



链滴

在 kubernetes 上运行 Postgres 以及 Postgres 特性

作者: [Gakkiyomi2019](#)

原文链接: <https://ld246.com/article/1607675068995>

来源网站: [链滴](#)

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)



在kubernetes上运行Postgres以及Postgres特性

Postgres 是一款开源的关系性数据库，并且自称是世界上最先进的开源关系性数据库。经过30多年积极开发，已在可靠性，功能健壮性和性能方面赢得了极高的声誉。

安装

官方文档对于源代码安装和二进制安装的教程已经十分详细这里不详细介绍了，这里我会介绍**如何在kubernetes上安装postgres数据库**

第一步：下载postgres的镜像,需要docker环境,这里以postgres:11.3为例

```
docker pull postgres:11.3
```

```
[root@fangcong ~]# docker images | grep postgres
docker.io/postgres      11.3          4e045cb8eecd      18 months ago      312 MB
[root@fangcong ~]#
```

第二步：创建namesapce postgres 我们会将postgres放在postgres 的namespace下

```
kubectl create ns postgres
```

```
[root@paas-54 fangcong]# kubectl get ns | grep postgres
postgres      Active    21d
[root@paas-54 fangcong]#
```

第三步：准备好postgres的资源定义文件，有三个: **cm.yaml(挂载配置文件), postgres.yaml(资源义), svc.yaml(服务发现)

cm.yaml

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: postgres-cm          # configmaps的名称
  namespace: postgres       # configmaps的域
  labels:
    app: postgres-cm        # 标签
data:
  master.conf: |
    listen_addresses = '*'   # 监听所有IP
    archive_mode = on        # 允许归档
    archive_command = 'cp %p /var/lib/postgresql/data/pg_archive/%f' # 用该命令来归档logfil
    segment
    wal_level = hot_standby  # 开启热备
    max_wal_senders = 32     # 这个设置了可以最多有几个流复制连接，差不多有几
    从，就设置几个
    wal_keep_segments = 64   # 设置流复制保留的最多的xlog数目，一份是 16M，
    意机器磁盘 16M*64 = 1G
    wal_sender_timeout = 60s # 设置流复制主机发送数据的超时时间
    max_connections = 100    # 这个设置要注意下，从库的max_connections必须
    大于主库的
  pg_hba.conf: |
    local all all trust
    host all all 127.0.0.1/32 trust
    host all all ::1/128 trust
    local replication all trust
    host replication all 127.0.0.1/32 trust
    host replication all ::1/128 trust
    host all all trust
    host all all 0.0.0.0/0 trust
    host replication postgres 0.0.0.0/0 trust
  slave.conf: |
    wal_level = hot_standby  # 热备
    max_connections = 1000   # 一般查多于写的从应用从库的最大连接数要比较大
    hot_standby = on         # 说明这台机器不仅仅是用于数据归档，也用于数据查询
    max_standby_streaming_delay = 30s # 数据流备份的最大延迟时间
    wal_receiver_status_interval = 10s # 多久向主报告一次从的状态，当然从每次数据复制
    会向主报告状态，这里只是设置最长的间隔时间
    hot_standby_feedback = on # 如果有错误的数据库复制，是否向主进行反馈
    log_destination = 'csvlog' # 日志文件的位置
    logging_collector = on    # 启动日志信息写到了当前terminal的stdout,系统操作
    志信息写到了pg_log/enterprisedb-*.log
    log_directory = 'log'    # 日志文件目录
  recovery.conf: |
    standby_mode = on        # 启动从节点
    primary_conninfo = 'host=postgres-0.postgres.postgres port=5432 user=postgres passwo
    d=r00tme' # 主节点信息
    recovery_target_timeline = 'latest' # 更新备份[root@paas-54 postgres]

```

svc.yaml

```

apiVersion: v1
kind: Service
metadata:

```

```

name: postgres      # 服务名
namespace: postgres # 服务所在域
labels:             # 标签
  app: postgres     # 键值对为{"app":"postgres"}的标签
spec:
  ports:            # 端口
  - name: postgres # 端口名
    port: 5432      # 内部服务访问Service的端口
  clusterIP: None  # Headless Service,设置后服务没有内网IP, 访问服务会直接寻路到Pod
  selector:        # 服务选择器
    app: postgres  # 服务选择标签键值对为{"app":"postgres"}的Pod
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-read # 服务名
  namespace: postgres # 服务所在域
  labels:             # 标签
    app: postgres     # 键值对为{"app":"postgres"}的标签
spec:
  externalIPs:      # 暴露Service到外部IP
  - 192.168.1.225  # 填宿主机ip即可
  ports:            # 端口
  - name: postgres # 端口名
    port: 5432      # 内部服务访问Service的端口
    targetPort: 5432 # Pod内的端口
  selector:        # 服务选择器
    app: postgres  # 服务选择标签键值对为{"app":"postgres-slave"}的Pod

```

postgres.yaml

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres # 主库
  namespace: postgres # 使用postgres域
spec:
  replicas: 1 # 创建副本数
  selector:
    matchLabels:
      app: postgres # 被{"app":"postgres"}的标签匹配
  serviceName: postgres # Statefulset使用的Headless Service为postgres
  template: # 创建Pod模板
    metadata:
      name: postgres # 创建Pod名
      labels:
        app: postgres # Pod对应的标签
        node: master # 只能在主节点上部署
    spec:
      tolerations: # 1分钟如果节点不可达视为异常
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 60

```

```

- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 60
affinity: # 亲和性
  nodeAffinity: # Pod亲和性
    requiredDuringSchedulingIgnoredDuringExecution: # 硬要求
      nodeSelectorTerms:
        - matchExpressions:
            - key: node # 根据label是node的键来配对
              operator: In # 适用表达式
              values: # 值如下
                - master # 调度到有master标签的节点
    podAntiAffinity: # 如果检测到节点有{app:postgres}则不部署, 避免postgr
s在同一节点重复部署
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - postgres
            topologyKey: "kubernetes.io/hostname"
terminationGracePeriodSeconds: 0 # 异常立即删除
initContainers: # 初始化容器
- command: # 命令
  - bash
  - "-c"
  - |
    set -ex # 写入环境变量
    [[ `hostname` =~ -([0-9]+)$ ]] # 获取容器的主机名, 用于数据库同步判断
    ordinal=${BASH_REMATCH[1]} # 获取主机名后获取主机编号
    if [[ $ordinal -eq 0 ]]; then # 如果是postgres-0,也就是主节点
      echo test
    else # 如果是非postgres-0,则为从节点
      rm /var/lib/postgresql/data1 -fr
      mkdir -p /var/lib/postgresql/data1
      pg_basebackup -h postgres-0.postgres.postgres -U postgres -D /var/lib/postgresql/d
ta1 -X stream -P # 与postgres-0进行同步
      \cp /mnt/config-map/slave.conf /var/lib/postgresql/data1/postgresql.conf -f #
入配置文件
      \cp /mnt/config-map/recovery.conf /var/lib/postgresql/data1/recovery.conf -f #
写入配置文件
      rm /var/lib/postgresql/data/* -fr
      mv /var/lib/postgresql/data1/* /var/lib/postgresql/data/
    fi
env:
- name: POSTGRES_USER # 数据库用户
  value: postgres
- name: POSTGRES_DB # 数据库DB
  value: test
name: init-postgres # 容器名
image: postgres:11.3 # 镜像名
imagePullPolicy: IfNotPresent # 若镜像存在, 则不拉取

```

```

volumeMounts:                                # 容器内挂载目录
- name: postgres-pv                          # 挂载文件名
  mountPath: /var/lib/postgresql/data/       # 挂载路径
- name: config-map                            # configmap的文件存储路径
  mountPath: /mnt/config-map                 # 存储路径
- name: tz
  mountPath: /etc/localtime
containers:                                    # Pod中容器
- name: postgres                              # 容器名
  image: postgres:11.3                       # 镜像名
  ports:                                       # 端口
- name: postgres                              # 端口名
  containerPort: 5432                         # 端口号
  volumeMounts:                               # 容器内挂载目录
- name: postgres-pv                          # 挂载文件名
  mountPath: /var/lib/postgresql/data/       # 挂载路径
- name: config-map                            # configmap的文件存储路径
  mountPath: /mnt/config-map                 # 存储路径
- name: tz
  mountPath: /etc/localtime
env:                                           # 环境变量
- name: TZ                                     # 时区, 键
  value: Asia/Shanghai                       # 值
volumes:                                       # Pod外挂载位置
- name: postgres-pv                          # 挂载文件名
  hostPath:                                   # 挂载在主机目录
    path: /srv/system/postgres/data          # 主机目录路径
- name: tz
  hostPath:
    path: /etc/localtime
- name: config-map                            # 挂载文件名
  configMap:                                  # 挂载在configmaps
    name: postgres-cm                       # configmaps的名字

```

创建数据挂载目录

```
mkdir data
```

启动

```
kubectl apply -f .
```

写入配置文件

这里由于一些原因，导致需要手动将文件写入，这一步是可以省略的，可以在yaml定义里面去优化，体优化以后在进行。

```
写入postgresql.conf
```

```
kubectl exec -it postgres-0 -n postgres -- sh -c 'cp /mnt/config-map/master.conf /var/lib/postgresql/data/postgresql.conf -f'
```

写入pg_hba.conf

```
kubectl exec -it postgres-0 -n postgres -- sh -c 'cp /mnt/config-map/pg_hba.conf /var/lib/postgresql/data/pg_hba.conf -f'
```

重启

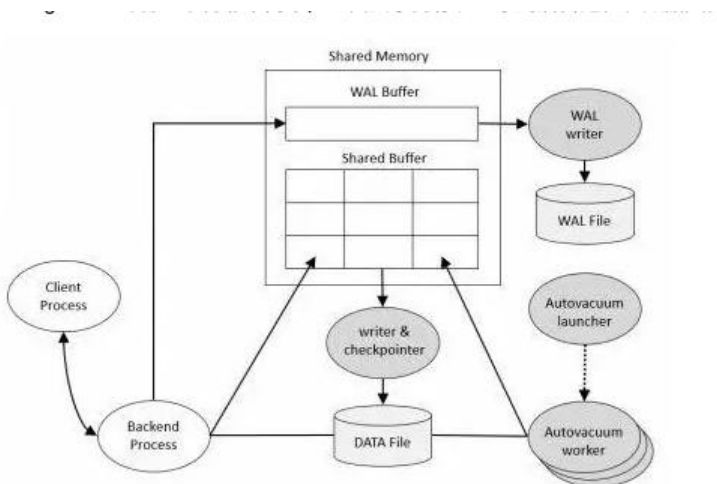
```
kubectl delete pods postgres-0 -n postgres
```

```
[root@nap-146 tasks]# kubectl get pods -n postgres
NAME                                READY  STATUS   RESTARTS  AGE
postgres-0                          1/1    Running  2         14d
[root@nap-146 tasks]#
```

架构

postgres是典型的C/S架构,并且是与mysql的多线程架构不同,postgres是多进程架构。

下面此为进程架构图



内存架构

postgres中的内存可以分为两大类:

1. 本地内存区域

- 由每个后端进程分配以供自己使用

2. 共享内存区域

- Shared Buffer
 - postgres为了减少磁盘IO次数, 将表和索引中的页面从持久性存储加载到此处, 并直接对其进行操作。
- WAL Buffer
 - 为确保服务器故障不会丢失任何数据, PostgreSQL支持WAL机制。WAL数据 (也称为XLOG)

记录) 是PostgreSQL中的事务日志; WAL缓冲区是在写入持久性存储之前WAL数据的缓冲区。

进程类型

名字	含义	作用
Postmaster (Daemon) Process		postgres server
后台驻留进程是PostgreSQL启动时第一个启动的进程。启动时, 他会执行恢复、初始化共享内存爱的运行后台进程操作。正常服役期间, 当有客户端发起链接请求时, 它还负责创建后端进程。		
Background Process		后台进程
Postmaster和Backend相比, 不可能简单地解释每个功能, 因为这些功能取决于个别的特定功能和PostgreSQL内部。		
Backend Process		后端进程
由postgres服务器进程启动, 并处理由一个连接的客户端发出的所有查询。它通过单个TCP连接与客户端通, 并在客户端断开连接时终止。		
Client Process	8 bytes	客户端
程需要和后端进程配合使用, 处理每一个客户链接。通常情况下, Postmaster进程会派生一个子进程来处理用户链接。		

Background进程详细列表

进程	作用
logger	将错误信息写到log日志中
checkpointer 数据文件	当检查点出现时, 将脏内存块写到
writer	周期性的将脏内存块写入文件
wal writer	将WAL缓存写入WAL文件
Autovacuum launcher	当自动vacuum被启用时
用来派生autovacuum工作进程。autovacuum进程的作用是在需要时自动对膨胀表执行vacuum操。	
archiver	在归档模式下时, 复制WAL文件到特定 路径下。
stats collector	用来收集数据库统计信息, 例如 话执行信息统计 (使用pg_stat_activity视图) 和表使用信息统计 (pg_stat_all_tables视图)

特性

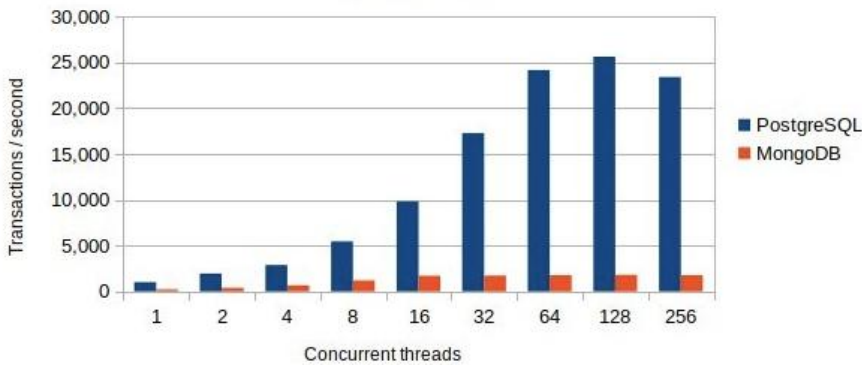
在使用方面, postgres性能十分强劲, 使用起来也十分方便,而且postgres深耕多年,其目标早已不仅是关系性数据库了。

不只是关系性数据库

postgresql 自称**世界上最先进的开源关系数据库**, 但其实postgres丰富的数据类型以及支持json,这它完全可以像mongodb一样来存储文档,并且postgres的性能更好

MongoDB vs PostgreSQL - Transaction Performance

(more is better)



postgres和mongodb 在文档数据方面的性能比较 [postgres vs mongodb](#)

丰富的数据类型

postgres内置了丰富的数据类型和支持自定类型的机制就决定了他的野心不至于关系性数据库，postgres在文档数据库方面的能力也十分强大，当然如果没有足够丰富的数据类型，postgres也不会来分mongodb和es的蛋糕

网络地址类型

PostgreSQL提供用来存储 IPv4、IPv6 和 MAC 地址等三种网络地址的数据类型，我们使用这些数据类型来保存网络地址比用varchar或者text好，因为postgres内置了这些类型时提供输入校验和一些络算法函数，这样我们就可以在数据库层面上进行一些简单的网络计算，可以减少我们自身应用的内消耗。下面是数据类型的简介。

名字	存储尺寸	描述
<code>cidr</code>	7或19字节	IPv4和IPv6网络
<code>inet</code> 及网络	7或19字节	IPv4和IPv6主机
<code>macaddr</code>	6字节	MAC地址
<code>macaddr8</code> UI-64格式)	8 bytes	MAC地址 (

json

JSON类型也是postgres的卖点之一，json类型也是postgres可以做文档数据库的基础。json也可以存储为text或者varchar类型，但是JSON数据类型提供输入正则和许多JSON相关的函数和操作符我们可以十分方便的检索整个json或者其中的key或者value，这样我们也可以在sql层面对json对进行计算，这与mongodb的理念不谋而合，但postgres的json操作符使用简单,性能更是完爆mongodb。

`json` 和 `jsonb`。它们几乎接受完全相同的值集合作为输入。主要的实际区别之一是效率。`json`数据类型存储输入文本的精准拷贝，处理函数必须在每次执行时必须重新解析该数据。而`jsonb`数据被存储一种分解好的二进制格式中，它在输入时要稍慢一些，因为需要做附加的转换。但是`jsonb`在处理时快很多，因为不需要解析。`jsonb`也支持索引，这也是一个令人瞩目的优势。

数组类型

PostgreSQL允许一个表中的列定义为变长多维数组。可以创建任何内建或用户定义的基类、枚举类、组合类型或者域的数组。

这给我们设计数据结构提供了非常高的灵活度。

```
CREATE TABLE sal_emp (  
  name      text,  
  pay_by_quarter integer[],  
  schedule  text[][],  
  body      json  
);
```

我可以使用postgres建立上面的这种表,这挣脱了关系性型数据库的束缚, 拥抱json

几何类型

几何数据类型表示二维的空间物体。

名字	存储尺寸	表示	描述
point (x,y)	16字节	平面上的点	
line (A,B,C)	32字节	无限长的线	{
lseg (x1,y1),(x2,y2)	32字节	有限线段	((x
box (x1,y1),(x2,y2)	32字节	矩形框	((x1,
path (x1,y1),...)	16+16n字节	封闭路径 (类似于多边形)	
path (x1,y1),...]	16+16n字节	开放路径	
polygon (x1,y1),...)	40+16n字节	多边形 (类似于封闭路径)	
circle (中心点和半径)	24字节	圆	<(x,y),r

Postgres有一系列丰富的函数和操作符可用来进行各种几何操作, 如缩放、平移、旋转和计算相交等

索引

postgres提供了多种适用于不同场景的索引供我们选择使用

- B-tree 索引: **CREATE INDEX**命令创建适合于大部分情况的B-tree 索引。
- Hash 索引: 只能处理简单等值比较。不论何时当一个索引列涉及到一个使用了 **=**操作符的比较时查询规划器将考虑使用一个Hash索引。下面的命令将创建一个Hash索引:

```
CREATE INDEX name ON policy USING HASH (column);
```

- GiST 索引: GiST索引并不是一种单独的索引，而是可以用于实现很多不同索引策略的基础设施 postgres的标准库提供了用于多种二维几何数据类型的GiST操作符。

-- 它将找到离给定目标点最近的10个位置。

```
SELECT * FROM places ORDER BY location <-> point '(101,456)' LIMIT 10;
```

- GIN 索引: GIN 索引是“倒排索引”，它适合于包含多个组成值的数据值，例如数组。倒排索引中每一个组成值都包含一个单独的项，它可以高效地处理测试指定组成值是否存在的查询。

```
CREATE INDEX idxgin ON policy USING gin (source,destination); -- 在 source 和 destination 段上建立联合索引 遵循最左法则
```

- BRIN 索引: (块范围索引的缩写) 存储有关存放在一个表的连续物理块范围上的值摘要信息。与 GiS、SP-GiST 和 GIN 相似，BRIN 可以支持很多种不同的索引策略，并且可以与一个 BRIN 索引配合使用的特定操作符取决于索引策略。

运维

这里会记录一些我常遇到的postgres问题和命令

postgres 数据备份恢复

#导出sql文件

```
pg_dump -U postgres nap > nap.sql
```

#导入

```
drop database nap; #删除原来的库
```

```
CREATE DATABASE nap;
```

```
psql -U postgres -d nap -f nap.sql
```