

# HashData

# 目录

|                                     |     |
|-------------------------------------|-----|
| 1 入门指南                              | 1.1 |
| 2 步骤1: 准备工作                         | 2.1 |
| 3 步骤2: 启动 HashData 数据仓库样例集群         | 3.1 |
| 4 步骤3: 授权连接样例集群                     | 4.1 |
| 5 步骤4: 连接样例集群                       | 5.1 |
| 6 步骤5: 将样例数据从对象存储加载到 HashData 数据仓库中 | 6.1 |
| 7 步骤6: 寻找额外的资料和重设你的环境               | 7.1 |
| 8 JSON 函数和操作符                       | 8.1 |

# 入门指南

欢迎来到 HashData 数据仓库指南。HashData 数据仓库是一个高性能，完全托管的 PB 级数据仓库服务。一个 HashData 数据仓库是由一组称之为节点的计算资源组成的集群。每个集群作为一个 HashData 数据仓库引擎，包含一个或多个数据库。

这个指南的目的是指导你创建一个 HashData 数据仓库样例集群。你可以通过这个样例集群来测试 HashData 数据仓库的功能。在这个教程中，你将执行如下步骤：

- 步骤1：准备工作
- 步骤2：启动 HashData 数据仓库样例集群
- 步骤3：授权连接样例集群
- 步骤4：连接样例集群
- 步骤5：将样例数据从对象存储加载到 HashData 数据仓库中
- 步骤6：寻找额外的资料和重设你的环境

这个教程的目的不是为了配置生产环境的，所以不会深入地讨论操作中的各种选项。当你完成了这个教程的所有步骤后，你能通过“额外的资料”章节找到关于集群计划、部署和维护，以及如何操作数据仓库中数据的更深入的信息。

# 步骤1: 准备工作

在你创建第一个 HashData 数据仓库集群前，确保在这个章节中完成如下准备工作：

- 注册青云账号
- 安装 SQL 客户端工具
- 确认防火墙规则

## 注册青云账号

你首先需要注册一个青云账号。如果你已经拥有了一个青云账号，那么你可以跳过这个步骤，使用已经拥有的账号进行接下来的操作。

1. 打开链接 <http://qingcloud.com>，点击注册。
2. 按照页面提示完成注册流程。

## 安装 SQL 客户端工具

你可以使用任何 postgres 兼容的客户端程序连接到 HashData 数据仓库，比如 psql。此外，你还可以通过绝大部分使用标准数据库应用接口，如 JDBC，ODBC 的客户端程序连接到 HashData 数据仓库。最后，你还可以使用标准数据库应用接口开发自己的客户端程序来访问 HashData 数据仓库。由于 HashData 数据仓库基于 greenplum，而后者又是基于 postgres 而来，所以你可以直接使用 postgres 驱动访问 HashData 数据仓库。在这个教程中，我们将通过 psql 这个 postgres 的客户端程序演示如何连接到 HashData 数据仓库。

## 安装 psql

1. 如果你正在使用 Linux 操作系统，你可以使用以下命令安装 psql。

```
Redhat/Centos:  
# yum install postgresql  
  
Ubuntu:  
# apt-get install postgresql-client
```

2. 或者访问 [PostgreSQL 官方网站](#)，根据你的操作系统下载安装包。

## 确认网络配置

HashData 数据仓库使用 5432 作为服务的端口地址。当你希望从云的外部访问 HashData 数据仓库服务时，你不仅需要拥有一个公网 IP 地址，还需要指定一个绑定在公网 IP 地址上的端口，并将此端口转发到 HashData 数据仓库的服务端口 5432 上。我们在随后的章节中帮助你完成端口转发的工作，现在你只需要明确这个端口的地址即可。你可以随意指定这个端口，为了便于记忆，我们推荐你仍然使用 5432 作为公网端口地址。

此外，你需要在青云防火墙中添加一个下行规则从而允许外来连接通过这个端口访问你的数据仓库集群。

如果你的客户端程序运行在云的内部，那么你既不需要进行端口转发，也不需要配置任何防火墙规则。



## 步骤2: 启动 HashData 数据仓库样例集群

完成上面的前提步骤后，现在你能够开始启动一个 HashData 数据仓库集群。

### 启动 HashData 数据仓库集群

1. 登陆青云并在应用中心找到对应的产品 [数据仓库\(HashData高性能MPP数据仓库\)](#)。
2. 点击部署到控制台，你可以选择在哪个数据中心创建你的数据仓库集群。在这个教程中，我们选择了北京3区。

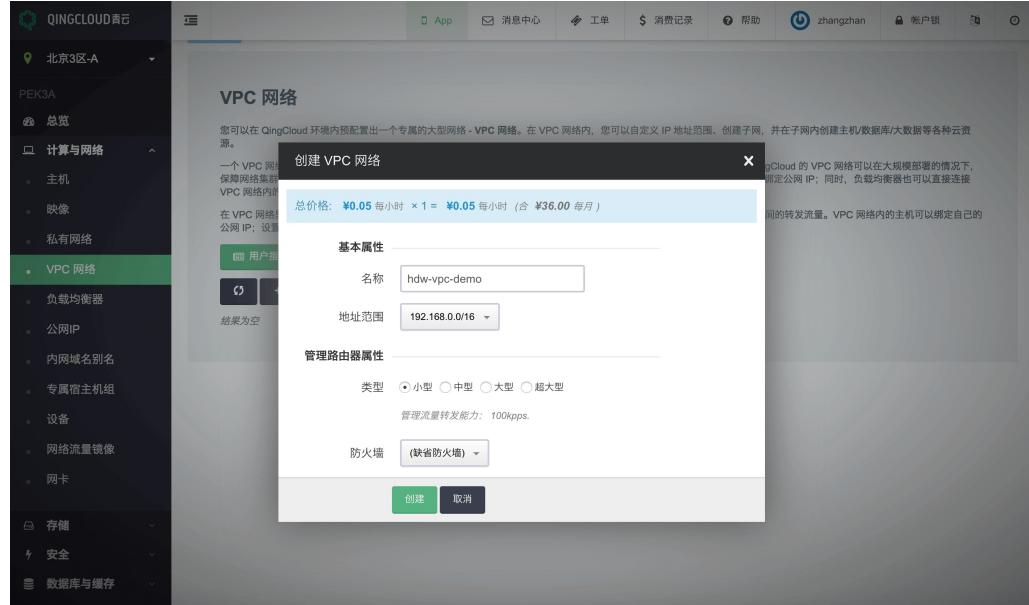
The screenshot shows the QINGCLOUD application center interface. On the left sidebar, under 'AppCenter', 'HashData' is selected. In the main area, a 'Create Application Instance' dialog is open for 'HashData Cluster'. The 'Basic Settings' tab is active, showing fields for 'Name' (HashData Cluster), 'Description' (HashData集群服务描述), 'Version' (v1.3.0), and 'Billing Method' (Hourly). To the right, a 'Cost Preview' panel displays resource details and a total price of 1.888 CNY/hour (合 1359.144 元/月).

3. 创建依赖资源：你需要有一个已连接到 VPC 的私有网络。如果您没有创建好依赖资源，点击创建后，可以按照提示完成下面的步骤：

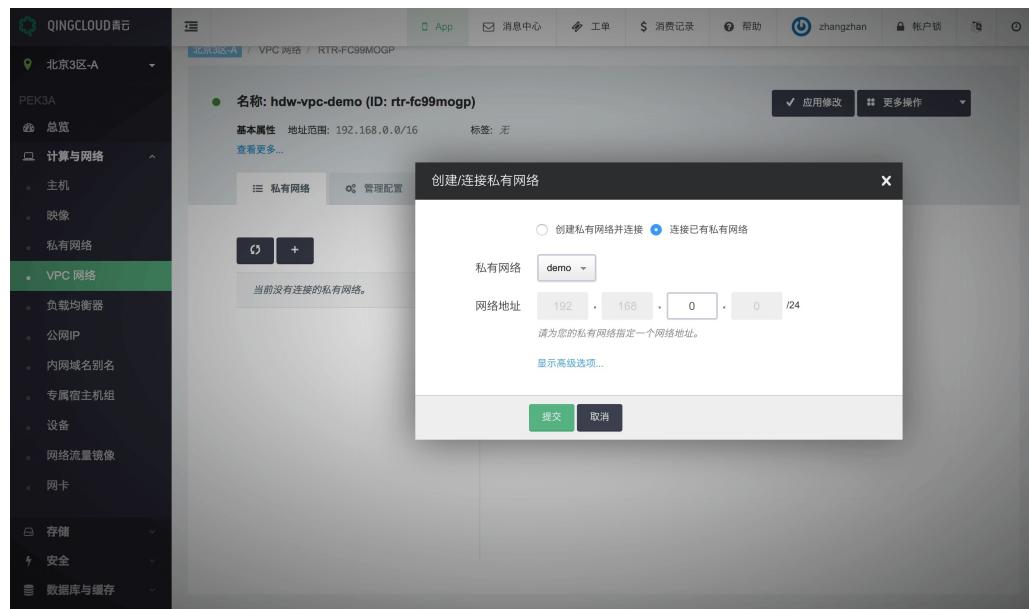
- 创建私有网络：计算机与网络 -> 私有网络，点击创建

The screenshot shows the QINGCLOUD control panel with 'Private Network' selected in the sidebar. A modal dialog titled 'Create Private Network' is open, prompting for a 'Name' (demo) and 'Quantity' (1). The 'Managed by' option is selected. The background shows a list of existing private networks.

- 创建 VPC 网络：计算机与网络 -> VPC网络，点击创建

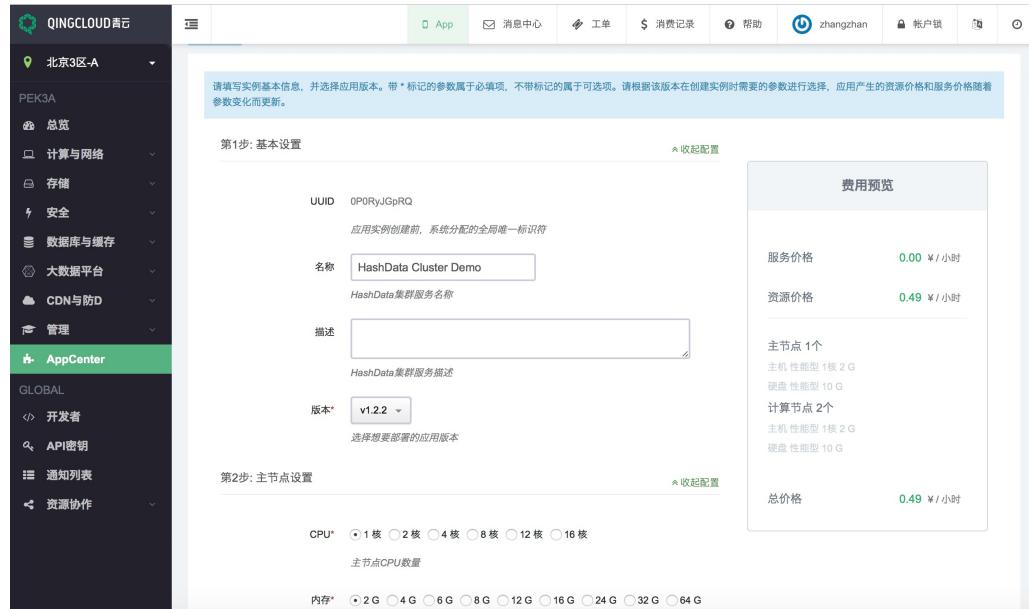


- 连接私有网络到 VPC 网络：计算机与网络 -> VPC网络，点击创建完成的VPC网络，将上面创建完成的私有网络添加到VPC中

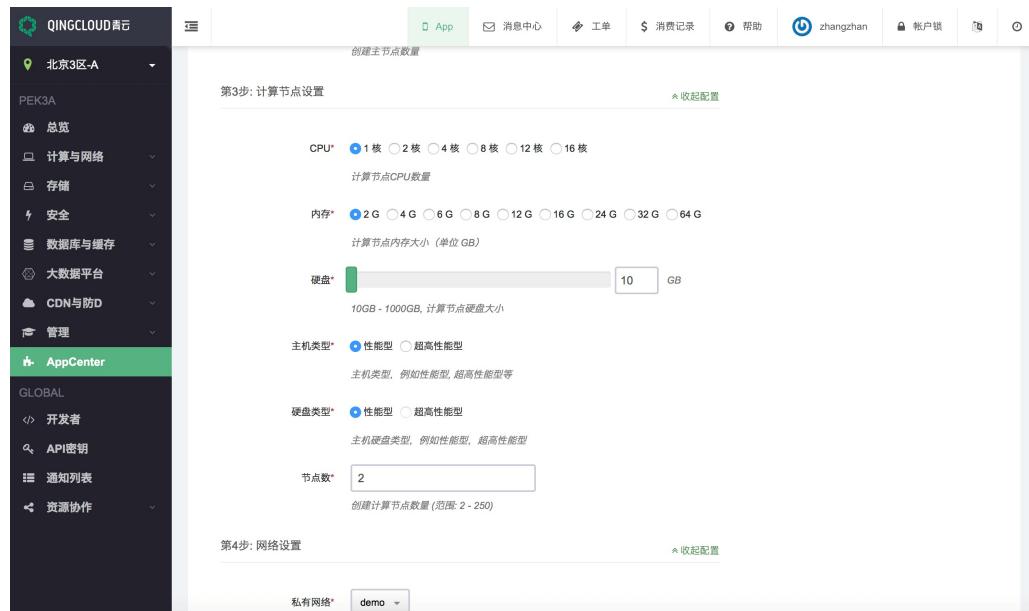


#### 4. 创建好私有网络后，就可以创建 HashData 数据仓库集群了：

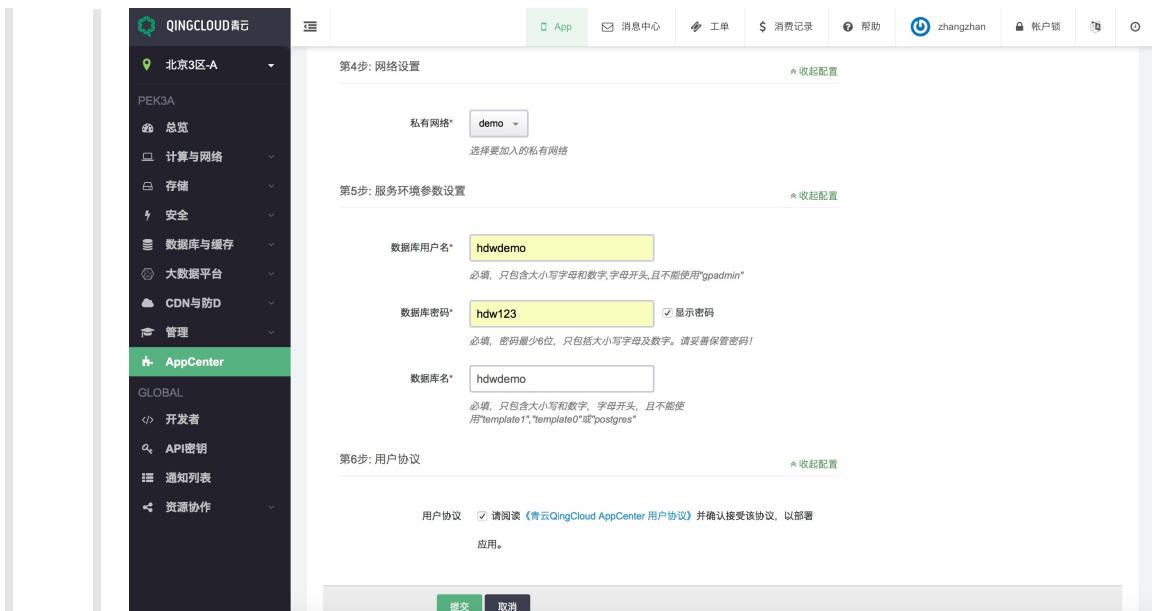
- 填写基本配置，选择软件版本
- 选择主节点的配置



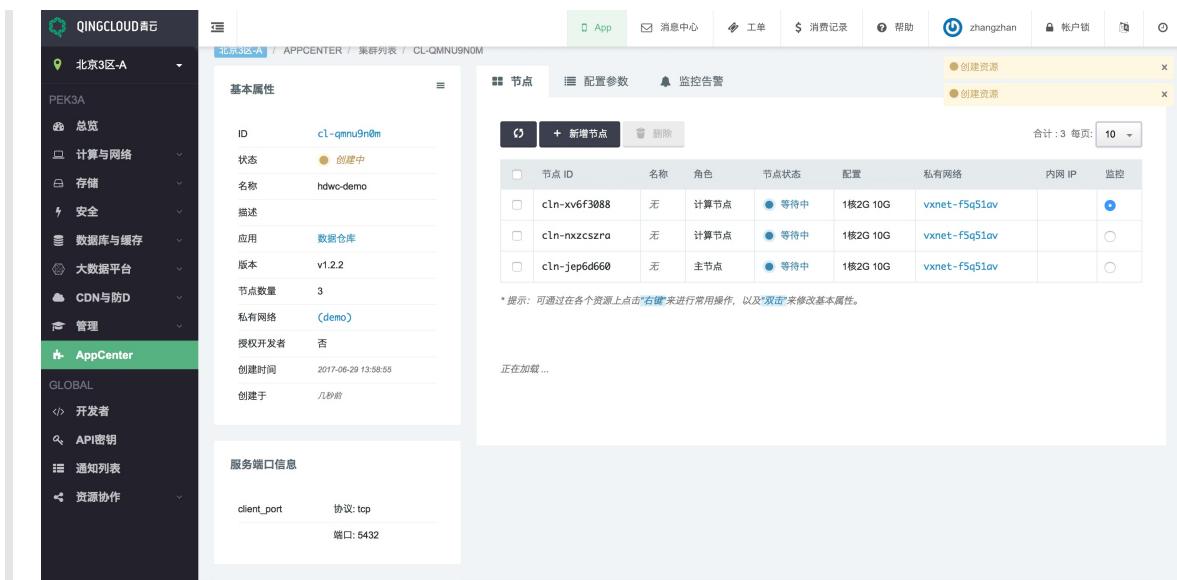
- 选择计算节点的配置和数量



- 选择之前创建的私有网络
- 设置数据库用户名，数据库密码以及初始数据库名字
- 阅读用户协议



5. 配置完以上参数，阅读并勾取用户协议，点击创建后，新的集群将会在几分钟之内创建完毕。下图是系统创建时的状态：



6. 在 AppCenter 控制面板中，选择新创建的集群并且查看集群状态信息。在你连接数据仓库之前，一定要确认集群的状态是可用的，并且数据库的健康状态是正常。

QINGCLOUD 青云

北京3区-A

PEK3A

总览

计算与网络

存储

安全

数据库与缓存

大数据平台

CDN与防DDoS

管理

AppCenter

GLOBAL

开发者

API密钥

通知列表

资源协作

北京3区-A / APPCENTER / 集群列表 / CL-QMNU9NOM

基本属性

ID: cl-qmnu9n0m  
状态: 活跃  
名称: hdwc-demo  
描述:  
应用: 数据仓库  
版本: v1.2.2  
节点数量: 3  
私有网络: (demo)  
授权开发者: 否  
创建时间: 2017-06-29 13:58:55  
创建于: 9分钟前

节点

+ 新增节点

合计: 3 每页: 10

| 节点 ID        | 名称 | 角色   | 节点状态 | 服务状态 | 配置       | 私有网络          | 内网 IP       | 监控 |
|--------------|----|------|------|------|----------|---------------|-------------|----|
| cln-xv6f3088 | 无  | 计算节点 | 活跃   | 获取中  | 1核2G 10G | vxnet-f5q51av | 192.168.0.3 |    |
| cln-nxzcszra | 无  | 计算节点 | 活跃   | 获取中  | 1核2G 10G | vxnet-f5q51av | 192.168.0.4 |    |
| cln-jep6d660 | 无  | 主节点  | 活跃   | 获取中  | 1核2G 10G | vxnet-f5q51av | 192.168.0.2 |    |

\* 提示: 可通过在各个资源上点击右键来进行常用操作, 以及双击来修改基本属性。

资源

最近6小时 最近一天 最近两周 最近一个月 最近6个月

CPU

单位: % 间隔: 5分钟 监控项: CPU

100  
90  
80  
70  
60  
50

服务端口信息

client\_port 协议: tcp  
端口: 5432

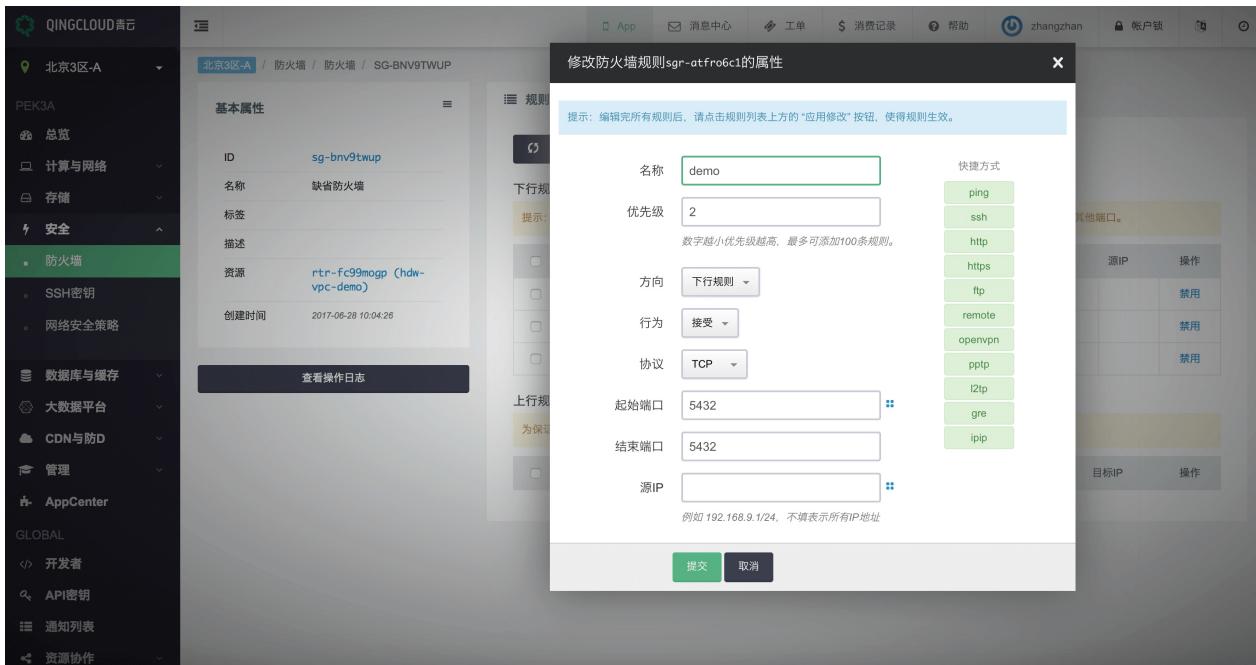
The screenshot shows the QINGCLOUD AppCenter interface for managing clusters. On the left is a sidebar with navigation links for various services like Compute & Network, Storage, Security, etc., and specific sections for AppCenter, GLOBAL, Developers, API Keys, Notifications, and Resource Collaboration. The main content area displays a cluster named 'CL-QMNU9NOM' with an ID of 'cl-qmnu9n0m'. It shows the cluster is active and has three nodes. The nodes are listed with their IDs, names, roles, states, service statuses, configurations, private networks, and internal IP addresses. A note at the bottom indicates that right-clicking on resources provides common operations and double-clicking allows modification of basic properties. Below the nodes is a resource monitoring section for CPU usage over the last 6 hours, showing a graph from 50% to 100%. At the bottom, it lists the service port information as client\_port:tcp with port 5432.

## 步骤3: 授权连接样例集群

在前面的步骤中，你已经创建启动了你的 HashData 数据仓库集群。在你连接到数据仓库集群之前，你需要对上一步骤中创建的路由器进行相应的配置。

## 防火墙配置

在路由器的详情页面，点击选用的防火墙进入其详情页面，添加一条打开 5432 端口的下行规则，如下所示：



这条下行规则允许你的 SQL 客户端工具能够访问路由器的 5432 端口。

## 路由器端口转发规则

回到路由器的详情页面，点击端口转发标签，添加一条 5432 端口的转发规则，如下所示：

The screenshot shows the QINGCLOUD console interface. On the left, there's a sidebar with categories like PEK3A, 总览 (Overview), 计算与网络 (Compute & Network), VPC 网络 (VPC Network), 公网IP (Public IP), etc. The 'VPC 网络' section is selected. In the main area, a VPC network named 'hdw-vpc-demo' (ID: rtr-fc99mogp) is displayed. A modal window titled '添加端口转发规则' (Add Port Forwarding Rule) is open. It contains fields for '名称' (Name: demo), '协议' (Protocol: TCP), '源端口' (Source Port: 5432), '内网 IP' (Internal IP: 192.168.0.2), and '内网端口' (Internal Port: 5432). A note at the bottom says: '如果私有网络中的主机绑定了公网 IP, 那么从 VPC' (If a host in the private network binds a public IP, then from VPC). At the bottom of the modal are '提交' (Submit) and '取消' (Cancel) buttons.

这条转发规则确保在你访问路由器 5432 端口时，请求转发到 HashData 数据仓库集群的主节点 5432 端口。

## 配置公网 IP

如果你的 SQL 客户端不在青云的网络里，你还需要申请一个公网 IP 地址，并绑定到前面步骤中创建的路由器。

The screenshot shows the QINGCLOUD console interface. The sidebar has sections like PEK3A, 总览 (Overview), 计算与网络 (Compute & Network), 公网IP (Public IP), etc. The '公网IP' section is selected. A modal window titled '申请公网IP' (Apply Public IP) is open. It shows a price of '¥0.13 每小时 × 1 = ¥0.130 每小时 (含 ¥93.600 每月)' (¥0.13 per hour × 1 = ¥0.130 per hour (including ¥93.600 per month)). The form fields include '名称' (Name: demo), '数量' (Quantity: 1), '计费模式' (Billing Mode: 按带宽计费 - Pay-by-bandwidth), '带宽上限' (Bandwidth Limit: 1 Mbps - 300 Mbps, set to 4 Mbps), 'IP 组' (IP Group: BGP多线 - BGP Multi-homing), and 'ICP备案' (ICP Registration: 不需要 (Not Required)). At the bottom are '提交' (Submit) and '取消' (Cancel) buttons.

将公网 IP 绑定到路由器：

QINGCLOUD 青云

北京3区-A / 公网IP

公网 IP (Elastic IP) 是在互联网上合法的静态 IP 地址。在 QingCloud 系统中，公网 IP 地址与您的账户而非特定的资源关联，您可以将申请到的公网 IP 地址与任意主机（包括私有网络内的主机）/路由器/负载均衡器绑定，并随时可以解绑、再分配到其他资源上。

选择要绑定公网 IP [eip-8j8hokj2] 的 VPC 网络

\* 提示：可通过在各个资源上点击 来选择。

| 带宽上限 (Mbps) | IP 组  | 告警状态 | 创建于   |
|-------------|-------|------|-------|
| 4           | BGP多线 | 无监控  | 2 分钟前 |

提交 取消

北京3区-A / 公网IP

PEK3A

总览

计算与网络

- 主机
- 映像
- 私有网络
- VPC 网络
- 负载均衡器

公网IP

- 内网域名别名
- 专属宿主组
- 设备
- 网络流量镜像
- 网卡

存储

安全

数据库与缓存

大数据平台

## 步骤4: 连接样例集群

现在你可以通过 SQL 客户端工具连接到你的数据仓库集群，并且跑一条简单的查询语句来测试连接。你能够使用几乎所有与 postgres 兼容的 SQL 客户端工具。在这个教程中，你将使用在准备工作中安装的 postgres 自带的 psql 客户端。

### 确定连接 IP 地址和端口

数据仓库集群的 IP 地址可由配置公网 IP 步骤中确定。在剩下的教程中，我们用 121.201.25.29 作为例子。在集群主控制台，选择 examplecluster 进入详情页面。从详情页面中，你能看到端口：5432。

### 使用 psql 连接到集群

你可以通过下面命令连接到集群：

```
psql -d postgres -h 121.201.25.29 -p 5432 -U admin
```

然后根据提示输入登陆密码。

### 简单测试查询

登陆数据仓库后，你可以运行如下命令做一些简单的测试查询：

```
postgres=# CREATE TABLE foo (a INT, b INT);
NOTICE: Table doesn't have 'DISTRIBUTED BY' clause -- Using column named 'a' as the Greenplum Database data distribution key for this table.
HINT: The 'DISTRIBUTED BY' clause determines the distribution of data. Make sure column(s) chosen are the optimal data distribution key to minimize skew.
CREATE TABLE
postgres=# INSERT INTO foo (SELECT i, i + 1 FROM generate_series(1, 10000) AS i);
INSERT 0 10000
postgres=# SELECT COUNT(*) FROM foo;

## count

10000
(1 row)

postgres=# SELECT SUM(a) FROM foo;

## sum

50005000
(1 row)
```

## 步骤5：将样例数据从对象存储加载到 HashData 数据仓库中

现在你已经有了一个名为 `postgres` 的数据库，并且你已经成功地连接上它了。接下来你可以在数据库中创建一些新表，然后加载数据到这些表中，并尝试一些查询语句。为了方便你的测试，我们准备了一些 TPC-H 的样例数据存储在青云对象存储中。

### 1. 创建表

拷贝并执行下面的建表语句在 `postgres` 数据库中创建相应的表对象。你可以通过 HashData 数据仓库 [开发指南](#) 查看更详细的建表语法。

其中定义的外部表（READABLE EXTERNAL TABLE）用来访问青云对象存储上面的数据。我们提供了 1GB、10GB、100GB 的 TPC-H 公共测试数据集，在此示例中我们使用 1GB 的 TPC-H 数据集。

```
CREATE TABLE NATION (
    N_NATIONKEY  INTEGER NOT NULL,
    N_NAME        CHAR(25) NOT NULL,
    N_REGIONKEY   INTEGER NOT NULL,
    N_COMMENT     VARCHAR(152));

CREATE TABLE REGION (
    R_REGIONKEY  INTEGER NOT NULL,
    R_NAME        CHAR(25) NOT NULL,
    R_COMMENT     VARCHAR(152));

CREATE TABLE PART (
    P_PARTKEY      INTEGER NOT NULL,
    P_NAME         VARCHAR(55) NOT NULL,
    P_MFGR          CHAR(25) NOT NULL,
    P_BRAND         CHAR(10) NOT NULL,
    P_TYPE          VARCHAR(25) NOT NULL,
    P_SIZE          INTEGER NOT NULL,
    P_CONTAINER     CHAR(10) NOT NULL,
    P_RETAILPRICE  DECIMAL(15,2) NOT NULL,
    P_COMMENT       VARCHAR(23) NOT NULL );

CREATE TABLE SUPPLIER (
    S_SUPPKEY      INTEGER NOT NULL,
    S_NAME         CHAR(25) NOT NULL,
    S_ADDRESS      VARCHAR(40) NOT NULL,
    S_NATIONKEY    INTEGER NOT NULL,
    S_PHONE        CHAR(15) NOT NULL,
    S_ACCTBAL     DECIMAL(15,2) NOT NULL,
    S_COMMENT      VARCHAR(101) NOT NULL);

CREATE TABLE PARTSUPP (
    PS_PARTKEY     INTEGER NOT NULL,
    PS_SUPPKEY     INTEGER NOT NULL,
    PS_AVAILQTY    INTEGER NOT NULL,
    PS_SUPPLYCOST  DECIMAL(15,2) NOT NULL,
    PS_COMMENT     VARCHAR(199) NOT NULL );

CREATE TABLE CUSTOMER (
```

```

C_CUSTKEY      INTEGER NOT NULL,
C_NAME         VARCHAR(25) NOT NULL,
C_ADDRESS       VARCHAR(40) NOT NULL,
C_NATIONKEY    INTEGER NOT NULL,
C_PHONE        CHAR(15) NOT NULL,
C_ACCTBAL     DECIMAL(15,2) NOT NULL,
C_MKTSEGMENT   CHAR(10) NOT NULL,
C_COMMENT      VARCHAR(117) NOT NULL);

CREATE TABLE ORDERS (
O_ORDERKEY      INT8 NOT NULL,
O_CUSTKEY      INTEGER NOT NULL,
O_ORDERSTATUS   CHAR(1) NOT NULL,
O_TOTALPRICE   DECIMAL(15,2) NOT NULL,
O_ORDERDATE    DATE NOT NULL,
O_ORDERPRIORITY CHAR(15) NOT NULL,
O_CLERK        CHAR(15) NOT NULL,
O_SHIPPRIORITY INTEGER NOT NULL,
O_COMMENT      VARCHAR(79) NOT NULL);

CREATE TABLE LINEITEM (
L_ORDERKEY      INT8 NOT NULL,
L_PARTKEY      INTEGER NOT NULL,
L_SUPPKEY      INTEGER NOT NULL,
L_LINENUMBER   INTEGER NOT NULL,
L_QUANTITY     DECIMAL(15,2) NOT NULL,
L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL,
L_DISCOUNT     DECIMAL(15,2) NOT NULL,
L_TAX          DECIMAL(15,2) NOT NULL,
L_RETURNFLAG   CHAR(1) NOT NULL,
L_LINESUPPLY   CHAR(1) NOT NULL,
L_SHIPDATE     DATE NOT NULL,
L_COMMITDATE   DATE NOT NULL,
L_RECEIPTDATE  DATE NOT NULL,
L_SHIPINSTRUCT CHAR(25) NOT NULL,
L_SHIPMODE     CHAR(10) NOT NULL,
L_COMMENT      VARCHAR(44) NOT NULL);

CREATE READABLE EXTERNAL TABLE e_NATION (LIKE NATION)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/nation/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_REGION (LIKE REGION)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/region/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_PART (LIKE PART)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/part/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_SUPPLIER (LIKE SUPPLIER)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/supplier/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_PARTSUPP (LIKE PARTSUPP)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/partsupp/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_CUSTOMER (LIKE CUSTOMER)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/customer/') FORMAT 'csv';

```

```
CREATE READABLE EXTERNAL TABLE e_ORDERS (LIKE ORDERS)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/orders/') FORMAT 'csv';

CREATE READABLE EXTERNAL TABLE e_LINEITEM (LIKE LINEITEM)
LOCATION ('qs://hashdata-public.pek3a.qingstor.com/tpch/1g/lineitem/') FORMAT 'csv';
```

2. 执行如下命令将保存在对象存储上面的 TPC-H 数据拷贝插入到数据仓库表中。

```
INSERT INTO NATION SELECT * FROM e_NATION;
INSERT INTO REGION SELECT * FROM e_REGION;
INSERT INTO PART SELECT * FROM e_PART;
INSERT INTO SUPPLIER SELECT * FROM e_SUPPLIER;
INSERT INTO PARTSUPP SELECT * FROM e_PARTSUPP;
INSERT INTO CUSTOMER SELECT * FROM e_CUSTOMER;
INSERT INTO ORDERS SELECT * FROM e_ORDERS;
INSERT INTO LINEITEM SELECT * FROM e_LINEITEM;
```

3. 现在可以开始运行样例查询了。

这里所采用的数据集和查询是商业智能计算测试 TPC-H。TPC-H 是美国交易处理效益委员会组织制定的用来模拟决策支持类应用的一个测试集。TPC-H 实现了一个数据仓库，共包含 8 个基本表，其数据量可以设定从 1G 到 3T 不等。在这个样例中，我们选择了 1G 的数据集。TPC-H 基准测试包括 22 个查询，其主要评价指标是各个查询的响应时间，即从提交查询到结果返回所需时间。这里只提供了前三条查询语句。关于 TPC-H 完整 22 条查询语句以及详细介绍可参考 [TPC-H 主页](#)。

```
-- This query reports the amount of business that was billed, shipped, and returned.

select
    l_returnflag,
    l_linenstatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= '1998-12-01'
group by
    l_returnflag,
    l_linenstatus
order by
    l_returnflag,
    l_linenstatus;

-- This query finds which supplier should be selected to place an order for a given part in a given region.

select
```

```

s.s_acctbal,
s.s_name,
n.n_name,
p.p_partkey,
p.p_mfgr,
s.s_address,
s.s_phone,
s.s_comment
from
supplier s,
partsupp ps,
nation n,
region r,
part p,
(select p_partkey, min(ps_supplycost) as min_ps_cost
from
part,
partsupp ,
supplier,
nation,
region
where
p_partkey=ps_partkey
and s_suppkey = ps_suppkey
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'EUROPE'
group by p_partkey ) g
where
p.p_partkey = ps.ps_partkey
and g.p_partkey = p.p_partkey
and g. min_ps_cost = ps.ps_supplycost
and s.s_suppkey = ps.ps_suppkey
and p.p_size = 45
and p.p_type like '%NICKEL'
and s.s_nationkey = n.n_nationkey
and n.n_regionkey = r.r_regionkey
and r.r_name = 'EUROPE'
order by
s.s_acctbal desc,
n.n_name,
s.s_name,
p.p_partkey
LIMIT 100;

-- This query retrieves the 10 unshipped orders with the highest value.

select
l_orderkey,
sum(l_extendedprice * (1 - l_discount)) as revenue,
o_orderdate,
o_shipppriority
from
customer,
orders,
lineitem

```

```
where
    c_mktsegment = 'MACHINERY'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < '1995-03-15'
    and l_shipdate > '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
LIMIT 10;
```

## 步骤6: 寻找额外的资料和重设你的环境

完成这个入门教程后，你可以寻找更多额外的资料来学习和理解这个教程中介绍的概念，或者你可以将你的环境重置回原来的状态。如果你想尝试其他学习资料中提到的任务，你可以让集群一直运行着。不过，需要注意的是，只要集群还运行着，你将一直被收取费用。

### 额外学习资料

我们建议您可以通过如下资料来学习和理解这个教程中介绍的概念：

- HashData 数据仓库 [管理指南](#)
- HashData 数据仓库 [开发指南](#)

### 重置你的环境

当你完成了这个入门指南，你可以通过如下步骤重置你的环境：

- 删除步骤 3 中添加的防火墙规则和路由器转发规则。
- 删除样例集群。

# JSON 函数和操作符

用于创建和操作 JSON 数据的内置函数和操作符。

- [JSON 操作符](#)
- [JSON 创建函数](#)
- [JSON 处理功能](#)

注解：对于 json 的值，对象包含重复的键/值对，也保留所有键/值对。处理功能将最后一个值视为可操作的。

## JSON 操作符

该表描述了可用于的操作符 json 的数据类型。

表 1. json 操作符

| 操作符 | 右操作数类型 | 描述                   | 示例                                                | 示例结果         |
|-----|--------|----------------------|---------------------------------------------------|--------------|
| ->  | int    | 获取 JSON 数组元素（从零索引）。  | '[{"a":"foo"}, {"b":"bar"}, {"c":"baz"}]':json->2 | {"c":"baz"}  |
| ->  | text   | 通过键获取 JSON 对象字段。     | '{"a": {"b": "foo"}}':json->'a'                   | {"b": "foo"} |
| ->> | int    | 获取 JSON 数组元素作为文本。    | '[1,2,3]':json->>2                                | 3            |
| ->> | text   | 获取 JSON 对象字段作为文本。    | '{"a":1, "b":2}':json->>'b'                       | 2            |
| #>  | text[] | 在指定的路径获取 JSON 对象。    | '{"a": {"b": {"c": "foo"} }}':json#>'{a,b}'       | {"c": "foo"} |
| #>> | text[] | 以指定路径获取 JSON 对象作为文本。 | '{"a": [1,2,3], "b": [4,5,6]}':json#>'{a,2}'      | 3            |

## JSON 创建函数

此表描述了创建的函数的 json 值。

表 2. JSON 创建函数

| 函数 | 描述                              | 示例 | 示例结果 |
|----|---------------------------------|----|------|
|    | 将值作为有效的 JSON 对象返回。数组和复合类型被递归处理并 |    |      |

|                                         |                                                                                                                        |                                                       |                                 |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|---------------------------------|
|                                         | <p>转换为数组和对象。如果输入包含类型为 json 的造型，则使用造型函数执行转换；否则，将生成 JSON 标量值。对于除数字，布尔值或空值之外的任何标量类型，将使用文本表示，正确引用和转义，以使其为有效的 JSON 字符串。</p> | <pre>to_json('Fred said "Hi."')::text</pre>           | <pre>"Fred said \"Hi.\""</pre>  |
| array_to_json(anyarray [, pretty_bool]) | <p>将数组作为 JSON 数组返回。HashData 数据库的多维数组成为数组的 JSON 数组。如果 pretty_bool 是 true，则在第一个元素之间添加换行符。</p>                            | <pre>array_to_json('{{1,5}, {99,100}}')::int[])</pre> | <pre>[[1,5], [99,100]]</pre>    |
| row_to_json(record [, pretty_bool])     | <p>将该行作为 JSON 对象返回。如果 elements if pretty_bool 是 true 则在第 1 级元素之间添加换行符。</p>                                             | <pre>row_to_json(row(1, 'foo'))</pre>                 | <pre>{"f1":1, "f2":"foo"}</pre> |

注解：除了提供精细打印选项外，array\_to\_json 和 row\_to\_json 具有与 to\_json 相同的行为。对于 to\_json 描述的行为也适用于由其他 JSON 创建函数转换的单个值。

## JSON 处理函数

此表描述处理 json 值的函数。

表 3. JSON 处理函数

| 函数                                                                 | 返回类型                                                     | 描述                                             | 示例                                                                                                 |
|--------------------------------------------------------------------|----------------------------------------------------------|------------------------------------------------|----------------------------------------------------------------------------------------------------|
| json_each(json)                                                    | set of key text, value json set of key text, value jsonb | 将最外层的 JSON 对象扩展为一组键/值对。                        | <pre>select * from json_each('{"a": "1", "b": "bar"}')</pre>                                       |
| json_each_text(json)                                               | setof key text, value text                               | 将最外层的 JSON 对象扩展为一组键/值对。返回的值是类型 text。           | <pre>select * from json_each_text('{"a": "foo", "b": "bar"}')</pre>                                |
| json_extract_path(from_json json, VARIADIC path_elems text[])      | json                                                     | 返回指定为的 JSON 值 path_elems。相当于 #> 操作符。           | <pre>json_extract_path('{"f2": {"f3": 1}, "f4": {"f5": 99, "f6": "foo"} }', 'f4')</pre>            |
| json_extract_path_text(from_json json, VARIADIC path_elems text[]) | text                                                     | 返回指定为的 JSON 值 path_elems 作为文本。相当于 #>> 操作符。     | <pre>json_extract_path_text('{"f2": {"f3": 1}, "f4": {"f5": 99, "f6": "foo"} }', 'f4', 'f6')</pre> |
| json_object_keys(json)                                             | setof text                                               | 返回最外层 JSON 对象中的键集。                             | <pre>json_object_keys('{"f1": "abc", "f2": "a", "f3": "b"}')</pre>                                 |
| json_populate_record(base anyelement, from_json json)              | anyelement                                               | 扩展对象 from_json 其列与由 base 定义的记录类型匹配。            | <pre>select * from json_populate_record(null::myrowt, '{"a": 1, "b": 2}')</pre>                    |
| json_populate_recordset(base anyelement, from_json json)           | setof anyelement                                         | 扩展最外层的对象数组 from_json 到一组行，其列与定义的记录类型 base 相匹配。 | <pre>select * from json_populate_recordset(null::myr, [{"a": 1, "b": 2}, {"a": 3, "b": 4}])</pre>  |
| json_array_elements(json)                                          | setof json                                               | 将 JSON 数组 扩展为一组 JSON 值。                        | <pre>select * from json_array_elements('[1, true, [2, false]])</pre>                               |

注意：许多这些函数和操作符将 Unicode 字符串中的 Unicode 转义转换为常规字符。这些函数为不能在数据库编码中表示的字符引发错误。

对于 json\_populate\_record 和 json\_populate\_recordset，从 JSON 类型强制是尽力而为，可能不会导致某些类型的期望值。JSON 键与目标行类型中的相同列名匹配。输出中不会出现未显示在目标行类型中的 JSON 字

段，并且不匹配任何 JSON 字段的目标列返回 NULL。