图数据库 KonisGraph

快速入门

腾讯云

文档目录

快速入门
入门概述

最近更新时间：2023-10-11 10:23:11

本文旨在介绍如何快速使用图数据库 KonisGraph ，帮助用户快速了解图数据库 KonisGraph 使用的全流程，从实例的创建到基本使用，您需要完成如下操作。

1. 创建 KonisGraph 实例

通过图数据库 KonisGraph 控制台，您可以创建不同配置的 KonisGraph 实例，请参见 创建 KonisGraph 实例 。

2. 初始化 KonisGraph 实例

创建 KonisGraph 实例后，您还需要进行 KonisGraph 实例的初始化，如设置账号密码等，以轻松启用实例，请参见 初始化 KonisGraph 实例 。

3. 连接 KonisGraph 实例

初始化 KonisGraph 实例后，您可以通过多种方式连接 KonisGraph 实例，连接后可进行各种数据库的管理操作，请参见 连接 KonisGraph 实例 。

创建 KonisGraph 实例

最近更新时间：2024-01-30 16:59:11

本文为您介绍如何通过控制台创建 KonisGraph 实例。

> **说明：**
> 图数据库 KonisGraph 目前处于内测阶段，如需使用，请单击 立即申请 进行申请。

> **注意：**
> 图数据库免费内测实例，暂不具备数据备份功能，不属于服务高可用的保障责任。

前提条件

已注册腾讯云账号并完成实名认证，且图数据库 KonisGraph 内测申请成功。

- 如需注册腾讯云账号： 点此注册腾讯云账号 。
- 如需完成实名认证： 点此完成实名认证 。
- 如需内测申请： 点此内测申请 。

操作步骤

1. 登录 KonisGraph 控制台 ，单击**新建实例**，在购买页根据实际需求选择各项配置信息，确认无误后，单击**立即购买**，即可在内测期间免费试用。

   > **说明：**
   > **架构**：架构介绍请参见 产品架构 。

   - **计费模式**：支持包年包月。
   - **地域**：选择您业务需要部署 KonisGraph 的地域。
   - **可用区**：选择您业务需要部署 KonisGraph 的可用区。
   - **计算节点规格**：选择您业务合适的计算节点规格。
   - **计算节点数量**：选择您业务合适的计算节点数量。
   - **数据存储类型**：图数据存储的云硬盘类型。
   - **数据副本容量**：图数据库 KonisGraph 中单份数据副本的存储容量规格。
   - **数据副本**：3副本。
   - **网络**：图数据库 KonisGraph 所属网络，建议您选择与云服务器同一个地域下的同一 私有网络 。
   - **实例名**：实例名支持立即设置和创建完后在实例列表进行设置。
   - **账号名/密码**：账号名和密码请按照相应的命名规则进行设置。

2. 返回控制台实例列表，会看到实例显示**创建中**，请耐心等待，待实例状态变为**运行中**，即可进行操作。

后续操作

通过控制台初始化 KonisGraph 实例，请参见 初始化 KonisGraph 实例 。

初始化 KonisGraph 实例

最近更新时间：2023-10-11 10:23:12

创建 KonisGraph 实例后，您还需要进行 KonisGraph 实例的初始化，以轻松安全启用实例。

操作步骤

1. 登录 KonisGraph 控制台，选择对应地域后，在实例列表，选择刚创建的实例，在**操作**列单击**管理**。
2. 在**账号管理**页面，重置默认账号的密码，请参见 重置密码 。

3. 返回实例列表，实例状态为**运行中**，即可正常使用。

后续操作

通过云服务器连接图数据库 KonisGraph，请参见 连接 KonisGraph 实例 。

连接 KonisGraph 实例

最近更新时间：2023-10-11 10:23:12

本文为您介绍创建初始化实例后，通过内网连接 KonisGraph 实例。

准备工作

- 准备已经初始化的 KonisGraph 实例。
- 准备好数据库 KonisGraph 实例的账号和密码。
- 配置云服务器 CVM 安全组出入站规则，允许访问 KonisGraph 的 IP，请参见 安全组 。

连接方式

连接图数据库 KonisGraph 的方式有3种：

- 图数据管理平台，参考 图数据管理 。
- 云服务器 CVM 连接，云服务器 CVM 连接图数据库 KonisGraph 步骤依次为：gremlin-console 连接、图的元数据操作、图数据的读写。
  连接的注意事项如下：
  内网地址连接：通过内网地址连接图数据库 KonisGraph，使用云服务器 CVM 直接连接云数据库的内网地址，这种连接方式使用内网高速网络，延迟低。
  - 云服务器和数据库须是同一账号，且同一个 VPC 内（保障同一个地域）。
  - 内网地址系统默认提供，可在 KonisGraph 控制台 的实例列表或实例详情页查看。

  > **说明**
  > 对于不同的 VPC 下（包括同账号/不同账号，同地域/不同地域）的云服务器和数据库，内网连接方式请参见 云联网 。

- SDK 连接和操作，参考 SDK 操作参考 。

图数据管理平台连接

通过图数据管理平台连接管理 KonisGraph 实例，请参考 图数据管理 。

云服务器连接

下面示例分别如何从云服务器内网连接图数据库 KonisGraph。

步骤1：gremlin-console 连接

1. 登录 Linux 云服务器 ，下载 apache-tinkerpop-gremlin-console-3.5.1-bin 。
2. 解压到任意目录，默认解压目录 `apache-tinkerpop-gremlin-console-3.5.1` 。

3. 进入解压目录，创建连接配置文件 `remote-secure-test.yaml` 放到 `conf` 目录下。文件内容如下：

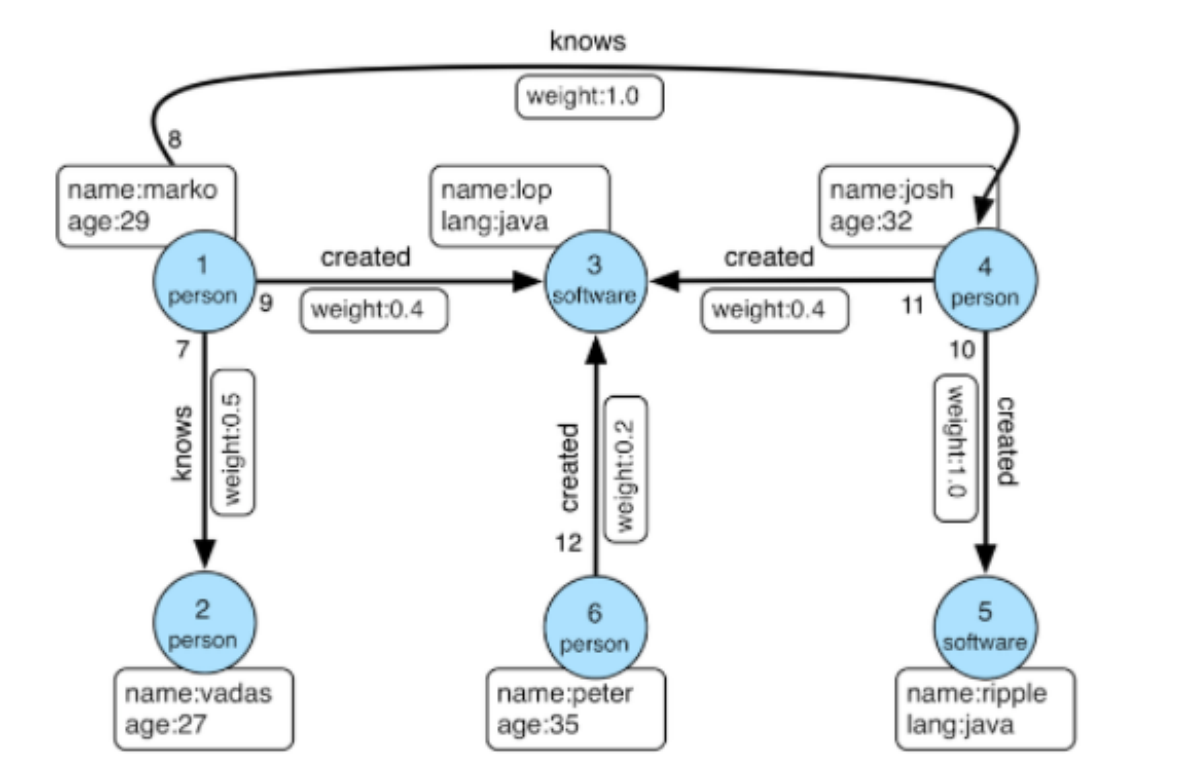
```
# hosts 图数据库 KonisGraph 实例的内网地址 vip，如 10.xx.xx.107
hosts: [10.xx.xx.107]
# port 图数据库 KonisGraph 实例的 Gremlin 端口，如 8186
port: 8186
# username/password 图数据库 KonisGraph 实例的账号和密码，如账号：steven，密
码：test-pwd-123
username: steven
password: test-pwd-123
connectionPool: {
  enableSsl: false,
  sslEnabledProtocols: [TLSv1.2] }
# serializer: { className:
org.apache.tinkerpop.gremlin.driver.ser.GraphBinaryMessageSerializerV1, config: {
serializeResultToString: true }}
serializer: { className:
org.apache.tinkerpop.gremlin.driver.ser.GraphSONMessageSerializerV3d0, config: {
serializeResultToString: true, useMapperFromGraph: graph }}
```

4. 使用 `bin/gremlin.sh` 命令开始连接：

```
      \,,,/
      (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> :remote connect tinkerpop.server conf/remote-secure-test.yaml
==>Configured 10.221.176.107/10.221.176.107:8186
gremlin> :remote console
==>All scripts will now be sent to Gremlin Server -
[10.221.176.107/10.221.176.107:8186] - type ':remote console' to return to local
mode
```

步骤2：图的元数据操作

以 Gremlin-console tutorials 中的人和软件的关系图为示例。



如图所示，整个图包含2类点 person 和 software ，2类边 knows 和 created ，和几类属性 id 、name 、age 、 lang 、 weight。

创建属性

```
gremlin> s.addP('id', "T_LONG", "0")
==>id: 11
propertyName: "id"
dataType: T_LONG
defaultValue: "0"

gremlin> s.addP('name', "T_STRING", "")
==>id: 1
propertyName: "name"
dataType: T_STRING

gremlin> s.addP("age", "T_LONG", "0")
==>id: 2
propertyName: "age"
dataType: T_LONG
defaultValue: "0"

gremlin> s.addP("lang", "T_STRING", "")
==>id: 3
```

```
propertyName: "lang"
dataType: T_STRING

gremlin> s.addP("weight", "T_DOUBLE", "0.0")
==>id: 4
propertyName: "weight"
dataType: T_DOUBLE
defaultValue: "0.0"
```

创建点

```
gremlin> s.addV("person", "id", ["name", "age"])
==>id: 7
name: "person"
primary_key: "id"
create_time: 1627894586026
properties: 1
properties: 2

gremlin> s.addV("software", "id", ["name", "lang"])
==>id: 8
name: "software"
primary_key: "id"
create_time: 1627894607209
properties: 1
properties: 3

gremlin>
```

创建边

```
gremlin> s.addE("knows", "person", "person", ["weight"])
==>id: 9
name: "knows"
src_label_id: 7
dst_label_id: 7
create_time: 1627894691660
properties: 4

gremlin> s.addE("created", "person", "software", ["weight"])
==>id: 10
name: "created"
src_label_id: 7
dst_label_id: 8
```

```
create_time: 1627894710659
properties: 4

gremlin>
```

步骤3：图数据的读写

写入点数据 person

```
gremlin> g.addV("person").property("id", 1).property("name",
"marko").property("age", 29)
==>v[7.1]
gremlin> g.addV("person").property("id", 2).property("name",
"vadas").property("age", 27)
==>v[7.2]
gremlin> g.addV("person").property("id", 4).property("name", "josh").property("age",
32)
==>v[7.4]
gremlin> g.addV("person").property("id", 6).property("name",
"peter").property("age", 35)
==>v[7.6]
gremlin>
```

写入点数据 software

```
gremlin> g.addV("software").property("id", 3).property("name",
"lop").property("lang", "java")
==>v[8.3]
gremlin> g.addV("software").property("id", 5).property("name",
"ripple").property("lang", "java")
==>v[8.5]
gremlin>
```

写入边数据 knows

```
gremlin> g.addE("knows").from(g.V().has("name", "marko")).to(g.V().has("name",
"vadas"))
==>e[9.1.2][7.1-knows->7.2]
gremlin> g.addE("knows").from(g.V().has("name", "marko")).to(g.V().has("name",
"josh"))
==>e[9.1.4][7.1-knows->7.4]
gremlin>
```

写入边数据 created

```
 g.addE("created").from(g.V().has("name", "marko")).to(g.V().has("name",
"lop")).property("weight", "0.4")
==>e[10.1.3][7.1-created->8.3]
gremlin> g.addE("created").from(g.V().has("name", "peter")).to(g.V().has("name",
"lop")).property("weight", "0.2")
==>e[10.6.3][7.6-created->8.3]
gremlin> g.addE("created").from(g.V().has("name", "josh")).to(g.V().has("name",
"lop")).property("weight", "0.4")
==>e[10.4.3][7.4-created->8.3]
gremlin> g.addE("created").from(g.V().has("name", "josh")).to(g.V().has("name",
"ripple")).property("weight", "1.0")
==>e[10.4.5][7.4-created->8.5]
gremlin>
```

批量写入点边数据

图数据库 KonisGraph 推荐使用实时批量写入点、边数据，相比于以上的逐个写入方式，批量写入点、边数据可以降低网络开销，从而大幅度提高图数据库的实时写入速度。
示例如下：

```
gremlin> g.addV("person").property("id", 1).property("name",
"marko").property("age", 29).addV("person").property("id", 2).property("name",
"vadas").property("age", 27).addV("person").property("id", 4).property("name",
"josh").property("age", 32).addV("person").property("id", 6).property("name",
"peter").property("age", 35)

gremlin> g.addE("created").from(g.V().has("name", "marko")).to(g.V().has("name",
"lop")).property("weight", "0.4").addE("created").from(g.V().has("name",
"peter")).to(g.V().has("name", "lop")).property("weight",
"0.2").addE("created").from(g.V().has("name", "josh")).to(g.V().has("name",
"lop")).property("weight", "0.4").addE("created").from(g.V().has("name",
"josh")).to(g.V().has("name", "ripple")).property("weight", "1.0")
```

查询点边和属性数据

```
# 查找所有的边
gremlin> g.E()
# 查找所有的点
gremlin> g.V()
==>v[7.1]
==>v[7.4]
```

```
==>v[7.2]
==>v[8.5]
==>v[7.6]
==>v[8.3]
gremlin> g.V(1).values('name')
==>marko
# 按属性 name=marko 过滤所有点
gremlin> g.V().has("name","marko")
# marko 认识的人
gremlin> g.V(1).outE('knows').inV().values('name')
==>josh
==>vadas
# marko 认识的人里谁的年龄超过了30岁
gremlin> g.V(1).out('knows').has('age', gt(30)).values('name')
==>josh
# 哪些人创建了软件 lop
gremlin> g.V(3).in('created').values('name')
==>peter
==>marko
==>josh

# 查询点4所有入边的起点
gremlin> g.V(4).inE().outV()
# 查询点4所有出边的终点并展示路径
gremlin> g.V(4).outE().inV().path().unfold()
# 查询点4的所有一度关系入边的起点
gremlin> g.V(4).repeat(inE().outV()).emit().times(1)
# 查询点4的所有入边二度关系的起点并展示路径
gremlin> g.V(4).repeat(inE().outV()).emit().times(2).path().unfold().dedup()
# select查询，展示所有边及起点终点并用别名展示，使用时尽量多加 has 过滤
gremlin> g.E().as("edge").bothV().as("vertex").select("edge", "vertex")
# repeat使用，查询点1的入边去重，直到入边为0时停止，并展示所有路径
gremlin>
g.V(1).repeat(inE().outV().dedup()).until(inE().count().is(0)).emit().path().unfold().dedu
p()
# 数值过滤,查询年龄大于18小于60的所有 person
gremlin>
g.V().hasLabel('person').has('age',gt(18)).has('age',lt(60)).limit(10).valueMap();
```

SDK 连接和操作

图数据库 KonisGraph 支持 TinkerPop Gremlin 查询语言。用户可以使用 Go、Python 等不同语言的 SDK 通过 Gremlin 来操作图数据库，请参考 SDK 操作参考 。