档中的 GraphSON 部分。

响应

- meta – 文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  有关 Gremlin 查询的元数据。

- requestId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin 查询的唯一标识符。

- result – 文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  来自服务器的 Gremlin 查询输出。

- status – 一个 GremlinQueryStatusAttributes 对象。

  Gremlin 查询的状态。

错误

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException

- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ExecuteGremlinExplainQuery（操作）

此 API 的 AWS CLI 名称为：`execute-gremlin-explain-query`。

执行 Gremlin Explain 查询。

Amazon Neptune 添加了一项名为 `explain` 的 Gremlin 特征，该特征提供了一种自助服务工具，用于了解 Neptune 引擎为查询所采用的执行方法。您可以通过将 `explain` 参数添加到提交 Gremlin 查询的 HTTP 调用来调用它。

Explain 特征提供了有关查询执行计划的逻辑结构的信息。您可以使用这些信息来识别潜在的评估和执行瓶颈，并调整查询，如调试 Gremlin 查询中所述。您还可以使用查询提示来改进查询执行计划。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加一个策略，该策略允许在该集群中执行以下 IAM 操作之一，具体取决于查询：

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:Gremlin IAM 条件键来限制 Gremlin 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- gremlinQuery（在 CLI 中：`--gremlin-query`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin explain 查询字符串。

响应

- output – 一个 ReportAsText，类型为：`blob`（未解释的二进制数据块）。

  包含 Gremlin explain 结果的文本 blob，如调整 Gremlin 查询中所述。

错误

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException

- MissingParameterException
- ConcurrentModificationException

ExecuteGremlinProfileQuery（操作）

此 API 的 AWS CLI 名称为：`execute-gremlin-profile-query`。

执行 Gremlin Profile 查询，该查询运行指定的遍历，收集有关运行的各种指标，并生成分析报告作为输出。有关详细信息，请参阅 Neptune 中的 Gremlin profile API。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ReadDataViaQuery IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:Gremlin IAM 条件键来限制 Gremlin 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- chop（在 CLI 中：`--chop`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  如果不为零，结果字符串将在该字符数处截断。如果设置为零，字符串将包含所有结果。
- gremlinQuery（在 CLI 中：`--gremlin-query`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要分析的 Gremlin 查询字符串。
- indexOps（在 CLI 中：`--index-ops`）– 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果此标志设置为 TRUE，则结果包含在查询执行和序列化期间发生的所有索引操作的详细报告。
- results（在 CLI 中：`--results`）– 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  如果此标志设置为 TRUE，则收集查询结果并在分析报告中显示。如果为 FALSE，则仅显示结果计数。
- serializer（在 CLI 中：`--serializer`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果不为空，则在以该参数指定的格式序列化的响应消息中返回收集的结果。有关更多信息，请参阅 Neptune 中的 Gremlin profile API。

响应

- output – 一个 ReportAsText，类型为：`blob`（未解释的二进制数据块）。

  包含 Gremlin Profile 结果的文本 blob。有关详细信息，请参阅 Neptune 中的 Gremlin profile API。

错误

- QueryTooLargeException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ListGremlinQueries（操作）

此 API 的 AWS CLI 名称为：`list-gremlin-queries`。

列出活动的 Gremlin 查询。有关输出的详细信息，请参阅 Gremlin 查询状态 API。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetQueryStatus IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:Gremlin IAM 条件键来限制 Gremlin 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- includeWaiting（在 CLI 中：`--include-waiting`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 `TRUE`，则返回的列表包括等待的查询。默认值为 `FALSE`；

响应

- acceptedQueryCount – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  已接受但尚未完成的查询数，包括队列中的查询。
- queries – GremlinQueryStatus 对象的数组。

  当前查询的列表。
- runningQueryCount – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  当前正在运行的 Gremlin 查询的数量。

错误

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException

- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

GetGremlinQueryStatus（操作）

此 API 的 AWS CLI 名称为：`get-gremlin-query-status`。

获取指定的 Gremlin 查询的状态。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetQueryStatus IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:Gremlin IAM 条件键来限制 Gremlin 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- queryId（在 CLI 中：`--query-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  标识 Gremlin 查询的唯一标识符。

响应

- queryEvalStats – 一个 QueryEvalStats 对象。

  Gremlin 查询的评估状态。
- queryId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  正在返回其状态的查询的 ID。
- queryString – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

Gremlin 查询字符串。

错误

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

CancelGremlinQuery（操作）

此 API 的 AWS CLI 名称为：`cancel-gremlin-query`。

取消 Gremlin 查询。有关更多信息，请参阅 Gremlin 查询取消。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelQuery IAM 操作的策略。

请求

- queryId（在 CLI 中：`--query-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

标识要取消的查询的唯一标识符。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消的状态

错误

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

openCypher 查询操作：

ExecuteOpenCypherQuery（操作）

此 API 的 AWS CLI 名称为：`execute-open-cypher-query`。

执行 openCypher 查询。有关更多信息，请参阅使用 openCypher 访问 Neptune 图形。

Neptune 支持使用 openCypher 构建图形应用程序，openCypher 是目前使用图形数据库的开发人员最常用的查询语言之一。开发人员、业务分析师和数据科学家都喜欢 openCypher 的声明式、受 SQL 启发的语法，因为它为查询属性图提供了一种熟悉的结构。

openCypher 语言最初由 Neo4j 开发，然后于 2015 年开源，并在 Apache 2 开源许可证下为 openCypher 项目做出了贡献。

请注意，在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加一个策略，该策略允许在该集群中执行以下 IAM 操作之一，具体取决于查询：

- neptune-db:ReadDataViaQuery
- neptune-db:WriteDataViaQuery
- neptune-db:DeleteDataViaQuery

另请注意，可以在策略文档中使用 neptune-db:QueryLanguage:OpenCypher IAM 条件键来限制 openCypher 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- openCypherQuery（在 CLI 中：`--open-cypher-query`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要执行的 openCypher 查询字符串。
- parameters（在 CLI 中：`--parameters`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于查询执行的 openCypher 查询参数。有关更多信息，请参阅 openCypher 参数化查询示例。

响应

- results – 必需：文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  openCypher 查询结果。

错误

- QueryTooLargeException

- InvalidNumericDataException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ExecuteOpenCypherExplainQuery（操作）

> 此 API 的 AWS CLI 名称为：`execute-open-cypher-explain-query`。

执行 openCypher `explain` 请求。有关更多信息，请参阅 openCypher explain 特征。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ReadDataViaQuery IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:OpenCypher IAM 条件键来限制 openCypher 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- explainMode（在 CLI 中：`--explain-mode`）– 必需：OpenCypherExplainMode，类型为：`string`（UTF-8 编码的字符串）。

  openCypher `explain` 模式。可以为以下值之一：`static`、`dynamic` 或 `details`。
- openCypherQuery（在 CLI 中：`--open-cypher-query`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  openCypher 查询字符串。
- parameters（在 CLI 中：`--parameters`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  openCypher 查询参数。

响应

- results – 必需：Blob，类型为：`blob`（未解释的二进制数据块）。

  一个包含 openCypher `explain` 结果的文本 blob。

错误

- QueryTooLargeException
- InvalidNumericDataException
- BadRequestException
- QueryLimitExceededException
- InvalidParameterException
- QueryLimitException
- ClientTimeoutException
- CancelledByUserException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException

- MemoryLimitExceededException
- PreconditionsFailedException
- MalformedQueryException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

ListOpenCypherQueries（操作）

此 API 的 AWS CLI 名称为：`list-open-cypher-queries`。

列出活动的 openCypher 查询。有关更多信息，请参阅 Neptune openCypher 状态端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetQueryStatus IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:OpenCypher IAM 条件键来限制 openCypher 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- includeWaiting（在 CLI 中：`--include-waiting`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 TRUE 且其它参数不存在，则会导致为等待的查询和正在运行的查询返回状态信息。

响应

- acceptedQueryCount – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  已接受但尚未完成的查询数，包括队列中的查询。
- queries – GremlinQueryStatus 对象的数组。

  当前 openCypher 查询的列表。

- runningQueryCount – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  当前正在运行的 openCypher 查询的数量。

错误

- InvalidNumericDataException
- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

GetOpenCypherQueryStatus（操作）

此 API 的 AWS CLI 名称为：`get-open-cypher-query-status`。

检索指定的 openCypher 查询的状态。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetQueryStatus IAM 操作的策略。

请注意，可以在策略文档中使用 neptune-db:QueryLanguage:OpenCypher IAM 条件键来限制 openCypher 查询的使用（请参阅 Neptune IAM 数据访问策略声明中提供的条件键）。

请求

- queryId（在 CLI 中：`--query-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要检索其查询状态的 openCypher 查询的唯一 ID。

响应

- queryEvalStats – 一个 QueryEvalStats 对象。

  openCypher 查询评估状态。
- queryId – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  正在返回其状态的查询的唯一 ID。
- queryString – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  openCypher 查询字符串。

错误

- InvalidNumericDataException
- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ReadOnlyViolationException

- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

CancelOpenCypherQuery（操作）

此 API 的 AWS CLI 名称为：`cancel-open-cypher-query`。

取消指定的 openCypher 查询。有关更多信息，请参阅 Neptune openCypher 状态端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelQuery IAM 操作的策略。

请求

- queryId（在 CLI 中：`--query-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要取消的 openCypher 查询的唯一ID。
- silent（在 CLI 中：`--silent`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 TRUE，则会导致以静默方式取消 openCypher 查询。

响应

- payload - 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  openCypher 查询的取消有效负载。
- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  openCypher 查询的取消状态。

错误

- InvalidNumericDataException

- BadRequestException
- InvalidParameterException
- ClientTimeoutException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- FailureByQueryException
- PreconditionsFailedException
- ParsingException
- ConstraintViolationException
- TimeLimitExceededException
- InvalidArgumentException
- MissingParameterException
- ConcurrentModificationException

查询结构：

QueryEvalStats（结构）

用于捕获查询统计数据的结构，例如正在运行、接受或等待的查询数量及其详细信息。

字段

- cancelled - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果查询已取消，则设置为 TRUE，否则设置为 FALSE。
- elapsed – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  到目前为止，查询已运行的毫秒数。
- subqueries – 这是一个文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  此查询中的子查询数。
- waited – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

表示查询等待了多长时间（以毫秒为单位）。

GremlinQueryStatus（结构）

捕获 Gremlin 查询的状态（请参阅 Gremlin 查询状态 API 页面）。

字段

- queryEvalStats –这是一个 QueryEvalStats 对象。

  Gremlin 查询的查询统计信息。
- queryId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin 查询的 ID。
- queryString – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin 查询的查询字符串。

GremlinQueryStatusAttributes（结构）

包含 Gremlin 查询的状态组件。

字段

- attributes – 这是一个文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  Gremlin 查询状态的属性。
- code – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  从 Gremlin 查询请求返回的 HTTP 响应代码。
- message – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  状态消息。

Neptune 数据面板批量加载程序 API

批量加载操作：

- StartLoaderJob（操作）
- GetLoaderJobStatus（操作）
- ListLoaderJobs（操作）
- CancelLoaderJob（操作）

批量加载结构：

- LoaderIdResult（结构）

StartLoaderJob（操作）

此 API 的 AWS CLI 名称为：`start-loader-job`。

启动 Neptune 批量加载程序任务，以将数据从 Amazon S3 桶加载到 Neptune 数据库实例中。请参阅使用 Amazon Neptune 批量加载程序摄取数据。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:StartLoaderJob IAM 操作的策略。

请求

- dependencies（在 CLI 中：`--dependencies`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  这是一个可选参数，可以使排队的加载请求取决于队列中的一个或多个先前任务的成功完成。

  Neptune 可以一次对多达 64 个加载请求进行排队，前提是它们的 `queueRequest` 参数设置为 `"TRUE"`。`dependencies` 参数允许您使此类排队请求的执行取决于成功完成队列中的一个或多个指定的先前请求。

  例如，如果加载 `Job-A` 和 `Job-B` 彼此独立，但加载 `Job-C` 在开始之前要求先完成 `Job-A` 和 `Job-B`，请按以下方式进行操作：

  1. 不分顺序接连提交 `load-job-A` 和 `load-job-B`，并保存它们的加载 ID。
  2. 提交 `load-job-C`，并在其 `dependencies` 字段中填入前两个作业的加载 ID：

  Example

```
"dependencies" : ["(job_A_load_id)", "(job_B_load_id)"]
```

由于 dependencies 参数，批量加载程序在 Job-A 和 Job-B 成功完成前将不会启动 Job-C。如果其中任何一个作业失败，则不会执行作业 C，并将其状态设置为 LOAD_FAILED_BECAUSE_DEPENDENCY_NOT_SATISFIED。

您可以通过这种方式设置多个依赖关系级别，这样，一个作业的失败将导致所有直接或间接依赖于该作业的请求被取消。

- failOnError（在 CLI 中：--fail-on-error）– 一个布尔值，类型为：boolean [布尔值（true 或 false）]。

  failOnError – 用于在出错时切换为完全停止的标志。

  允许的值："TRUE"、"FALSE"。

  默认值："TRUE"。

  如果此参数设置为 "FALSE"，则加载程序尝试加载指定位置中的所有数据并跳过任何出错的条目。

  将此参数设置为 "TRUE" 时，加载程序会在遇到错误时立即停止。截至该点加载的数据仍然存在。
- format（在 CLI 中：--format）– 必需：格式，类型为：string（UTF-8 编码的字符串）。

  数据的格式。有关 Neptune Loader 命令的数据格式的更多信息，请参阅加载数据格式。

  允许的值

  - **csv** 适用于 Gremlin CSV 数据格式。
  - **opencypher** 适用于 openCypher CSV 数据格式。
  - **ntriples** 适用于 N-Triples RDF 数据格式。
  - **nquads** 适用于 N-Quads RDF 数据格式。
  - **rdfxml** 适用于 RDF\XML RDF 数据格式。
  - **turtle** 适用于 Turtle RDF 数据格式。
- iamRoleArn（在 CLI 中：--iam-role-arn）– 必需：一个字符串，类型为：string（UTF-8 编码的字符串）。

  Neptune 数据库实例为访问 S3 桶而代入的 IAM 角色的 Amazon 资源名称 (ARN)。此处提供的 IAM 角色 ARN 应附加到数据库集群（请参阅向 Amazon Neptune 集群添加 IAM 角色）。
- mode（在 CLI 中：--mode）– 一种模式，类型为：string（UTF-8 编码的字符串）。

  加载作业模式。

允许的值：RESUME、NEW、AUTO

默认值：AUTO。

- RESUME – 在 RESUME 模式下，加载程序查找来自此源的先前加载，如果找到一个加载，则恢复该加载任务。如果未找到之前的加载作业，则加载程序将停止。

  加载程序将避免重新加载之前作业中已成功加载的文件。它仅尝试处理失败的文件。如果您从 Neptune 集群中删除了之前加载的数据，则此模式下不加载该数据。如果之前的加载任务成功加载了来自同一源的所有文件，则不会重新加载任何文件，并且加载程序返回成功。
- NEW – 在 NEW 模式下，将创建新的加载请求而不管任何之前的加载。您可以使用此模式在从 Neptune 集群删除之前加载的数据之后从源重新加载数据，或加载同一个源处可用的新数据。
- AUTO – 在 AUTO 模式下，加载程序查找来自此同一个源的先前加载任务，如果找到一个加载任务，则恢复该任务，如同在 RESUME 模式中一样。

  如果加载程序没有找到来自同一源的先前加载作业，则会从源加载所有数据，就像在 NEW 模式中一样。

- parallelism（在 CLI 中：--parallelism）– 并行度，类型为：string（UTF-8 编码的字符串）。

  可以设置可选 parallelism 参数以减少批量加载过程使用的线程数。

  允许的值：

  - LOW – 所使用的线程数是可用 vCPU 数除以 8。
  - MEDIUM – 所使用的线程数是可用 vCPU 数除以 2。
  - HIGH – 所使用的线程数与可用 vCPU 数相同。
  - OVERSUBSCRIBE – 所使用的线程数是可用 vCPU 数乘以 2。如果使用此值，则批量加载程序将占用所有可用资源。

    但是，这并不意味着 OVERSUBSCRIBE 设置会导致 100% 的 CPU 利用率。由于负载操作受到 I/O 限制，因此预期的最高 CPU 利用率在 60% 到 70% 之间。

  默认值：HIGH

  在加载 openCypher 数据时，parallelism 设置有时会导致线程之间出现死锁。发生这种情况时，Neptune 会返回 LOAD_DATA_DEADLOCK 错误。通常，您可以通过将 parallelism 设为较低的设置并重试加载命令来解决此问题。

- parserConfiguration（在 CLI 中：`--parser-configuration`）– 它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  `parserConfiguration` – 具有额外解析程序配置值的可选对象。每个子参数也是可选的：

  - **`namedGraphUri`** – 未指定任何图形时所有 RDF 格式的默认图形（适用于非 quads 格式和无图形的 NQUAD 条目）。

    默认为 `https://aws.amazon.com/neptune/vocab/v01/DefaultNamedGraph`。

  - **`baseUri`** – RDF/XML 和 Turtle 格式的基本 URI。

    默认为 `https://aws.amazon.com/neptune/default`。

  - **`allowEmptyStrings`** – 加载 CSV 数据时，Gremlin 用户需要能够将空字符串值 ("") 作为节点和边缘属性传递。如果 `allowEmptyStrings` 设置为 `false`（默认值），则此类空字符串将视为 null 且不会加载。

    如果 `allowEmptyStrings` 设置为 `true`，则加载程序会将空字符串视为有效的属性值并相应地加载它们。

- queueRequest（在 CLI 中：`--queue-request`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  这是一个可选的标志参数，用于指示是否可以对加载请求进行排队。

  您不必等到一个加载任务完成就可以发出下一个任务，因为 Neptune 可以一次对多达 64 个任务进行排队，前提是它们的 `queueRequest` 参数都设置为 `"TRUE"`。任务的队列顺序将为先进先出 (FIFO)。

  如果省略 `queueRequest` 参数或将其设置为 `"FALSE"`，则如果另一个加载作业已在运行，该加载请求将失败。

  允许的值：`"TRUE"`、`"FALSE"`。

  默认值：`"FALSE"`。

- s3BucketRegion（在 CLI 中：`--s-3-bucket-region`）– 必需：一个 S3BucketRegion，类型为：`string`（UTF-8 编码的字符串）。

S3 桶的 Amazon 区域。这必须与数据库集群的 Amazon 区域相匹配。

- source（在 CLI 中：`--source`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  `source` 参数接受标识单个文件、多个文件、一个文件夹或多个文件夹的 S3 URI。Neptune 将每个数据文件加载到任何指定的文件夹中。

  URI 可以采用以下任意格式。

  - `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)`

  URI 的 `object-key-name` 元素等同于 S3 ListObjects API 调用中的前缀参数。它可以识别指定的 S3 桶中名称以该前缀开头的所有对象。可以是单个文件或文件夹，也可以是多个文件和/或文件夹。

  指定的一个或多个文件夹可以包含多个顶点文件和多个边缘文件。

- updateSingleCardinalityProperties（在 CLI 中：`--update-single-cardinality-properties`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  `updateSingleCardinalityProperties` 是一个可选参数，用于控制批量加载程序如何处理单基数顶点或边缘属性的新值。加载 openCypher 数据时不支持这样做。

  允许的值：`"TRUE"`、`"FALSE"`。

  默认值：`"FALSE"`。

  默认情况下，或者当 `updateSingleCardinalityProperties` 被显式设置为 `"FALSE"` 时，加载程序将新值视为错误，因为它违反了单个基数。

  另一方面，当 `updateSingleCardinalityProperties` 设置为 `"TRUE"` 时，批量加载程序会用新值替换现有值。如果在要加载的源文件中提供了多个边缘或单基数顶点属性值，则批量加载结束时的最终值可以是这些新值中的任何一个值。加载程序仅保证现有值已被其中一个新值替换。

- userProvidedEdgeIds（在 CLI 中：`--user-provided-edge-ids`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  只有在加载包含关系 ID 的 openCypher 数据时，才需要此参数。当加载数据中显式提供 openCypher 关系 ID 时，必须包含此参数并将它设置为 `True`（推荐）。

如果 `userProvidedEdgeIds` 不存在或设置为 `True`，则加载过程中的每个关系文件中都必须存在 `:ID` 列。

如果 `userProvidedEdgeIds` 存在且设置为 `False`，则加载中的关系文件不得包含 `:ID` 列。相反，Neptune 加载程序会自动为每个关系生成一个 ID。

显式提供关系 ID 非常有用，这样加载程序就可以在 CSV 数据中的错误得到修复后恢复加载，而不必重新加载任何已经加载的关系。如果没有显式分配关系 ID，当必须更正任何关系文件时，加载程序将无法恢复失败的加载，而是必须重新加载所有关系。

响应

- payload – 必需：它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含为加载操作提供标识符的 `loadId` 名称/值对。
- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  HTTP 返回代码，指示加载任务的状态。

错误

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- InternalFailureException
- S3Exception
- PreconditionsFailedException

- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

GetLoaderJobStatus（操作）

此 API 的 AWS CLI 名称为：`get-loader-job-status`。

获取有关指定的加载任务的状态信息。Neptune 跟踪最近的 1024 个批量加载任务，并且存储每个任务的最后 10000 个错误详细信息。

有关更多信息，请参阅 Neptune 加载程序获取状态 API。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetLoaderJobStatus IAM 操作的策略。

请求

- details（在 CLI 中：`--details`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  表示是否要包括总体状态之外的详细信息的标志（TRUE 或 FALSE；默认为 FALSE）。
- errors（在 CLI 中：`--errors`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  指示是否包含遇到的错误列表的标志（TRUE 或 FALSE；默认为 FALSE）。

  将对错误的列表进行分页。`page` 和 `errorsPerPage` 参数允许浏览所有错误。
- errorsPerPage（在 CLI 中：`--errors-per-page`）– PositiveInteger，类型为：`integer`（带符号的 32 位整数），至少为 1。

  每页返回的错误数（正整数；默认为 10）。仅当 `errors` 参数设置为 TRUE 时才有效。
- loadId（在 CLI 中：`--load-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要获取其状态的加载任务的加载 ID。
- page（在 CLI 中：`--page`）– PositiveInteger，类型为：`integer`（带符号的 32 位整数），至少为 1。

  错误页码（正整数；默认为 1）。仅当 `errors` 参数设置为 TRUE 时才有效。

响应

- payload – 必需：文档，类型为：document（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  有关加载任务的状态信息，其布局可能如下所示：

  Example

```
{
          "status" : "200 OK",
          "payload" : {
            "feedCount" : [
              {
                "LOAD_FAILED" : (number)
              }
            ],
            "overallStatus" : {
              "fullUri" : "s3://(bucket)/(key)",
              "runNumber" : (number),
              "retryNumber" : (number),
              "status" : "(string)",
              "totalTimeSpent" : (number),
              "startTime" : (number),
              "totalRecords" : (number),
              "totalDuplicates" : (number),
              "parsingErrors" : (number),
              "datatypeMismatchErrors" : (number),
              "insertErrors" : (number),
            },
            "failedFeeds" : [
              {
                "fullUri" : "s3://(bucket)/(key)",
                "runNumber" : (number),
                "retryNumber" : (number),
                "status" : "(string)",
                "totalTimeSpent" : (number),
                "startTime" : (number),
                "totalRecords" : (number),
                "totalDuplicates" : (number),
                "parsingErrors" : (number),
                "datatypeMismatchErrors" : (number),
                "insertErrors" : (number),
```

```
            }
          ],
          "errors" : {
            "startIndex" : (number),
            "endIndex" : (number),
            "loadId" : "(string),
            "errorLogs" : [ ]
          }
        }
      }
```

- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  请求的 HTTP 响应代码。

错误

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- InternalFailureException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

ListLoaderJobs（操作）

此 API 的 AWS CLI 名称为：`list-loader-jobs`。

检索所有活动加载程序任务的 `loadIds` 列表。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ListLoaderJobs IAM 操作的策略。

请求

- includeQueuedLoads（在 CLI 中：`--include-queued-loads`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  一个可选参数，可用于在请求加载 ID 列表时排除排队加载请求的加载 ID，方法是将该参数设置为 `FALSE`。默认值为 `TRUE`。
- limit（在 CLI 中：`--limit`）– ListLoaderJobsInputLimitInteger，类型为：`integer`（带符号的 32 位整数），不小于 1 或大于 100。

  要列出的加载 ID 的数量。必须是大于零且不大于 `100`（这是默认值）的正整数。

响应

- payload – 必填：一个 LoaderIdResult 对象。

  请求的任务 ID 列表。
- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回任务列表请求的状态。

错误

- UnsupportedOperationException
- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- InternalFailureException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- LoadUrlAccessDeniedException

- IllegalArgumentException
- TooManyRequestsException

CancelLoaderJob（操作）

此 API 的 AWS CLI 名称为：`cancel-loader-job`。

取消指定的加载任务。这是一个 HTTP `DELETE` 请求。有关更多信息，请参阅 Neptune 加载程序获取状态 API。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelLoaderJob IAM 操作的策略。

请求

- loadId（在 CLI 中：`--load-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要删除的加载任务的 ID。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消状态。

错误

- BadRequestException
- InvalidParameterException
- BulkLoadIdNotFoundException
- ClientTimeoutException
- LoadUrlAccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException

- InternalFailureException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

批量加载结构：

LoaderIdResult（结构）

包含加载 ID 的列表。

字段

- loadIds – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  加载 ID 的列表。

Neptune 流数据面板 API

流访问操作：

- GetPropertygraphStream（操作）

流数据结构：

- PropertygraphRecord（结构）
- PropertygraphData（结构）

GetPropertygraphStream（操作）

此 API 的 AWS CLI 名称为：`get-propertygraph-stream`。

获取属性图的流。

借助 Neptune Streams 特征，您可以生成完整的更改日志条目序列，以即时记录对图形数据所做的每一个更改。`GetPropertygraphStream` 使您可以为属性图收集这些更改日志条目。

需要在您的 Neptune 数据库集群上启用 Neptune 流特征。要启用流，请将 neptune_streams 数据库集群参数设置为 1。

请参阅使用 Neptune Streams 实时捕获图形更改。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetStreamRecords IAM 操作的策略。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加一个策略，该策略允许执行以下 IAM 操作之一，具体取决于查询：

请注意，您可以使用以下 IAM 上下文键限制属性图查询：

- neptune-db:QueryLanguage:Gremlin
- neptune-db:QueryLanguage:OpenCypher

请参阅 Neptune IAM 数据访问策略语句中提供的条件键。

请求

- commitNum（在 CLI 中：`--commit-num`）– 长整型，类型为：`long`（带符号的 64 位整数）。

  从更改日志流中读取的起始记录的提交编号。当 `iteratorType` 为 `AT_SEQUENCE_NUMBER` 或 `AFTER_SEQUENCE_NUMBER` 时，此参数为必需的，当 `iteratorType` 为 `TRIM_HORIZON` 或 `LATEST` 时，忽略此参数。
- encoding（在 CLI 中：`--encoding`）– 一种编码，类型为：`string`（UTF-8 编码的字符串）。

  如果设置为 TRUE，Neptune 将使用 gzip 编码压缩响应。
- iteratorType（在 CLI 中：`--iterator-type`）– IteratorType，类型为：`string`（UTF-8 编码的字符串）。

  可以是以下值之一：
  - `AT_SEQUENCE_NUMBER` – 指示读取应该从由 `commitNum` 和 opNum 参数共同指定的事件序列号开始。
  - `AFTER_SEQUENCE_NUMBER` – 指示读取应该从紧接在由 `commitNum` 和 opNum 参数共同指定的事件序列号之后的事件序列号开始。
  - `TRIM_HORIZON` – 指示读取应该从系统中最后一条未剪裁的记录开始，该记录是更改日志流中时间最久的未过期（尚未删除）记录。

  - LATEST – 指示读取应该从系统中最近的记录开始，该记录是更改日志流中最新的未过期（尚未删除）记录。
- limit（在 CLI 中：`--limit`）– GetPropertygraphStreamInputLimitLong，类型为：`long`（有符号的 64 位整数），不小于 1 或大于 100000。

  指定要返回的最大记录数。响应还受 10 MB 大小的限制，该大小限制不能修改，并且优先级高于 `limit` 参数中指定的记录数。如果达到了 10 MB 限制，响应中将包含一条超出阈值记录。

  `limit` 的范围为 1 到 100000，默认值为 10。
- opNum（在 CLI 中：`--op-num`）– 长整型，类型为：`long`（带符号的 64 位整数）。

  从更改日志流数据中开始读取的指定提交中的操作序列号。默认为 1。

响应

- format – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回的更改记录的序列化格式。目前，唯一支持的值是 PG_JSON。
- lastEventId – 必需：它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  流响应中最后一次更改的序列标识符。

  事件 ID 由两个字段组成：commitNum，标识更改图形的事务；opNum，标识该事务中的特定操作：

  Example

```
"eventId": {
          "commitNum": 12,
          "opNum": 1
        }
```

- lastTrxTimestampInMillis – 必需：长整型，类型为：`long`（带符号的 64 位整数）。

  请求提交事务的时间（使用 Unix 纪元时间表示，单位为毫秒）。
- records – 必填：PropertygraphRecord 对象的数组。

包含在响应中的序列化更改日志流记录的数组。

- totalRecords – 必需：一个整数，类型为：`integer`（带符号的 32 位整数）。

  响应中的记录总数。

错误

- UnsupportedOperationException
- ExpiredStreamException
- InvalidParameterException
- MemoryLimitExceededException
- StreamRecordsNotFoundException
- ClientTimeoutException
- PreconditionsFailedException
- ThrottlingException
- ConstraintViolationException
- InvalidArgumentException
- IllegalArgumentException
- TooManyRequestsException

流数据结构：

PropertygraphRecord（结构）

属性图记录的结构。

字段

- commitTimestampInMillis – 这是必需的：长整型，类型为：`long`（带符号的 64 位整数）。

  请求提交事务的时间（使用 Unix 纪元时间表示，单位为毫秒）。
- data – 这是必需的：一个 PropertygraphData 对象。

  序列化的 Gremlin 或 openCypher 更改记录。

- eventId – 这是必需的：它是键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  流更改记录的序列标识符。

- isLastOp - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  仅当此操作是其事务中的最后一个操作时才存在。如果存在，则设置为 true。它有助于确保整个事务都被消耗掉。

- op – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  造成更改的操作。

PropertygraphData（结构）

Gremlin 或 openCypher 更改记录。

字段

- from – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果这是一个边缘（类型 = `e`），则为相应的 `from` 顶点或源节点的 ID。

- id – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin 或 openCypher 元素的 ID。

- key – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  属性名称。对于元素标签，这是 `label`。

- to – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  如果这是一个边缘（类型 = `e`），则为相应的 `to` 顶点或目标节点的 ID。

- type – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Gremlin 或 openCypher 元素的类型。必须为以下值之一：

  - **v1** - Gremlin 的顶点标签或 openCypher 的节点标签。
  - **vp** - Gremlin 的顶点属性或 openCypher 的节点属性。

  - **e** - Gremlin 的边缘和边缘标签，或 openCypher 的关系和关系类型。
  - **ep** - Gremlin 的边缘属性或 openCypher 的关系属性。
- value – 这是必需的：文档，类型为：`document`（由类似 JSON 的数据模型表示的与协议无关的开放内容）。

  这是一个 JSON 对象，其中包含值本身的值字段和该值的 JSON 数据类型的数据类型字段：

  Example

```
"value": {
          "value": "(the new value"),
          "dataType": "(the JSON datatypenew value")
        }
```

Neptune 数据面板统计数据和图形摘要 API

属性图统计数据操作：

- GetPropertygraphStatistics（操作）
- ManagePropertygraphStatistics（操作）
- DeletePropertygraphStatistics（操作）
- GetPropertygraphSummary（操作）

统计数据结构：

- Statistics（结构）
- StatisticsSummary（结构）
- DeleteStatisticsValueMap（结构）
- RefreshStatisticsIdMap（结构）
- NodeStructure（结构）
- EdgeStructure（结构）
- SubjectStructure（结构）
- PropertygraphSummaryValueMap（结构）
- PropertygraphSummary（结构）

GetPropertygraphStatistics（操作）

此 API 的 AWS CLI 名称为：`get-propertygraph-statistics`。

获取属性图统计数据（Gremlin 和 openCypher）。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetStatisticsStatus IAM 操作的策略。

请求

- 无请求参数。

响应

- payload – 必填：一个 统计数据 对象。

  属性图数据的统计数据。
- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  请求的 HTTP 返回代码。如果请求成功，则代码为 200。有关常见错误的列表，请参阅 DFE 统计数据请求的常见错误代码。

错误

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException

- InvalidArgumentException
- MissingParameterException

ManagePropertygraphStatistics（操作）

此 API 的 AWS CLI 名称为：`manage-propertygraph-statistics`。

管理属性图统计数据的生成和使用。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ManageStatistics IAM 操作的策略。

请求

- mode（在 CLI 中：`--mode`）– StatisticsAutoGenerationMode，类型为：`string`（UTF-8 编码的字符串）。

  统计数据生成模式。其中之一：`DISABLE_AUTOCOMPUTE`、`ENABLE_AUTOCOMPUTE` 或 `REFRESH`，最后一个模式手动触发 DFE 统计数据生成。

响应

- payload – 一个 RefreshStatisticsIdMap 对象。

  这仅在刷新模式下才会返回。
- status – 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  请求的 HTTP 返回代码。如果请求成功，则代码为 200。

错误

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException

- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

DeletePropertygraphStatistics（操作）

此 API 的 AWS CLI 名称为：`delete-propertygraph-statistics`。

删除 Gremlin 和 openCypher（属性图）数据的统计数据。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:DeleteStatistics IAM 操作的策略。

请求

- 无请求参数。

响应

- payload – 一个 DeleteStatisticsValueMap 对象。

  删除有效负载。
- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消状态。
- statusCode – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  HTTP 响应代码：如果删除成功，则为 200；如果没有要删除的统计数据，则为 204。

错误

- BadRequestException

- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

GetPropertygraphSummary（操作）

此 API 的 AWS CLI 名称为：`get-propertygraph-summary`。

获取属性图的图形摘要。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetGraphSummary IAM 操作的策略。

请求

- mode（在 CLI 中：`--mode`）– GraphSummaryType，类型为：`string`（UTF-8 编码的字符串）。

  模式可以取两个值之一：`BASIC`（默认）和 `DETAILED`。

响应

- payload – 一个 PropertygraphSummaryValueMap 对象。

  包含属性图摘要响应的有效负载。
- statusCode – 一个整数，类型为：`integer`（带符号的 32 位整数）。

  请求的 HTTP 返回代码。如果请求成功，则代码为 200。

错误

- BadRequestException
- InvalidParameterException
- StatisticsNotAvailableException
- ClientTimeoutException
- AccessDeniedException
- IllegalArgumentException
- TooManyRequestsException
- UnsupportedOperationException
- PreconditionsFailedException
- ReadOnlyViolationException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException

统计数据结构：

Statistics（结构）

包含统计信息。在计划查询执行时，DFE 引擎使用有关 Neptune 图形中数据的信息来进行有效的权衡。这些信息采用统计数据的形式，包括可以指导查询计划的所谓特性集和谓词统计数据。请参阅管理 Neptune DFE 要使用的统计数据。

字段

- active - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  表示是否启用了 DFE 统计数据生成。
- autoCompute - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  表示是否启用了自动统计数据生成。
- date – 这是 SyntheticTimestamp_date_time，类型为：`string`（UTF-8 编码的字符串）。

  最近生成 DFE 统计数据的 UTC 时间。

- note – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  有关统计数据无效时出现的问题的注释。
- signatureInfo –这是一个 StatisticsSummary 对象。

  包含以下内容的 StatisticsSummary 结构：
  - `signatureCount` - 所有特性集中的签名总数。
  - `instanceCount` – 特性集实例的总数。
  - `predicateCount` – 唯一谓词的总数。
- statisticsId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  报告当前统计数据生成运行的 ID。值为 -1 表示未生成任何统计数据。

StatisticsSummary（结构）

有关统计数据中生成的特性集的信息。

字段

- instanceCount – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  特性集实例的总数。
- predicateCount – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  唯一谓词的总数。
- signatureCount – 这是一个整数，类型为：`integer`（带符号的 32 位整数）。

  所有特性集中的签名总数。

DeleteStatisticsValueMap（结构）

DeleteStatistics 的有效负载。

字段

- active - 这是一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  统计数据的当前状态。

- statisticsId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  当前正在运行的统计数据生成运行的 ID。

RefreshStatisticsIdMap（结构）

`REFRESH` 模式的统计数据。

字段

- statisticsId – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  当前正在运行的统计数据生成运行的 ID。

NodeStructure（结构）

节点结构。

字段

- count – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  具有此特定结构的节点数量。
- distinctOutgoingEdgeLabels – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此特定结构中存在的不同传出边缘标签列表。
- nodeProperties – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此特定结构中存在的节点属性的列表。

EdgeStructure（结构）

边缘结构。

字段

- count – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  具有此特定结构的边缘数量。

- edgeProperties – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此特定结构中存在的边缘属性的列表。

SubjectStructure（结构）

主题结构。

字段

- count – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  此特定结构的出现次数。
- predicates – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此特定结构中存在的谓词的列表。

PropertygraphSummaryValueMap（结构）

属性图摘要响应的有效负载。

字段

- graphSummary –这是一个 PropertygraphSummary 对象。

  图形摘要。
- lastStatisticsComputationTime – 这是 SyntheticTimestamp_date_time，类型为：`string`（UTF-8 编码的字符串）。

  Neptune 上次计算统计数据的时间戳，采用 ISO 8601 格式。
- version – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此图形摘要响应的版本。

PropertygraphSummary（结构）

图形摘要 API 会返回节点和边缘标签以及属性键的只读列表，以及节点、边缘和属性的计数。请参阅属性图 (PG) 的图形摘要响应。

字段

- edgeLabels – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  图形中不同边缘标签的列表。
- edgeProperties – 这是 LongValuedMap 对象。它是一个键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同边缘属性的列表以及使用每个属性的边缘计数。
- edgeStructures – 这是 EdgeStructure 对象数组。

  仅当请求的模式为 `DETAILED` 时，此字段才会出现。它包含边缘结构列表。
- nodeLabels – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  图形中不同节点标签的列表。
- nodeProperties – 这是 LongValuedMap 对象。它是一个键值对的映射数组，其中：

  每个键都是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  每个值都是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同节点属性的数量。
- nodeStructures – 这是 NodeStructure 对象数组。

  仅当请求的模式为 `DETAILED` 时，此字段才会出现。它包含节点结构的列表。
- numEdgeLabels – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同边缘标签的数量。
- numEdgeProperties – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同边缘属性的数量。
- numEdges – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中的边缘数。
- numNodeLabels – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同节点标签的数量。

- numNodeProperties – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中不同节点属性的列表，以及使用每个属性的节点计数。
- numNodes – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  图形中的节点数。
- totalEdgePropertyValues – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  所有边缘属性的总使用次数。
- totalNodePropertyValues – 这是长整型，类型为：`long`（有符号的 64 位整数）。

  所有节点属性的总使用次数。

Neptune ML 数据处理 API

数据处理操作：

- StartMLDataProcessingJob（操作）
- ListMLDataProcessingJobs（操作）
- GetMLDataProcessingJob（操作）
- CancelMLDataProcessingJob（操作）

ML 通用结构：

- MlResourceDefinition（结构）
- MlConfigDefinition（结构）

StartMLDataProcessingJob（操作）

此 API 的 AWS CLI 名称为：`start-ml-data-processing-job`。

创建新的 Neptune ML 数据处理任务，用于处理从 Neptune 导出的用于训练的图形数据。请参阅 `dataprocessing` 命令。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:StartMLModelDataProcessingJob IAM 操作的策略。

请求

- configFileName（在 CLI 中：`--config-file-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述如何加载导出的图形数据进行训练的数据规范文件。该文件由 Neptune 导出工具包自动生成。默认为 `training-data-configuration.json`。

- id（在 CLI 中：`--id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新任务的唯一标识符。默认值为自动生成的 UUID。

- inputDataS3Location（在 CLI 中：`--input-data-s3-location`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  您希望 SageMaker 下载运行数据处理任务所需数据的 Amazon S3 位置的 URI。

- modelType（在 CLI 中：`--model-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Neptune ML 当前支持的两种模型类型之一：异构图模型 (`heterogeneous`) 和知识图谱 (`kge`)。默认值为“无”。如果未指定，Neptune ML 会根据数据自动选择模型类型。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  SageMaker 可以代表您执行任务的 IAM 角色的 Amazon 资源名称 (ARN)。必须将其列在您的数据库集群参数组中，否则将发生错误。

- previousDataProcessingJobId（在 CLI 中：`--previous-data-processing-job-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  在较早版本的数据上运行的已完成数据处理任务的任务 ID。

- processedDataS3Location（在 CLI 中：`--processed-data-s3-location`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  您希望 SageMaker 保存数据处理任务结果的 Amazon S3 位置的 URI。

- processingInstanceType（在 CLI 中：`--processing-instance-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据处理期间使用的机器学习实例的类型。它的内存应该足够大，可以容纳处理后的数据集。默认为最小的 ml.r5 类型，其内存比磁盘上导出的图形数据大小大十倍。

- processingInstanceVolumeSizeInGB（在 CLI 中：`--processing-instance-volume-size-in-gb`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

处理实例的磁盘卷大小。输入数据和处理后的数据都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。默认值为 0。如果未指定或为 0，则 Neptune ML 会根据数据大小自动选择卷大小。

- processingTimeOutInSeconds（在 CLI 中：`--processing-time-out-in-seconds`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  数据处理任务的超时（以秒为单位）。默认值为 86400（1 天）。

- s3OutputEncryptionKMSKey（在 CLI 中：`--s-3-output-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  SageMaker 用于加密处理任务的输出的 Amazon Key Management Service（Amazon KMS）密钥。默认值为“无”。

- sagemakerIamRoleArn（在 CLI 中：`--sagemaker-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于执行 SageMaker 的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

- securityGroupIds（在 CLI 中：`--security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  VPC 安全组 ID。默认值为 None (无)。

- subnets（在 CLI 中：`--subnets`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Neptune VPC 中子网的 ID。默认值为 None (无)。

- volumeEncryptionKMSKey（在 CLI 中：`--volume-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Amazon Key Management Service (Amazon KMS) 密钥，SageMaker 使用它来加密连接到运行训练任务的 ML 计算实例的存储卷上的数据。默认值为 None (无)。

响应

- arn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据处理任务的 ARN。

- creationTimeInMillis – 长整型，类型为：`long`（有符号的 64 位整数）。

  创建新的处理任务所花费的时间，以毫秒为单位。

- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新数据处理任务的唯一 ID。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLDataProcessingJobs（操作）

此 API 的 AWS CLI 名称为：`list-ml-data-processing-jobs`。

返回 Neptune ML 数据处理任务的列表。请参阅使用 Neptune ML 数据处理命令列出处于活动状态的数据处理任务。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ListMLDataProcessingJobs IAM 操作的策略。

请求

- maxItems（在 CLI 中：`--max-items`）– ListMLDataProcessingJobsInputMaxItemsInteger，类型为：`integer`（带符号的 32 位整数），不小于 1 或大于 1024。

  要返回的最大项目数（从 1 到 1024；默认值为 10）。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- ids – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  列出数据处理任务 ID 的页面。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLDataProcessingJob（操作）

此 API 的 AWS CLI 名称为：`get-ml-data-processing-job`。

检索有关指定的数据处理任务的信息。请参阅 `dataprocessing 命令`。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:neptune-db:GetMLDataProcessingJobStatus IAM 操作的策略。

请求

- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

要检索的数据处理任务的唯一标识符。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此数据处理任务的唯一标识符。

- processingJob – 一个 MlResourceDefinition 对象。

  数据处理任务的定义。

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据处理任务的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLDataProcessingJob（操作）

此 API 的 AWS CLI 名称为：`cancel-ml-data-processing-job`。

取消 Neptune ML 数据处理任务。请参阅 `dataprocessing 命令`。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelMLDataProcessingJob IAM 操作的策略。

请求

- clean（在 CLI 中：`--clean`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 TRUE，则此标志指定在任务停止时应删除所有 Neptune ML S3 构件。默认为 FALSE。
- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  数据处理任务的唯一标识符。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消请求的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException

- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ML 通用结构：

MlResourceDefinition（结构）

定义 Neptune ML 资源。

字段

- arn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  资源 ARN。
- cloudwatchLogUrl – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  资源的 CloudWatch 日志 URL。
- failureReason – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  发生故障时的故障原因。
- name – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  资源名称。
- outputLocation – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  输出位置。
- status – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  资源状态。

MlConfigDefinition（结构）

包含 Neptune ML 配置。

字段

- arn – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  配置的 ARN。
- name – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  配置名称。

Neptune ML 模型训练 API

模型训练操作：

- StartMLModelTrainingJob（操作）
- ListMLModelTrainingJobs（操作）
- GetMLModelTrainingJob（操作）
- CancelMLModelTrainingJob（操作）

模型训练结构：

- CustomModelTrainingParameters（结构）

StartMLModelTrainingJob（操作）

此 API 的 AWS CLI 名称为：`start-ml-model-training-job`。

创建新的 Neptune ML 模型训练任务。请参阅使用 `modeltraining` 命令进行模型训练。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:StartMLModelTrainingJob IAM 操作的策略。

请求

- baseProcessingInstanceType（在 CLI 中：`--base-processing-instance-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于准备和管理机器学习模型训练的机器学习实例的类型。这是根据用于处理训练数据和模型的内存要求选择的 CPU 实例。

- customModelTrainingParameters（在 CLI 中：`--custom-model-training-parameters`）– CustomModelTrainingParameters 对象。

  自定义模型训练的配置。这是 JSON 对象。

- dataProcessingJobId（在 CLI 中：`--data-processing-job-id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  已完成的数据处理任务的任务 ID，该任务已创建训练将使用的数据。

- enableManagedSpotTraining（在 CLI 中：`--enable-managed-spot-training`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  使用 Amazon Elastic Compute Cloud 竞价型实例优化训练机器学习模型的成本。默认为 `False`。

- id（在 CLI 中：`--id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新任务的唯一标识符。默认值为自动生成的 UUID。

- maxHPONumberOfTrainingJobs（在 CLI 中：`--max-hpo-number-of-training-jobs`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  超参数调整任务要启动的最大训练任务总数。默认值为 2。Neptune ML 会自动调整机器学习模型的超参数。要获得性能良好的模型，请至少使用 10 个任务（换句话说，将 `maxHPONumberOfTrainingJobs` 设置为 10）。通常，调整次数越多，结果越好。

- maxHPOParallelTrainingJobs（在 CLI 中：`--max-hpo-parallel-training-jobs`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  为超参数调整任务启动的最大并行训练任务数。默认值为 2。您可以运行的并行任务数量受训练实例上可用资源的限制。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

- previousModelTrainingJobId（在 CLI 中：`--previous-model-training-job-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  已完成的模型训练任务的任务 ID，您要根据更新的数据以递增方式更新该任务。

- s3OutputEncryptionKMSKey（在 CLI 中：`--s-3-output-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

SageMaker 用于加密处理任务的输出的 Amazon Key Management Service (KMS) 密钥。默认值为“无”。

- sagemakerIamRoleArn（在 CLI 中：`--sagemaker-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于 SageMaker 执行的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

- securityGroupIds（在 CLI 中：`--security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  VPC 安全组 ID。默认值为 None (无)。

- subnets（在 CLI 中：`--subnets`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Neptune VPC 中子网的 ID。默认值为 None (无)。

- trainingInstanceType（在 CLI 中：`--training-instance-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于模型训练的 ML 实例的类型。所有 Neptune ML 模型都支持 CPU、GPU 和多 GPU 训练。默认为 `ml.p3.2xlarge`。为训练选择正确的实例类型取决于任务类型、图形大小和预算。

- trainingInstanceVolumeSizeInGB（在 CLI 中：`--training-instance-volume-size-in-gb`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  训练实例的磁盘卷大小。输入数据和输出模型都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。默认值为 0。如果未指定或为 0，Neptune ML 会根据数据处理步骤中生成的建议选择磁盘卷大小。

- trainingTimeOutInSeconds（在 CLI 中：`--training-time-out-in-seconds`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

  训练任务的超时（以秒为单位）。默认值为 86400（1 天）。

- trainModelS3Location（在 CLI 中：`--train-model-s3-location`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Amazon S3 中要存储模型构件的位置。

- volumeEncryptionKMSKey（在 CLI 中：`--volume-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

Amazon Key Management Service (KMS) 密钥，SageMaker 使用它来加密连接到运行训练任务的 ML 计算实例的存储卷上的数据。默认值为 None (无)。

响应

- arn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新模型训练任务的 ARN。
- creationTimeInMillis – 长整型，类型为：`long`（有符号的 64 位整数）。

  模型训练任务创建时间，以毫秒为单位。
- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新模型训练任务的唯一 ID。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLModelTrainingJobs（操作）

此 API 的 AWS CLI 名称为：`list-ml-model-training-jobs`。

列出 Neptune ML 模型训练任务。请参阅使用 `modeltraining` 命令进行模型训练。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:neptune-db:ListMLModelTrainingJobs IAM 操作的策略。

请求

- maxItems（在 CLI 中：`--max-items`）– ListMLModelTrainingJobsInputMaxItemsInteger，类型为：`integer`（带符号的 32 位整数），不小于 1 或大于 1024。

  要返回的最大项目数（从 1 到 1024；默认值为 10）。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- ids – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  模型训练任务 ID 列表的页面。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLModelTrainingJob（操作）

此 API 的 AWS CLI 名称为：`get-ml-model-training-job`。

检索有关 Neptune ML 模型训练任务的信息。请参阅使用 `modeltraining` 命令进行模型训练。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetMLModelTrainingJobStatus IAM 操作的策略。

请求

- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要检索的模型训练任务的唯一标识符。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- hpoJob – 一个 MlResourceDefinition 对象。

  HPO 任务。
- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  此模型训练任务的唯一标识符。
- mlModels – MlConfigDefinition 对象的数组。

  正在使用的机器学习模型的配置列表。
- modelTransformJob – 一个 MlResourceDefinition 对象。

  模型转换任务。
- processingJob – 一个 MlResourceDefinition 对象。

  数据处理任务。
- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  模型训练任务的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLModelTrainingJob（操作）

此 API 的 AWS CLI 名称为：`cancel-ml-model-training-job`。

取消 Neptune ML 模型训练任务。请参阅使用 `modeltraining` 命令进行模型训练。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelMLModelTrainingJob IAM 操作的策略。

请求

- clean（在 CLI 中：`--clean`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果设置为 TRUE，则此标志指定在任务停止时应删除所有 Amazon S3 构件。默认为 FALSE。
- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要取消的模型训练任务的唯一标识符。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

模型训练结构：

CustomModelTrainingParameters（结构）

包含自定义模型训练参数。请参阅 Neptune ML 中的自定义模型。

字段

- sourceS3DirectoryPath – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 `model-hpo-configuration.json` 文件。
- trainingEntryPointScript – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  执行模型训练并将超参数作为命令行参数（包括固定的超参数）的脚本模块中入口点的名称。默认为 `training.py`。

- transformEntryPointScript – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。默认值为 `transform.py`。

Neptune ML 学习模型转换 API

模型转换操作：

- StartMLModelTransformJob（操作）
- ListMLModelTransformJobs（操作）
- GetMLModelTransformJob（操作）
- CancelMLModelTransformJob（操作）

模型转换结构：

- CustomModelTransformParameters（结构）

StartMLModelTransformJob（操作）

此 API 的 AWS CLI 名称为：`start-ml-model-transform-job`。

创建新的模型转换任务。请参阅使用经过训练的模型生成新的模型构件。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:StartMLModelTransformJob IAM 操作的策略。

请求

- baseProcessingInstanceType（在 CLI 中：`--base-processing-instance-type`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于准备和管理机器学习模型训练的机器学习实例的类型。这是根据用于处理训练数据和模型的内存要求选择的 ML 计算实例。

- baseProcessingInstanceVolumeSizeInGB（在 CLI 中：`--base-processing-instance-volume-size-in-gb`）– 一个整数，类型为：`integer`（带符号的 32 位整数）。

训练实例的磁盘卷大小（以 GB 为单位）。默认值为 0。输入数据和输出模型都存储在磁盘上，因此卷大小必须足够大，以容纳两个数据集。如果未指定或为 0，Neptune ML 会根据数据处理步骤中生成的建议选择磁盘卷大小。

- customModelTransformParameters（在 CLI 中：`--custom-model-transform-parameters`）– CustomModelTransformParameters 对象。

  使用自定义模型进行模型转换的配置信息。`customModelTransformParameters` 对象包含以下字段，这些字段的值必须与训练任务中保存的模型参数兼容：

- dataProcessingJobId（在 CLI 中：`--data-processing-job-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  已完成的数据处理任务的任务 ID。您必须包含 `dataProcessingJobId` 和 `mlModelTrainingJobId` 或 `trainingJobName`。

- id（在 CLI 中：`--id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新任务的唯一标识符。默认值为自动生成的 UUID。

- mlModelTrainingJobId（在 CLI 中：`--ml-model-training-job-id`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  已完成的模型训练任务的任务 ID。您必须包含 `dataProcessingJobId` 和 `mlModelTrainingJobId` 或 `trainingJobName`。

- modelTransformOutputS3Location（在 CLI 中：`--model-transform-output-s3-location`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Amazon S3 中要存储模型构件的位置。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

- s3OutputEncryptionKMSKey（在 CLI 中：`--s-3-output-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  SageMaker 用于加密处理任务的输出的 Amazon Key Management Service (KMS) 密钥。默认值为“无”。

- sagemakerIamRoleArn（在 CLI 中：`--sagemaker-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  用于执行 SageMaker 的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

- securityGroupIds（在 CLI 中：`--security-group-ids`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  VPC 安全组 ID。默认值为 None (无)。

- subnets（在 CLI 中：`--subnets`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Neptune VPC 中子网的 ID。默认值为 None (无)。

- trainingJobName（在 CLI 中：`--training-job-name`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  已完成的 SageMaker 训练任务的名称。您必须包含 `dataProcessingJobId` 和 `mlModelTrainingJobId` 或 `trainingJobName`。

- volumeEncryptionKMSKey（在 CLI 中：`--volume-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Amazon Key Management Service (KMS) 密钥，SageMaker 使用它来加密连接到运行训练任务的 ML 计算实例的存储卷上的数据。默认值为 None (无)。

响应

- arn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  模型转换任务的 ARN。

- creationTimeInMillis – 长整型，类型为：`long`（有符号的 64 位整数）。

  模型转换任务的创建时间，以毫秒为单位。

- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新模型转换任务的唯一 ID。

错误

- UnsupportedOperationException

- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLModelTransformJobs（操作）

此 API 的 AWS CLI 名称为：`list-ml-model-transform-jobs`。

返回模型转换任务 ID 的列表。请参阅使用经过训练的模型生成新的模型构件。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ListMLModelTransformJobs IAM 操作的策略。

请求

- maxItems（在 CLI 中：`--max-items`）– ListMLModelTransformJobsInputMaxItemsInteger，类型为：`integer`（带符号的 32 位整数），不小于 1 或大于 1024。

  要返回的最大项目数（从 1 到 1024；默认值为 10）。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- ids – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  模型转换 ID 列表中的一页。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLModelTransformJob（操作）

此 API 的 AWS CLI 名称为：`get-ml-model-transform-job`。

获取有关指定的模型转换任务的信息。请参阅使用经过训练的模型生成新的模型构件。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetMLModelTransformJobStatus IAM 操作的策略。

请求

- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要检索的模型转换的唯一标识符。

- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- baseProcessingJob – 一个 MlResourceDefinition 对象。

  基础数据处理任务。

- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要检索的模型转换任务的唯一标识符。

- models – MlConfigDefinition 对象的数组。

  所用模型的配置信息列表。

- remoteModelTransformJob – 一个 MlResourceDefinition 对象。

  远程模型转换任务。

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  模型转换任务的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

CancelMLModelTransformJob（操作）

此 API 的 AWS CLI 名称为：`cancel-ml-model-transform-job`。

取消指定的模型转换任务。请参阅使用经过训练的模型生成新的模型构件。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CancelMLModelTransformJob IAM 操作的策略。

请求

- clean（在 CLI 中：`--clean`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果将此标志设置为 TRUE，则应在任务停止时删除所有 Neptune ML S3 构件。默认为 FALSE。
- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  要取消的模型转换任务的唯一 ID。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  取消的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

模型转换结构：

CustomModelTransformParameters（结构）

包含自定义模型转换参数。请参阅使用经过训练的模型生成新的模型构件。

字段

- sourceS3DirectoryPath – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  实现您的模型的 Python 模块所在的 Amazon S3 位置的路径。这必须指向有效的现有 Amazon S3 位置，其中至少包含训练脚本、转换脚本和 `model-hpo-configuration.json` 文件。
- transformEntryPointScript – 这是一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  脚本模块中入口点的名称，该脚本应在确定超参数搜索中的最佳模型之后运行，以计算模型部署所需的模型构件。它应该能够在没有命令行参数的情况下运行。默认为 `transform.py`。

Neptune ML 推理端点 API

推理端点操作：

- CreateMLEndpoint（操作）
- ListMLEndpoints（操作）
- GetMLEndpoint（操作）
- DeleteMLEndpoint（操作）

CreateMLEndpoint（操作）

此 API 的 AWS CLI 名称为：`create-ml-endpoint`。

创建新的 Neptune ML 推理端点，此推理端点允许您查询模型训练过程构造的一个特定模型。请参阅使用端点命令管理推理端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:CreateMLEndpoint IAM 操作的策略。

请求

- id (在 CLI 中：`--id`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  新推理端点的唯一标识符。默认为自动生成的带有时间戳的名称。
- instanceCount (在 CLI 中：`--instance-count`) – 一个整数，类型为：`integer` (带符号的 32 位整数) 。

  部署到端点以进行预测的最少 Amazon EC2 实例数量。默认值为 1
- instanceType (在 CLI 中：`--instance-type`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  用于在线服务的 Neptune ML 实例的类型。默认为 `ml.m5.xlarge`。为推理端点选择 ML 实例取决于任务类型、图形大小和预算。
- mlModelTrainingJobId (在 CLI 中：`--ml-model-training-job-id`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  已完成的模型训练任务的任务 ID，该任务创建了推理端点将指向的模型。您必须提供 `mlModelTrainingJobId` 或 `mlModelTransformJobId`。
- mlModelTransformJobId (在 CLI 中：`--ml-model-transform-job-id`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  已完成的模型转换任务的任务 ID。您必须提供 `mlModelTrainingJobId` 或 `mlModelTransformJobId`。
- modelName (在 CLI 中：`--model-name`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  用于训练的模型类型。默认情况下，Neptune ML 模型是根据数据处理中使用的 `modelType` 自动生成的，但您可以在此处指定不同的模型类型。默认情况下，`rgcn` 用于异构图形，`kge` 用于知识图谱。对于异构图，唯一有效值为 `rgcn`。知识图谱的有效值为：`kge`、`transe`、`distmult` 和 `rotate`。
- neptuneIamRoleArn (在 CLI 中：`--neptune-iam-role-arn`) – 一个字符串，类型为：`string` (UTF-8 编码的字符串) 。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将引发错误。
- update (在 CLI 中：`--update`) – 一个布尔值，类型为：`boolean` [布尔值 (true 或 false)]。

  如果设置为 `true`，则 `update` 表示这是更新请求。默认为 `false`。您必须提供 `mlModelTrainingJobId` 或 `mlModelTransformJobId`。

- volumeEncryptionKMSKey（在 CLI 中：`--volume-encryption-kms-key`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  Amazon Key Management Service (Amazon KMS) 密钥，SageMaker 使用它来加密连接到运行训练任务的 ML 计算实例的存储卷上的数据。默认值为 None (无)。

响应

- arn – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新推理端点的 ARN。
- creationTimeInMillis – 长整型，类型为：`long`（有符号的 64 位整数）。

  端点创建时间，以毫秒为单位。
- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  新推理端点的唯一 ID。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

ListMLEndpoints（操作）

此 API 的 AWS CLI 名称为：`list-ml-endpoints`。

列出现有的推理端点。请参阅使用端点命令管理推理端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:ListMLEndpoints IAM 操作的策略。

请求

- maxItems（在 CLI 中：`--max-items`）– ListMLEndpointsInputMaxItemsInteger，类型为：`integer`（带符号的 32 位整数），不小于 1 或大于 1024。

  要返回的最大项目数（从 1 到 1024；默认值为 10）。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- ids – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  推理端点 ID 列表中的一页。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

GetMLEndpoint（操作）

此 API 的 AWS CLI 名称为：`get-ml-endpoint`。

检索有关推理端点的详细信息。请参阅使用端点命令管理推理端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:GetMLEndpointStatus IAM 操作的策略。

请求

- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  推理端点的唯一标识符。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将发生错误。

响应

- endpoint – 一个 MlResourceDefinition 对象。

  端点定义。
- endpointConfig – 一个 MlConfigDefinition 对象。

  端点配置。
- id – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  推理端点的唯一标识符。
- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  推理端点的状态。

错误

- UnsupportedOperationException
- BadRequestException

- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

DeleteMLEndpoint（操作）

此 API 的 AWS CLI 名称为：`delete-ml-endpoint`。

取消创建 Neptune ML 推理端点。请参阅使用端点命令管理推理端点。

在启用了 IAM 身份验证的 Neptune 集群中调用此操作时，发出请求的 IAM 用户或角色必须附加允许在该集群中执行 neptune-db:DeleteMLEndpoint IAM 操作的策略。

请求

- clean（在 CLI 中：`--clean`）– 一个布尔值，类型为：`boolean` [布尔值（true 或 false）]。

  如果将此标志设置为 `TRUE`，则应在任务停止时删除所有 Neptune ML S3 构件。默认为 `FALSE`。
- id（在 CLI 中：`--id`）– 必需：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  推理端点的唯一标识符。
- neptuneIamRoleArn（在 CLI 中：`--neptune-iam-role-arn`）– 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  向 Neptune 提供对 SageMaker 和 Amazon S3 资源的访问权限的 IAM 角色的 ARN。必须将其列在您的数据库集群参数组中，否则将引发错误。

响应

- status – 一个字符串，类型为：`string`（UTF-8 编码的字符串）。

取消的状态。

错误

- UnsupportedOperationException
- BadRequestException
- MLResourceNotFoundException
- InvalidParameterException
- ClientTimeoutException
- PreconditionsFailedException
- ConstraintViolationException
- InvalidArgumentException
- MissingParameterException
- IllegalArgumentException
- TooManyRequestsException

Neptune 数据面板 API 异常

例外：

- AccessDeniedException（结构）
- BadRequestException（结构）
- BulkLoadIdNotFoundException（结构）
- CancelledByUserException（结构）
- ClientTimeoutException（结构）
- ConcurrentModificationException（结构）
- ConstraintViolationException（结构）
- ExpiredStreamException（结构）
- FailureByQueryException（结构）
- IllegalArgumentException（结构）
- InternalFailureException（结构）

AccessDeniedException（结构）

在身份验证或授权失败时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

返回 HTTP 状态代码，带有异常。

- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

BadRequestException（结构）

当提交了无法处理的请求时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。

- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  不正确的请求的 ID。

BulkLoadIdNotFoundException（结构）

当找不到指定的批量加载任务 ID 时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。

- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

找不到的批量加载任务 ID

CancelledByUserException（结构）

当用户取消请求时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ClientTimeoutException（结构）

当客户端中的请求发生超时时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ConcurrentModificationException（结构）

当请求尝试修改其它进程同时修改的数据时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ConstraintViolationException（结构）

当请求字段中的值不满足所需的约束条件时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ExpiredStreamException（结构）

当请求尝试访问已过期的流时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

FailureByQueryException（结构）

请求失败时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

IllegalArgumentException（结构）

不支持请求中的参数时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

InternalFailureException（结构）

请求处理意外失败时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

InvalidArgumentException（结构）

当请求中的参数具有无效值时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

InvalidNumericDataException（结构）

在为请求提供服务时遇到无效的数值数据时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

InvalidParameterException（结构）

当参数值无效时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  包含无效参数的请求的 ID。

LoadUrlAccessDeniedException（结构）

当拒绝访问指定的加载 URL 时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

MalformedQueryException（结构）

当提交的查询语法不正确或未通过额外验证时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  格式错误的查询请求的 ID。

MemoryLimitExceededException（结构）

当请求因内存资源不足而失败时引发。可以重试该请求。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  失败的请求的 ID。

MethodNotAllowedException（结构）

当所使用的端点不支持请求使用的 HTTP 方法时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

MissingParameterException（结构）

缺少必需的参数时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  缺少参数的请求的 ID。

MLResourceNotFoundException（结构）

当找不到指定的机器学习资源时引发。

字段

- code – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：string（UTF-8 编码的字符串）。

  相关请求的 ID。

ParsingException（结构）

遇到解析问题时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

PreconditionsFailedException（结构）

当处理请求的先决条件未得到满足时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

QueryLimitExceededException（结构）

当活动查询的数量超过服务器可以处理的数量时引发。当系统不太繁忙时，可以重试有问题的查询。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  超出限制的请求的 ID。

QueryLimitException（结构）

当查询大小超过系统限制时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  超出限制的请求的 ID。

QueryTooLargeException（结构）

当查询的正文太大时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  过大的请求的 ID。

ReadOnlyViolationException（结构）

当请求尝试写入只读资源时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  缺少参数的请求的 ID。

S3Exception（结构）

在访问 Amazon S3 而出现问题时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ServerShutdownException（结构）

当服务器在处理请求时关闭时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

StatisticsNotAvailableException（结构）

当满足请求所需的统计数据不可用时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

StreamRecordsNotFoundException（结构）

当找不到查询所请求的流记录时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

ThrottlingException（结构）

当请求的速率超出吞吐量上限时引发。遇到此异常后可以重试请求。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由于此原因而无法处理的请求的 ID。

TimeLimitExceededException（结构）

当操作超过允许的时间限制时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由于此原因而无法处理的请求的 ID。

TooManyRequestsException（结构）

当正在处理的请求数量超过限制时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。

- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  由于此原因而无法处理的请求的 ID。

UnsupportedOperationException（结构）

当请求尝试启动不支持的操作时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

UnloadUrlAccessDeniedException（结构）

当被拒绝访问作为卸载目标的 URL 时引发。

字段

- code – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  返回 HTTP 状态代码，带有异常。
- detailedMessage – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  描述问题的详细消息。
- requestId – 这是必需的：一个字符串，类型为：`string`（UTF-8 编码的字符串）。

  相关请求的 ID。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。