为亚马逊 Neptune 应用 AWS Well-Architected 框架

AWS 规范性指导

AWS 规范性指导: 为亚马逊 Neptune 应用 AWS Well-Architected 框架

Table of Contents

简介 1
目标受众 1
目标 1
卓越运营支柱 3
使用 IaC 方法自动部署 3
频繁进行可逆的小规模更改 3
预测故障 4
从所有操作故障中吸取教训 5
使用日志记录功能来监控未经授权或异常的活动 5
安全支柱 6
实现数据安全 6
保护网络安全 7
实现身份验证和授权 7
可靠性支柱 9
了解 Neptune 服务配额 9
了解 Neptune 部署模式 10
管理和扩展 Neptune 集群 10
管理备份和失效转移事件 11
性能效率支柱 12
了解图形建模 12
优化查询 13
优化集群规模 14
优化写入 15
成本优化支柱 17
了解使用模式和所需的服务 17
选择资源时要注意成本 18
为您的工作负载选择最佳 Neptune 实例配置 19
适当调整数据存储和传输的大小 20
可持续发展支柱 21
AWS 区域 选择 21
基于用户行为模式的消费 21
优化软件开发和架构模式 22
资源 23
参考 23

博客文章 23
免费的“ AWS 技能大师”课程 23
贡献者 24
文档历史记录 25
..... xxvi

为亚马逊 Neptune 应用 AWS Well-Architected 框架

Amazon Web Services（贡献者）

2026 年 1 月（文件历史记录）

您可以使用 Amazon Neptune 在 Amazon Web Services（AWS）上构建基于图形的解决方案。本指南为在规划 Neptune 部署时应用 AWS Well-Architected Framework 原则提供了规范指引。

Wel AWS l-Architected Framework 可帮助您为各种应用程序和工作负载构建安全、高性能、弹性和高效的基础架构。它还为您提供了评估架构和实施可扩展设计的一致方法。

Well AWS -Architected 框架围绕以下六大支柱构建：

- 卓越运营
- 安全性
- 可靠性
- 性能效率
- 成本优化
- 可持续性

本指南提供了来自 Well-Architect AWS ed Framework 设计支柱和最佳实践的信息，以及部署 Neptune 时需要记住的注意事项。 AWS

目标受众

本指南适用于设计和实施使用 AWS上的图表的解决方案的数据工程师、解决方案架构师和数据分析师。

目标

本指南可以帮助您和您的组织执行以下操作：

- 根据您的使用案例和查询模式，选择支持的部署选项和查询语言。
- 遵循 Wel AWS l-Architected 的设计模式，这将有助于提高灵活性和安全性。
- 设计查询以获得最佳性能并节省成本。

- 了解如何在生产环境中管理 Neptune 集群时提高运营效率。

卓越运营支柱

Well-Architect AWS ed Framework 的卓越运营支柱侧重于运行和监控系统，以及不断改进流程和程序。其中包括有效地支持开发和运行工作负载的能力，获取对运营的洞察，以及不断改进支持流程和程序以实现业务价值。您可以通过自我修复工作负载降低运营复杂性，这些工作负载无需人工干预即可检测和修复大多数问题。您可以通过遵循本节中描述的最佳实践来努力实现这一目标。当您的工作负载偏离预期行为时 APIs，使用 Amazon Neptune 指标和机制来正确响应。

本次对卓越运营支柱的讨论侧重于以下关键领域：

- 基础设施即代码（IaC）
- 变更管理
- 韧性策略
- 事件管理
- 合规性审计报告
- 日志记录和监控

使用 IaC 方法自动部署

使用 IaC 在 Neptune 上自动部署的最佳实践包括：

- 尽可能应用基础设施即代码 (IaC) 来部署 Neptune 集群。为了实现一致的环境配置，请使用AWS CloudFormation模板或 HashiCorp Terraform 为集群创建所有必需的资源。AWS Cloud Development Kit (AWS CDK)
- 尽可能自动执行 Neptune 操作程序，例如调整实例大小、添加或删除只读副本，或者对全局表进行手动失效转移。
- 将连接字符串存储在客户端外部。使用提取、转换和加载 (ETL) 流程来促进 blue/green 部署策略、灾难恢复 (DR) 以及向新集群的近乎零的停机迁移。连接字符串可以存储在 AWS Secrets Manager、Amazon DynamoDB 或任何可以动态更改它们的位置。
- 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，请参阅标记 Amazon Neptune 资源。

频繁进行可逆的小规模更改

以下建议侧重于小的、可逆的更改，以最大限度地降低复杂性并降低工作负载中断的可能性：

- 将 IaC 模板和脚本存储在源代码控制服务中，例如 GitHub 或 GitLab。

> **Important**
>
> 不要在源代码管理中存储 AWS 凭据。

- 要求 IaC 部署才能使用持续集成和持续交付（CI/CD）服务，例如 AWS CodePipeline 或 AWS CodeBuild。这些服务会在包含临时 Neptune 集群的非生产环境中编译、测试和部署代码，然后再影响您的生产 Amazon Neptune 集群。
- 在将基础设施和应用程序查询部署到生产环境之前，先在较低的环境中对其进行测试。这将最大限度地减少中断的可能性，并有助于确保它们在您的工作负载和扩展下表现良好。

预测故障

自我修复的基础设施通过预测故障并尝试在没有干预的情况下解决任何问题来体现卓越运营。以下建议可以帮助您使用 Neptune 实现该成熟度：

- 创建使用 Amazon CloudWatch 指标监控数据库实例的 CPU 和内存使用情况并了解使用模式的监控计划。为应用程序日志中的关键指标和 Neptune 客户端响应创建 CloudWatch仪表板和警报。有关 CPU 利用率高或低指标的更多信息，请参阅 Neptune CloudWatch 文档中的使用在 Neptune 中监控数据库实例性能。

  如果您经常在查询中遇到 out-of-memory异常，可以考虑减少查询遍历的节点总数，或者尝试使用该X2系列中的实例，该实例的比例更高 RAM-to-CPU。
- 设置通知以监控 Neptune 集群的运行状况。例如，`BufferCacheHitRatio` 应始终处于高位（大于 99.9%），而 `MainRequestQueuePendingRequests` 应始终保持较低水平（理想情况下为 0，但取决于您的要求和延迟容忍度）。
- 考虑使用只读副本在 Neptune 内实现高可用性。您应该在与写入器实例不同的可用区中至少有两个只读副本，以确保在失效转移事件期间，实例始终可用于处理读取查询。
- 根据利用率指标自动扩展只读副本。有关更多信息，请参阅自动扩缩 Amazon Neptune 数据库集群中的副本数量。
- 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。
- 如果您的应用程序需要在完全 AWS 区域 中断后存活下来，请考虑将全局数据库作为灾难恢复计划的一部分。

从所有操作故障中吸取教训

自我修复基础设施是一项长期的工作，当出现罕见问题或响应效果不如预期时，它会不断迭代发展。采取以下实践有助于集中精力实现该目标：

- 从所有故障中吸取教训，推动改进。
- 在团队和组织内部分享经验教训。如果组织内有多个团队使用 Neptune，请创建一个公共聊天室或用户群组，以便分享经验和最佳实践。

使用日志记录功能来监控未经授权或异常的活动

要观察异常性能和活动模式，请将日志存储在 Ama CloudWatch zon 日志中。考虑下面的最佳实践：

- 启用慢速查询日志记录。定期查看日志并诊断某些查询缓慢的原因。使用适用于 Gremlin、SPARQL 或 openCypher 的 Neptune explain 和 profile 端点，来深入了解这些查询缓慢的原因。
- 启用 Neptune 审核日志，并定期查看日志中是否存在未经授权的访问或异常情况。
- 如果您使用的是慢查询日志记录或审核日志记录，请启用发布到 CloudWatch 日志。这将帮助您避免实例上的磁盘空间不足。Neptune 实例的日志存储容量有限，超过日志空间时会覆盖较旧的日志文件。 CloudWatch 日志支持长期保留日志。 CloudWatch 日志中增强的监控功能将提高您查询日志和诊断问题的能力。
- 为了便于使用更好的审计日志分析工具，您可以将 Neptune 数据库集群配置为在 Logs 中将审计日志数据发布到日志组中 CloudWatch 。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，请参阅将 Nep tune 日志发布到 Ama CloudWatch zon 日志。
- Neptune 支持使用 AWS CloudTrail记录控制面板操作。有关更多信息，请参阅使用记录亚马逊 Neptune API 调用。 AWS CloudTrail

安全支柱

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。责任共担模型将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云 AWS 服务 中运行的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划的一部分，第三方审计师定期测试和验证 AWS 安全的有效性。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 AWS 服务 使用的内容决定。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。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 Shared Responsibility Model and GDPR 博客文章。

安全支柱可帮助您了解在使用 Neptune 时如何应用分担责任模型。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 方法来监控和保护您的 Neptune 资源。

安全支柱包括以下关键重点领域：

- 数据安全性
- 网络安全
- 身份验证和授权

实现数据安全

数据泄露和违规行为会使您的客户处于危险之中，并可能对您的公司造成严重的负面影响。以下最佳实践有助于保护您的客户数据免遭无意和恶意泄露：

- 集群名称、标签、参数组、 AWS Identity and Access Management (IAM) 角色和其他元数据不应包含机密或敏感信息，因为这些数据可能会出现在账单或诊断日志中。
- URIs 或者指向作为数据存储在 Neptune 中的外部服务器的链接不应包含用于验证请求的凭据信息。
- 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 Neptune 加密来满足静态数据的合规性要求。

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” 部分中选择 “是”（默认选中），或者在中设置属性。AWS::Neptune::DBCluster::StorageEncrypted CloudFormation如果启用了加密，Neptune 将默认使用 Amazon Relational Database Service（Amazon RDS）AWS 托管式密钥，或者您可以创建客户自主管理型密钥。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。有关更多详细信息，请参阅静态加密 Neptune 资源。您的自动快照和手动快照使用的加密方式与您为 Neptune 集群选择的加密方式相同。

- 使用 SPARQL 和 OpenCypher 语言时，练习正确的输入验证和参数化技术，以防止 SQL 注入及其他形式的攻击。避免构造使用字符串连接和用户提供的输入的查询。使用参数化查询或预准备语句将输入参数安全地传递到图形数据库。有关更多信息，请参阅 openCypher 参数化查询示例和 SPARQL 注入防御。
- 对于 Gremlin 语言，请使用 Gremlin 语言变体，而不是直接传递基于字符串的 Gremlin 脚本，以避免潜在的注入问题。

保护网络安全

只能 AWS上在虚拟私有云（VPC）中创建 Amazon Neptune 数据库集群。在 Neptune 1.4.6.0 之前，Neptune 数据库集群的终端节点只能在该 VPC 内访问。从 Neptune 1.4.6.0 及更高版本开始，可以将 Neptune 实例配置为可通过互联网公开访问。最佳做法是仅在非生产环境中使用此功能，以简化开发人员对 Neptune 的访问（尽管始终需要启用 IAM 身份验证才能实现公共可访问性）。如果您启用了公共可访问性，请考虑将数据库端口的入站安全组规则设置为仅限已知的 IP 地址流量。在生产环境或包含敏感数据的集群中，通过不允许公众访问和限制对 Neptune 数据库集群所在的 VPC 的访问来保护您的 Neptune 数据。有关更多信息，请参阅连接到您的 Amazon Neptune 图形。

为了保护传输中数据，Neptune 使用安全协议和密码，通过 HTTPS 强制执行与任何实例或集群端点的 SSL 连接。Neptune 为 Neptune 数据库实例提供 SSL 证书。Neptune SSL 证书仅支持集群端点、读取器端点和实例端点主机名。

如果您使用的是负载平衡器或代理服务器（例如 HAProxy），则必须使用 SSL 终止并在代理服务器上拥有自己的 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。有关使用 SSL 连接到 Neptune 端点的更多信息，请参阅使用 HTTP REST 端点连接到 Neptune 数据库实例。

实现身份验证和授权

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，请启用 IAM 数据库身份验证并使用 IAM 证书。使用 IAM 凭证连接到 AWS 时，您的 IAM 角色必须具有授予执行 Neptune 管

理操作所需的权限的 IAM 策略。请确保遵循最低权限原则，仅授予完成任务所需的权限。有关更多信息，请参阅使用不同类型的 IAM 策略控制对 Neptune 的访问权限和使用临时凭证进行 IAM 身份验证。

要控制谁可以连接到 Neptune 集群并查询数据，您可以使用 IAM 对 Neptune 数据库实例或数据库集群进行身份验证。如果您在 Neptune 数据库集群中启用 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。有关启用 IAM 身份验证的步骤的更多信息，请参阅在 Neptune 中启用 IAM 数据库身份验证。

启用 IAM 数据库身份验证后，每个请求都必须使用 AWS 签名版本 4 进行签名。要了解如何向启用了 IAM 身份验证的所有 Neptune 端点发送已签名的请求，请参阅使用 AWS 签名版本 4 进行连接和签名。许多库和工具（例如 awscurl）已支持 AWS 签名版本 4。

为了与其他人互动 AWS 服务，Amazon Neptune 使用 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，并包含服务代表您调用其他 AWS 服务 所需的所有权限。有关更多信息，请参阅为 Neptune 使用服务相关角色。

可靠性支柱

可靠性支柱包括工作负载在预期时正确、一致地执行其预期功能的能力。这包括在其全部生命周期内运行和测试工作负载的能力。

可靠的工作负载始于前期的软件和基础设施设计决策。您的架构选择将影响您在所有 Well-Architecte AWS d 支柱上的工作负载行为。针对可靠性，您必须遵循特定的模式。

可靠性支柱重点关注以下关键领域：

- 工作负载架构，包括服务配额和部署模式
- 变更管理
- 故障管理

了解 Neptune 服务配额

除中国外，Neptune 集群的最大容量可以增长到 128 TiB (TiB)，中国 AWS 区域 除外，其配额为 64 T GovCloud iB。

128TiB 配额足以在图表中存储大约 2000-4000 亿个对象。在带标签的属性图（LPG）中，对象是节点、边缘，或者是节点或边缘上的属性。在资源描述框架（RDF）图中，对象是四元组。

对于任何 Neptune Serverless 集群，您可以设置海王星容量单位的最小和最大数量 ()。NCUs每个 NCU 由 2GiB 的内存及关联的 vCPU 和网络组成。这些最小和最大 NCU 值适用于集群中的任何无服务器实例。您可以设置的最大 NCU 值为 128.0 NCUs，最低的最小值为 1.0。 NCUs通过观察 Amazon CloudWatch 指标`ServerlessDatabaseCapacity`来优化最适合您的应用程序的 NCU 范围，捕捉您经常遇到的范围，并将该范围内的不良行为或成本关联起来。`NCUUtilization`在许多工作负载中，1.0 NCU 的起点太低，在一段时间不活动后会导致行为不可靠。如果您发现工作负载的扩展速度不够快，请增加最小值， NCUs 以便在扩展时为最初的激增提供足够的处理能力。

每个 AWS 账户 区域对您可以创建的数据库资源数量都有配额。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。有些配额是软配额，可以通过请求增加。有关 Amazon Neptune 和 Amazon RDS、Amazon Aurora 和 Amazon DocumentDB（兼容 MongoDB）之间共享的配额列表，以及请求增加配额的链接（如果有），请参阅 Amazon RDS 中的配额。

了解 Neptune 部署模式

在 Neptune 数据库集群中，有一个主数据库实例和最多 15 个 Neptune 副本。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 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。Neptune 副本可以从主数据库实例分载读取工作负载。

要实现高可用性，请使用只读副本。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。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（通常少于 30 秒），在此期间，对主实例的读写请求将失败，同时引发异常。要获得最高的可靠性，请考虑在不同的可用区中使用两个只读副本。如果可用区 1 中的主实例离线，则可用区 2 中的实例会升级为主实例，但是在这种情况发生时它无法处理查询。因此，在过渡期间，需要可用区 3 中的实例来处理读取查询。

如果您使用的是 Neptune 无服务器，则所有可用区中的读取器和写入器实例将根据数据库负载相互独立地纵向扩展和缩减。您可以将读取器实例的提升层设置为 0 或 1，这样它就可以随写入器实例的容量一起纵向扩展和缩减。这样，它随时可以接管当前工作负载。

如果您的应用程序遍布全球或需要多区域故障转移，请考虑使用 Neptun e 全局数据库。Amazon Neptune 全球数据库跨越多个数据库 AWS 区域，可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 Neptune 全局数据库由一个区域中的主数据库集群和位于不同区域的多达五个辅助数据库集群组成。

管理和扩展 Neptune 集群

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。通过自动扩缩，您的 Neptune 数据库集群可以应对工作负载的突然增加。工作负载减少时，自动扩缩会删除不必要的副本，这样您就不会为未使用的容量付费。请注意，启动新实例可能需要长达 15 分钟的时间，因此仅靠自动扩缩不足以应对需求的快速变化。

您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。

如果您遇到需求的快速变化，请考虑使用无服务器实例。无服务器实例可以在短时间内垂直扩展，而自动扩缩可以在较长的时间段内水平扩展。此配置提供了最佳的可扩展性，因为无服务器实例可以垂直扩展，而自动扩缩则实例化新的只读副本以处理超出单个无服务器实例最大容量的工作负载。有关 Amazon Neptune Serverless 容量扩展的更多信息，请参阅 Neptune 无服务器数据库集群中的容量扩展。

如果您的扩展需求在可预测的时间发生变化，则可以计划更改最小实例数、最大实例数和阈值，以更好地应对这些不断变化的需求。请记住至少提前 15 分钟安排横向扩展事件，以便这些实例在需要时可以上线。

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。修改参数组中的集群参数时，了解静态参数和动态参数之间的区别，以及如何和何时进行应用。使用状态端点查看当前应用的配置。

管理备份和失效转移事件

Neptune 自动备份您的集群卷，并在备份保留期内保留备份的数据。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。在创建或修改数据库集群时，可指定 1-35 天备份保留期。

要将备份保留期延长，您还可以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。

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。恢复数据库集群时，您需要提供用于恢复的数据库集群快照的名称，然后提供恢复所创建的新数据库集群的名称。

测试您的系统如何响应失效转移事件。使用 Neptune API 强制执行失效转移事件。如果需要模拟数据库实例故障以进行测试，或是在失效转移进行之后将操作恢复到原始可用区，通过失效转移重启十分有用。有关更多信息，请参阅配置和管理多可用区部署。当您重启数据库写入器集群时，它会失效转移到备用副本。重启 Neptune 副本不会初始化失效转移。

设计客户端时要注重可靠性。测试它们在失效转移事件期间的行为。使用指数回退逻辑在客户端中实现重试逻辑。可以在 Amazon Neptune AWS Lambda 函数示例下的文档中找到实现此逻辑的代码示例。

如果您有一组适用于多个数据库引擎的常见备份要求，请考虑使用 AWS Backup。

性能效率支柱

Well-Architect AWS ed Framework 的性能效率支柱侧重于如何在摄取或查询数据时优化性能。性能优化是一个循序渐进的持续过程，包括：

- 确认业务需求
- 衡量工作负载性能
- 识别性能不佳的组件
- 优化组件以满足您的业务需求

性能效率支柱提供了特定于使用案例的准则，可以帮助识别要使用的正确图表数据模型和查询语言。还包括将数据摄取到 Amazon Neptune 以及从 Amazon Neptune 使用数据时应遵循的最佳实践。

性能效率支柱侧重于以下关键领域：

- 图形建模
- 查询优化
- 优化集群规模
- 写入优化

了解图形建模

了解标签属性图（LPG）和资源描述框架（RDF）模型之间的区别。在大多数情况下，这是一个偏好问题。但是，在一些使用案例中，一种模型比另一种模型更适合。如果您需要了解连接图表中两个节点的路径，请选择 LPG。如果要跨 Neptune 集群或其他图三元组存储联合数据，请选择 RDF。

如果您正在构建软件即服务（SaaS）应用程序或需要多租户的应用程序，请考虑在数据模型中纳入租户的逻辑分离，而不是为每个集群设置一个租户。要实现该类型的设计，您可以使用 SPARQL 命名图形和标注策略，例如在标签前添加客户标识符或添加代表租户标识符的属性键值对。确保您的客户端层注入这些值以保持逻辑分离。有关多租户建议的更多信息，请参阅运行 Amazon ISVs Neptune 数据库的多租户指南。

查询的性能取决于在处理查询时需要评估的图形对象（节点、边缘、属性）的数量。因此，图形模型可能会对应用程序的性能产生重大影响。尽可能使用精细标签，并仅存储实现路径确定或筛选所需的属性。要获得更高的性能，请考虑预计算图表的各个部分，例如创建汇总节点或连接公共路径的更直接的边缘。

尽量避免在具有相同标签的边缘数异常多的节点之间导航。此类节点通常有数千个边缘（其中大多数节点的边缘数仅在几十条左右）。结果是计算和数据复杂度大大增加。在某些查询模式中，这些节点可能不会出现问题，但我们建议您以不同的方式对数据进行建模以避免这种情况，尤其是在您需要通过节点作为中间步骤进行导航时。您可以使用慢速查询日志来帮助识别跨越这些节点的查询。您可能会观察到比平均查询模式更高的延迟和数据访问指标，尤其是在使用调试模式时。

如果您的用例支持，请 IDs 为节点和边使用确定性节点，而不是使用 Neptune 为其分配随机 GUID 值。 IDs通过 ID 访问节点是最有效的方法。

优化查询

在 LPG 模型上，openCypher 和 Gremlin 语言可以互换使用。如果性能是首要考虑因素，请考虑交替使用这两种语言，因为对于特定的查询模式，其中一种语言的性能可能优于另一种语言。

Neptune 正在转换为其替代查询引擎（DFE）。openCypher 仅在 DFE 上运行，但可以选择使用查询注释将 Gremlin 和 SPARQL 查询设置为在 DFE 上运行。考虑在激活 DFE 的情况下测试您的查询，并比较不使用 DFE 时的查询模式性能。

Neptune 针对事务型查询进行了优化，这些查询从单个节点或一组节点开始，然后从那里扇出，而不是评估整个图表的分析查询。对于您的分析查询工作负载，请使用 Neptune Analytics。Neptune Analytics 是需要快速迭代数据、分析和算法处理的研究、探索性或数据科学工作负载的理想选择。它还可以对图形数据执行矢量搜索，并可以直接从 Neptune 数据库实例加载数据。如果 Neptune Analytics 不能满足你的需求，你也可以考虑使用适用于 Pandas 的AWS SDK，或者将 nept une-export 与或 Amazon EMR 结合使用。AWS Glue

要识别模型和查询中的效率低下和瓶颈，请使用每种查询语言的`profile`和`explain` APIs 来获取查询计划和查询指标的详细解释。有关更多信息，请参阅 Gremlin profile、openCypher explain 和 SPARQL explain。

了解您的查询模式。如果图形中的不同边缘数太大，默认 Neptune 访问策略可能会效率低下。以下查询可能会变得非常低效：

- 未提供边缘标签时跨边缘向后导航的查询。
- 内部使用相同模式的子句（例如 Gremlin 中的 `.both()`），或者任何语言中用于删除节点的子句（这需要在不知道标签的情况下删除传入的边缘）。
- 在不指定属性标签的情况下访问属性值的查询。这些查询可能会变得非常低效。如果您的使用模式符合这种情况，请考虑启用 OSGP 索引（对象、主题、图表、谓词）。

使用慢速查询日志记录来识别慢速查询。查询速度慢可能是由于未优化的查询计划或不必要的大量索引查找造成的，这可能会增加成本。 I/O 适用于 Gremlin、SPARQL 或 openCypher 的 Neptune explain 和 profile 端点可以帮助您了解为什么这些查询速度很慢。原因可能包括：

- 与图中的平均节点相比，边缘数异常多的节点（例如，数千条和十条）可能会增加计算复杂性，从而延长延迟并增加资源消耗。确定这些节点的建模是否正确，或者是否可以改进访问模式，以减少必须遍历的边缘数。
- 未优化的查询将包含警告，提示特定步骤未经过优化。重写这些查询以使用优化步骤可能会提高性能。
- 冗余筛选条件可能会导致不必要的索引查找。同样，冗余模式可能会导致重复的索引查找，而这可以通过改进查询来优化（请参阅配置文件输出中的 `Index Operations - Duplication ratio`）。
- 某些语言（例如 Gremlin）没有强类型的数值，而是使用类型提升。例如，如果值为 55，则 Neptune 会查找与 55 等效的整数、长数、浮点数和其他数值类型的值。这会导致其他操作。如果您事先知道自己的类型匹配，则可以使用查询提示来避免这种情况。
- 您的图形模型会极大地影响性能。考虑通过使用更精细的标签或预计算多跳线性路径的快捷方式，来减少需要评估的对象数量。

如果仅靠查询优化无法满足性能要求，请考虑将各种缓存技术与 Neptune 配合使用来满足这些要求。

每个版本的 Neptune 性能都在不断提高。查看发行说明，了解每个版本的改进细节。考虑计划定期更新 Neptune 数据库集群，以帮助实现最佳性能。较新的版本也支持较新的实例。考虑升级到 1.4.5.0 或更高版本，以便能够使用这些实例。`r8g`有关这如何提高工作负载性能的更多信息，请参阅使用 Amazon N eptune v1.4.5 的 G AWS raviton4 r8g 实例的写入查询性价比提高 4.7 倍。

优化集群规模

根据您的并发和吞吐量要求调整集群规模。集群中每个实例可以处理的并发查询数量等于该实例上虚拟 CPUs (vCPUs) 数量的两倍。在所有 Worker 都被占用时到达的额外查询将放入服务器端队列。当工作线程可用时，将在 first-in-first-out (FIFO) 基础上处理这些查询。A `MainRequestQueuePendingRequests` mazon CloudWatch 指标显示每个实例的当前队列深度。如果此值经常大于零，请考虑选择一个具有更多 v 的实例CPUs。 如果队列深度超过 8,192，Neptune 将返回`ThrottlingException`错误。

每个实例的大约 65% 的 RAM 是为缓冲区缓存预留的。缓冲区缓存保存数据的工作数据集（不是整个图表；只是正在查询的数据）。要确定从缓冲区缓存而不是存储中提取的数据的百分比，请监控该

CloudWatch 指标`BufferCacheHitRatio`。如果该指标经常降至99.9%以下，请考虑尝试使用具有更多内存的实例，以确定它能否降低延迟和 I/O 成本。

只读副本的大小不必与您的写入器实例的大小相同。但是，繁重的写入工作负载可能会导致较小的副本落后并重启，因为它们无法跟上复制的速度。因此，我们建议创建等于或大于写入器实例的副本。

对只读副本使用自动扩缩时，请记住，将新的只读副本上线可能需要长达 15 分钟的时间。当客户端流量快速但可预测地增加时，考虑使用计划扩展来设置更高的只读副本的最小数量，以考虑该初始化时间。

无服务器实例支持多种不同的使用案例和工作负载。在以下情况下，考虑使用无服务器而不是预调配实例：

- 您一天中的工作负载经常波动。
- 您创建了一个新应用程序，但不确定工作负载的大小。
- 您正在进行开发和测试。

值得注意的是，按每 GB RAM 的美元计算，无服务器实例比同等预调配实例更昂贵。每个无服务器实例由 2GB 的 RAM 以及关联的 vCPU 和网络组成。在您的选项之间进行成本分析，以避免意外账单。通常，只有当您的工作负载每天只有几小时非常繁重，而其余时间几乎为零，或者当您的工作负载在一天中波动很大时，使用无服务器才能节省成本。

利用 Amazon Neptune 定价计算器，根据 queries-per-second (QPS) 要求等因素，帮助评估集群的正确配置。

优化写入

要优化写入情况，请考虑以下事项：

- Neptune 批量加载程序是初始加载数据库或附加到现有数据的最佳方式。Neptune 加载程序不是事务性的，也无法删除数据，因此如果您有这些要求，请不要使用它。
- 可以使用支持的查询语言进行事务更新。要优化写入 I/O 操作，请在每次提交时以 50-100 个对象为单位批量写入数据。对象是 LPG 中节点或边缘上的节点、边缘或属性，或者是 RDF 中的三元组存储或四元组。
- 每个连接的所有事务性 Neptune 写入操作都是单线程的。向 Neptune 发送大量数据时，考虑建立多个并行连接，每个连接都写入数据。当您选择 Neptune 预配置的实例时，实例大小与数字 v 相关联。CPUs Neptune 为实例上的每个 vCPU 创建两个数据库线程，因此在测试最佳并行化CPUs 时，从 v 数的两倍开始。 无服务器实例以大约每 4 NCUs 个 1 的速率缩放 v CPUs 的数量。

> **Note**
>
> 这不适用于批量加载 API，仅适用于直接连接。

- 即使任何时候只有一个连接ConcurrentModificationExceptions在写入数据，也要做好所有写入过程的规划和高效处理。设计客户端时，要确保在发生 `ConcurrentModificationExceptions` 时也能可靠运行。
- 如果您想删除所有数据，请考虑使用快速重置 API，而不是发出并发删除查询。与前者相比，后者将花费更长的时间并产生可观 I/O 的成本。
- 如果您要删除大部分数据，请考虑使用 neptune-export 将数据加载到新集群来导出要保留的数据。然后删除原始集群。

成本优化支柱

Well-Architecte AWS d Framework 的成本优化支柱侧重于避免不必要的成本。以下建议可以帮助您满足 Amazon Neptune 的成本优化设计原则和架构最佳实践。

成本优化支柱侧重于以下关键领域：

- 了解一段时间内的支出并控制资金分配
- 选择类型和数量正确的资源
- 在不超支的情况下进行扩展以满足业务需求

了解使用模式和所需的服务

如果您的数据模型具有清晰的图形结构，且您的查询需要探索关系并遍历多个跃点，则 Neptune 非常适合您的工作负载。图形数据库不适合以下模式：

- 主要是单跳查询（考虑您的数据是否可以更好地表示为对象的属性）
- JSON 或 BLOB 数据存储为属性
- 对数据集进行聚合的查询，例如计算大量节点的数值属性之和

考虑将多个专用数据库一起用于特定的访问模式是否可以满足您的所有需求。例如：

- 对于需要不那么频繁的复杂图形导航以及对单个节点属性进行高度并发检索的 API，最好使用一个或多个 Neptune、DynamoDB 或 Amazon DocumentDB 来呈现。
- 关系数据库可以与 Neptune 共存以维持您现有的功能，但只能将 Neptune 用于在关系数据库中性能不佳和扩展性不佳的多跳遍历。

了解与 Neptune 交互和补充 Neptune 的服务关联的成本，包括：

- Amazon Simple Storage Service（Amazon S3）存储成本，用于将数据文件批量加载到 Neptune 中
- 用于插入或更新插入查询、读取查询和 Neptune 流处理的 Lambda 函数
- 基于 Neptune 的 API 层用于与 Amazon API Gateway 中的客户端应用程序交互（而不是直接连接到数据库）或 AWS AppSync
- AWS Glue 用于在 Neptune 之间传输数据和从海王星传输数据的作业

- Amazon Kinesis 或 Amazon Managed Streaming for Apache Kafka（Amazon MSK）实例接收流数据，以近乎实时地将数据摄取到 Neptune。
- AWS Database Migration Service 用于将关系数据迁移到 Neptune
- Jupyter 笔记本和深度图库机器学习模型的 Amazon SageMaker Runtime 成本

选择资源时要注意成本

Neptune 定价基于每小时实例成本（或无服务器消耗的 Neptune 计算单元）、数据 I/O 和存储使用量。平均而言，实例占总成本的 85%，因此优化规模可能会带来显著的成本影响。调整实例大小的最佳方法是在各种实例上测试应用程序性能并比较以下因素：

- 该`MainRequestQueuePendingRequests` CloudWatch 指标是否一直保持在接近零的低水平？
- 该`BufferCacheHitRatio` CloudWatch 指标是否在大多数时间都保持在或高于 99.9%？
- 实例成本和相关数据 I/O 成本的成本和性能曲线是什么？ 如果实例容量过小，需要频繁地将缓冲区缓存与存储交换，则数据读取成本可能会显著增加。在这些情况下，`BufferCacheHitRatio` 将会频繁下降。

实例成本随相同实例系列内的大小线性扩展。`db.r6i.2xlarge` 实例的每小时成本是 `db.r6i.xlarge` 实例的两倍，资源分配也是后者的两倍。`db.r6i.24xlarge` 实例的每小时成本是 `db.r6i.xlarge` 实例每小时成本的 24 倍。

估算您必须支持的并发查询数量。您可以拥有 0 到 15 个只读副本来处理只读查询。如果您的要求因一天、一周或一个月的时间而异，则可以使用多个较小的实例按计划进行扩展。实例上的每个 vCPU 都提供两个用于处理并发查询的线程。三个 `db.r6i.xlarge` 只读副本（每个副本有 4 个 vCPU）可以处理 24 个并发查询。

如果您的流量改为以每秒查询数（QPS）来衡量，则必须进行实验以确定查询的平均延迟。Neptune 集群每秒可以支持的查询数等于 `vCPU × 2 × (1 second/average query latency)`。例如，如果您有 4 个 vCPU，查询延迟为 100 毫秒（0.1 秒），则 `QPS = 4 × 2 × (1s/0.1s) = 80 queries per second`。

对于持续、稳定和可预测的工作负载，预调配实例比无服务器实例更便宜。当您的工作负载每天只需几小时即可达到非常高的使用率（例如，`db.r6i.4xlarge`），而一天中的其余时间几乎没有流量（例如，1 个 Neptune 计算单位）时，无服务器可提供优化成本的机会。使用一个可纵向扩展几小时然后再缩减的无服务器实例，比全天使用预调配 `db.r6i.4xlarge` 实例更便宜。

考虑升级到 Neptune 1.4.5.0 或更高版本，并利用r8g实例以比老一代实例（例如或）更低的成本实现更好的读取和写入吞吐量。r7g r6g有关更多信息，请参阅使用 Amazon Neptune v1.4.5 的 G AWS raviton4 r8g 实例的写入查询性价比提高 4.7 倍（博客文章）。AWS

默认情况下，Neptune 集群是使用标准存储创建的（如果您使用控制台创建，则默认选择I/O-optimized storage). With I/O-optimized storage, you pay a slightly higher cost for storage and instances, but there are no I/O costs. This leads to more predictable recurring costs, but if your I/O usage is generally low, it may be more cost efficient to utilize standard storage. If you intend to load a lot of data initially, you can optimize cost by choosing I/O经过优化的存储，执行初始数据加载，然后切换到标准存储。存储类型仅影响计费模式，在 Neptune 数据库集群或实例配置中没有技术差异。您可以每 30 天更改一次存储类型。30 天后，查看您的 Neptune 详细费用，然后使用 Nept une 定价页面来计算使用优化后的费用是否会更高。I/O-optimized storage. If they would have been, continue to use standard storage, otherwise switch back to I/O

为您的工作负载选择最佳 Neptune 实例配置

如果您在 2025 年 7 月 15 日 AWS 账户 之前创建，则可以使用AWS 免费套餐进行 Neptune 的入门级实验。750 小时的 db.t3.medium 和 db.t4g.medium 实例免费使用时间足以让您很好地了解小规模下的 Neptune。在免费试用期结束后，您的集群仍将保留，但从那时起您将需要支付使用费用。

db.t3.medium和db.t4g.medium实例适用于不使用 OpenCypher、Graph Explorer 或各种生成式 AI 集成的低成本开发环境。这些实例的 RAM-to-vCPU比例 (2:1) 小于R系列实例 (8:1) 或X系列实例 (16:1)。这降低了比率，从而阻止了使用支持 OpenCypher 性能的 DFE 引擎统计信息、GenAI 集成（向 LLM 通报图形架构）和 Graph Explorer。使用T系列实例时，性能配置文件可能会有很大差异，特别是对于前面提到的工作负载。OutOfMemoryExceptions当查询在图表的很大一部分中导航时，这些实例还会增加出现的次数。要确定后一种情况是否可能受到影响，请检查BufferCacheHitRatio CloudWatch 指标。

我们强烈建议不要对T系列实例进行任何性能或负载测试，因为您可能会遇到不一致的结果，而这些结果并不代表生产环境。

当您的工作负载相当稳定且可预测时，预调配实例可为您提供最佳成本和性能组合。根据所需的请求并发性和查询复杂性选择实例大小。更高的并发性需要更多的 v CPUs。 查询复杂度越高，需要更多的 RAM。使用该MainRequestQueuePendingRequests CloudWatch 指标来确定前者的影响（大于零表示并发请求数多于可以处理的数量）。使用该BufferCacheHitRatio CloudWatch 指标来确定后者的影响。如果 RAM 使用率经常低于 99.9%，则表明 RAM 不足以容纳正在评估的图形的工作部分，从而导致更频繁地交换缓存。如果 R 系列实例提供了足够的并发性，但内存不足，请考虑尝试使用该X系列实例。

Neptune 文档中描述了无服务器实例的理想使用案例。如果您不确定预配置还是无服务器最适合您，并且成本是您的主要考虑因素，请在无服务器中测试您的工作负载以确定 NCUs 使用的数量，并将预配置 () 与无服务器 (`N hours × hourly provisioned cost`) 的成本进行比较。`sum of NCUs × hourly cost per NCU`如果您不确定同等大小的预调配实例，则一个 NCU 相当于大约 2GB 的 RAM 以及关联的 vCPU 和网络。如果您的预配置实例来自该`r6i`系列，则该比率为每 8 GB RAM 1 个 vCPU，或者 NCUs 4 个，再加上关联的网络。Amazon Neptune 定价计算器还提供比较，以帮助您确定最佳的成本配置。

在对主实例和副本实例使用无服务器时，请记住，升级层 0 和 1 中的只读副本将根据写 NCUs 入器实例进行扩展，以便在发生故障转移事件时可以正确扩展它们。根据您的哪个实例（写入器或读取器）接收的流量最多，为这些实例设置 NCU 限制。

在不需要集群全天候运行的环境中，可以考虑编写脚本，以便在不使用 Neptune 实例时将其关闭，并在需要使用之前重新启动它们。Neptune 实例将每 7 天自动重启一次，以确保应用所需的维护更新。如果您打算长时间关闭实例，请使用每周脚本再次将其关闭。

适当调整数据存储和传输的大小

更高效的查询（例如，需要触及图中较少节点、边和属性的查询）需要更少的 I/O 传输，并且由于需要的缓冲区缓存较少，因此有可能使用较小的实例。使用您的查询语言的 profile 或 explain 端点来优化查询，并考虑优化图表模型以提高查询性能。

Neptune 对大型字符串使用词典编码，该词典针对性能而非效率进行了优化。如果你的属性字符串很大 BLOBs、JSON 或者经常更改，可以考虑将它们存储在 Neptune 之外的亚马逊 S3、亚马逊 DynamoDB 或 Amazon DocumentDB 中，并且只在 Neptune 节点中存储引用。

在某些情况下，选择更大的实例大小可能更便宜。如果由于较低而导致 I/O 成本非常高`BufferCacheHitRatio`，则较大的缓冲区缓存可能会显著降低该成本。这是因为所有数据都将存放在缓存中，而不是经常从存储中交换并产生 I/O 传输速率。

Neptune 使用克隆 copy-on-write。在克隆以将图表拆分为多个分片时，不删除克隆集群上不需要的数据可能更有效，因为这将涉及创建新的数据页面，从而导致存储成本增加。克隆事件发生前未发生变化的数据将存在于两个集群之间共享的单个数据页面中，且仅对该单个副本收费。

请勿启用 OSGP 索引或使用 R5d 实例，除非您已测试确认它们对您的工作负载有重大影响。两者都是为极少发生的情况而设计的，它们可能会增加您的成本，而收益微乎其微或根本没有收益。

可持续发展支柱

可持续发展支柱侧重于最大限度地减少运行云工作负载对环境的影响。关键主题包括可持续发展责任共担模式、了解影响以及最大限度地使用以尽量减少所需资源和减轻下游影响。

可持续性支柱包含以下关键重点领域：

- 您的影响
- 可持续性目标
- 最大程度提高使用率
- 预测和采用更高效的新硬件和软件产品
- 使用托管服务
- 降低下游影响

本指南重点关注您的影响。有关其他可持续发展设计原则的更多信息，请参阅 Well-Architecte AWS d Framew ork。

您的选择和要求会对环境产生影响。如果您可以选择碳排放强度较低的 AWS 区域 ，且如果您的要求反映实际工作负载需求，而不仅仅是最大限度地提高正常运行时间和持久性，那么工作负载的可持续性就会提高。接下来的章节将讨论最佳实践和深思熟虑的注意事项，如果在您的工作负载设计和持续运营中采用这些实践和注意事项，将会对环境产生积极的影响。

AWS 区域 选择

AWS 区域 有些位于亚马逊可再生能源项目附近，或者位于电网公布的碳强度低于其他项目的地方。考虑可能适合您的工作负载的区域的可持续性影响，并将您的列表与 Neptune 可用的区域进行交叉引用。

基于用户行为模式的消费

根据用户的流量和行为调整资源消耗，有助于 AWS 最大限度地减少服务对环境的影响。设计解决方案时，考虑以下最佳实践：

- 监控CPUUtilization、MainRequestQueuePendingRequests、和之类的 Amazon CloudWatch 指标，TotalRequestsPerSec以确定您的需求何时为最高和最低，并确保在这段时间内您的集群资源大小合适。

- 在不使用非生产环境的时间段内，自动停止这些环境。有关更多信息，请参阅博客文章 Automate the stopping and starting of Amazon Neptune environment resources using resource tags。
- 如果您的流量模式变化频繁且不可预测，请考虑使用可随需求纵向扩展和缩减的 Neptune Serverless 实例，而不是使用为峰值流量预调配的实例。
- 除了业务连续性目标外，还可以考虑使您的服务水平协议与可持续性目标保持一致。放宽多区域灾难恢复、高可用性或长期备份保留等要求（尤其是对于非生产环境或非任务关键型工作负载）可以减少实现这些目标所需的资源量。

优化软件开发和架构模式

为防止浪费，请优化模型和查询，并共享计算资源，以便使用 Neptune 实例和集群中的所有可用资源。具体最佳实践包括：

- 让开发者共享 Neptune 实例和 Jupyter Notebook 应用程序实例，而不是各自创建自己的实例。通过使用多租户分区策略，在单个 Neptune 集群中为每位开发者提供自己的逻辑分区，并在单个 Jupyter 实例上为每位开发者创建单独的笔记本文件夹。
- 实现可最大限度利用资源并最大限度减少空闲时间的模式，例如使用并行线程加载数据，并将记录一起批处理到更大的事务中。
- 优化您的查询和图形模型，以最大限度地减少计算结果所需的资源。
- 对于 Gremlin 查询结果，请使用结果缓存功能最大限度地减少重新计算分页或经常重复的查询所花费的资源。
- 确保您的 Neptune 环境保持最新状态。最新版本的 Neptune 支持效率更高的最新亚马逊 EC2 实例，例如 Graviton。它们还改进了查询优化并修复了一些错误，从而减少了计算查询所需的资源量。

资源

参考

- AWS Well-Architected
- AWS WellArchited Framework 文档
- Neptune 最新更新
- 最佳实践：充分利用 Neptune
- 亚马逊 Neptune 定价计算器

博客文章

- 使用 Apache Gremlin 自动测试亚马逊 Neptune 数据访问权限 TinkerPop
- 使用资源标签自动停止和启动 Amazon Neptune 环境资源
- Amazon Neptune 数据面板操作的精细访问控制
- 使用 Amazon Neptune v1.4.5 的 AWS Graviton4 r8g 实例的写入查询性价比提高 4.7 倍
- Orca Security 如何优化其亚马逊 Neptune 数据库性能
- 使用 Amazon Neptune 公共终端节点更快地构建图形应用程序
- 全新 Amazon Neptune 引擎版本为 OpenCypher 查询性能提供高达 9 倍的速度和 10 倍的吞吐量

免费的“ AWS 技能大师”课程

- Amazon Neptune 入门
- 在 Amazon Neptune 上构建应用程序
- Amazon Neptune 的数据建模

贡献者

本指南的贡献者包括：

- Brian O'Keefe，Neptune 解决方案首席架构师 AWS
- Abhishek Mishra，Neptune 解决方案高级架构师， AWS
- Ganesh Sawhney，团队负责人——战略合作伙伴成功解决方案架构师， AWS
- Michael Havey，Neptune 解决方案高级架构师， AWS
- 凯文·菲利普斯，Neptune 解决方案架构师， AWS
- Melissa Kwok，Neptune 解决方案架构师， AWS
- Sakti Mishra，首席解决方案架构师 AWS
- Javed Ali，高级解决方案架构师， AWS

文档历史记录

下表介绍了本指南的一些重要更改。如果您希望收到有关未来更新的通知，可以订阅 RSS 源。

变更	说明	日期
Neptune 版本更新	我们更新了文档，加入了有关 Amazon Neptune 1.4.6.0 及更高版本的信息。	2026年1月2日
初次发布	—	2023 年 9 月 27 日

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。