阿里云

PostgreSQL监控实战

基于Pigsty解决实际监控问题

冯若航

PostgreSQL DBA

2021/01/07

阿里云

PostgreSQL监控实战

基于Pigsty解决实际监控问题

冯若航

PostgreSQL 专家 / Pigsty 作者

2021/01/07

课程大纲

- 快速上手：利用监控系统观察数据库状态
- 解决方案：如何部署属于自己的监控系统
- 解决问题：如何用监控系统解决实际问题

01 快速上手

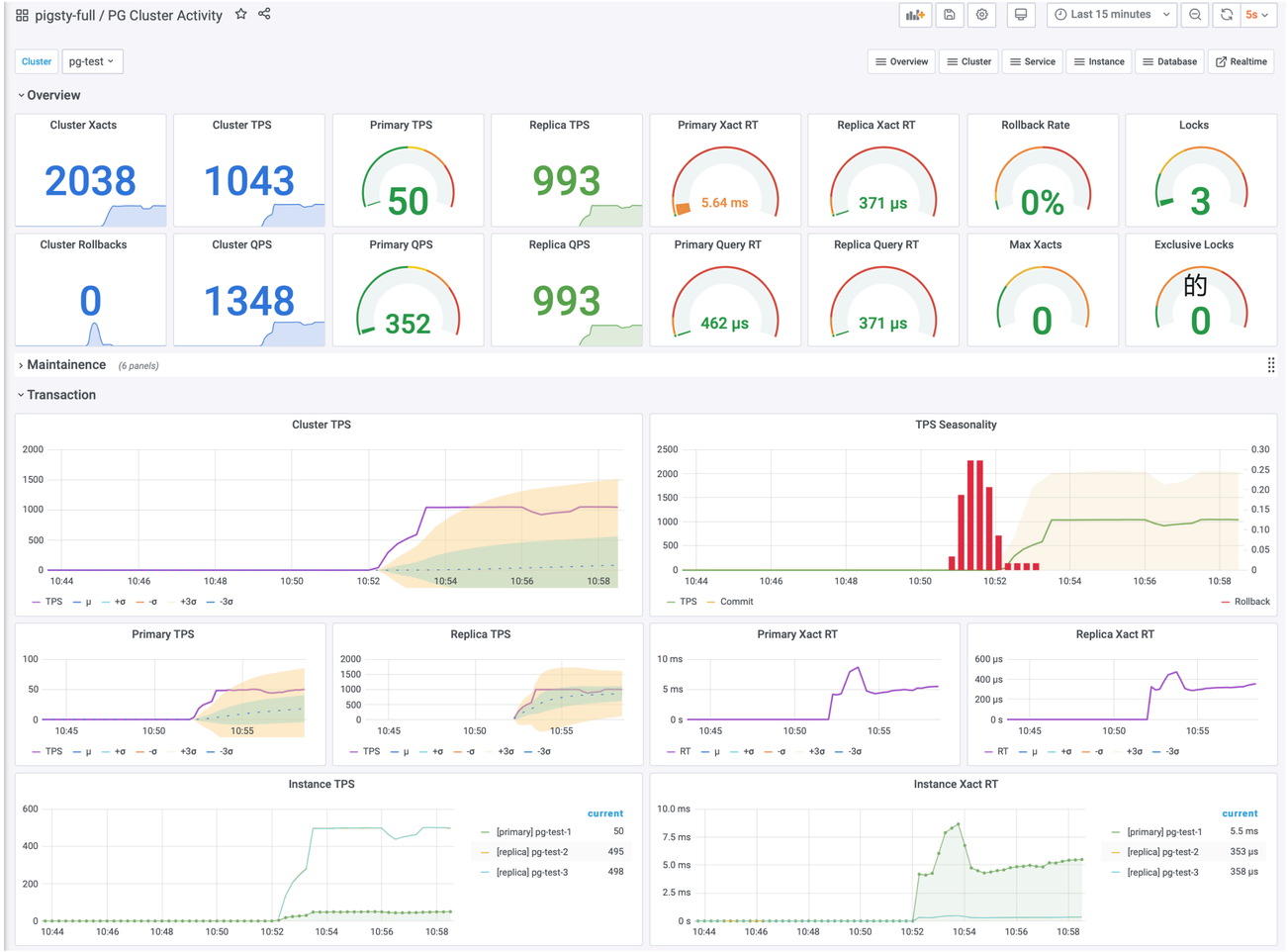
利用监控系统观察数据库状态

实验1：利用监控系统观测查询负载

已有一主两从的数据库集群 pg-test ，为集群添加50的读写TPS，1000的只读TPS，从监控系统观察负载。

从库1K只读TPS
```
while true; do pgbench -nv -P1 -c4 --select-only --rate=1000 -T10 postgres://test:test@pg-test:5434/test; done
```

主库50写入TPS
```
while true; do pgbench -nv -P1 -c2 --rate=50 -T10 postgres://test:test@pg-test:5433/test; done
```



用监控系统的图表数据
印证我们主动施加的负载

集群总TPS ≈ 1050

主库TPS ≈ 50

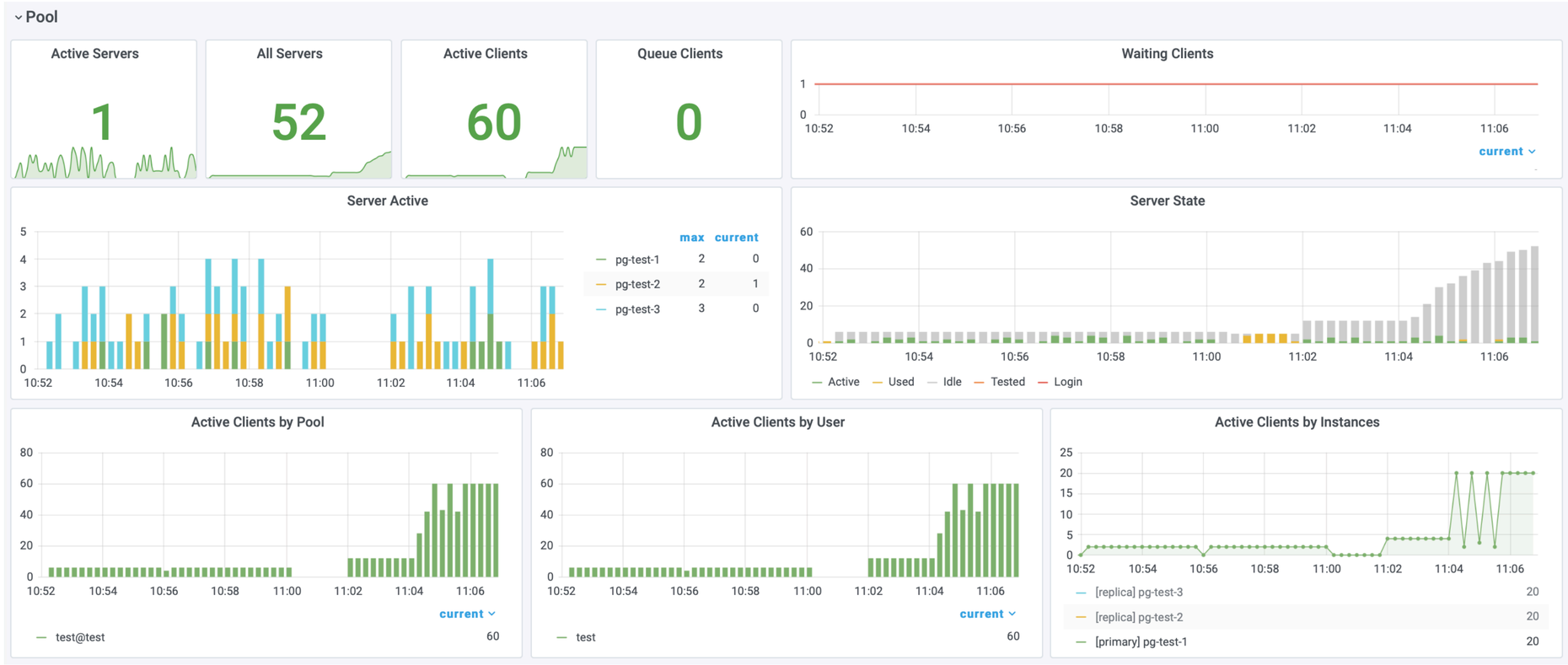
从库TPS ≈ 1000

每个从库实例TPS ≈ 500

使用十倍的连接数施加同样负载，观察系统状态

```
while true; do pgbench -nv -P1 -c40 --select-only --rate=1000 -T10 postgres://test:test@pg-test:5434/test; done
while true; do pgbench -nv -P1 -c20 --rate=50 -T10 postgres://test:test@pg-test:5433/test; done
```

原来我们使用了2条读写连接，4条只读连接，现在翻十倍变为20条和40条。观察连接池在这一过程中起到的作用。

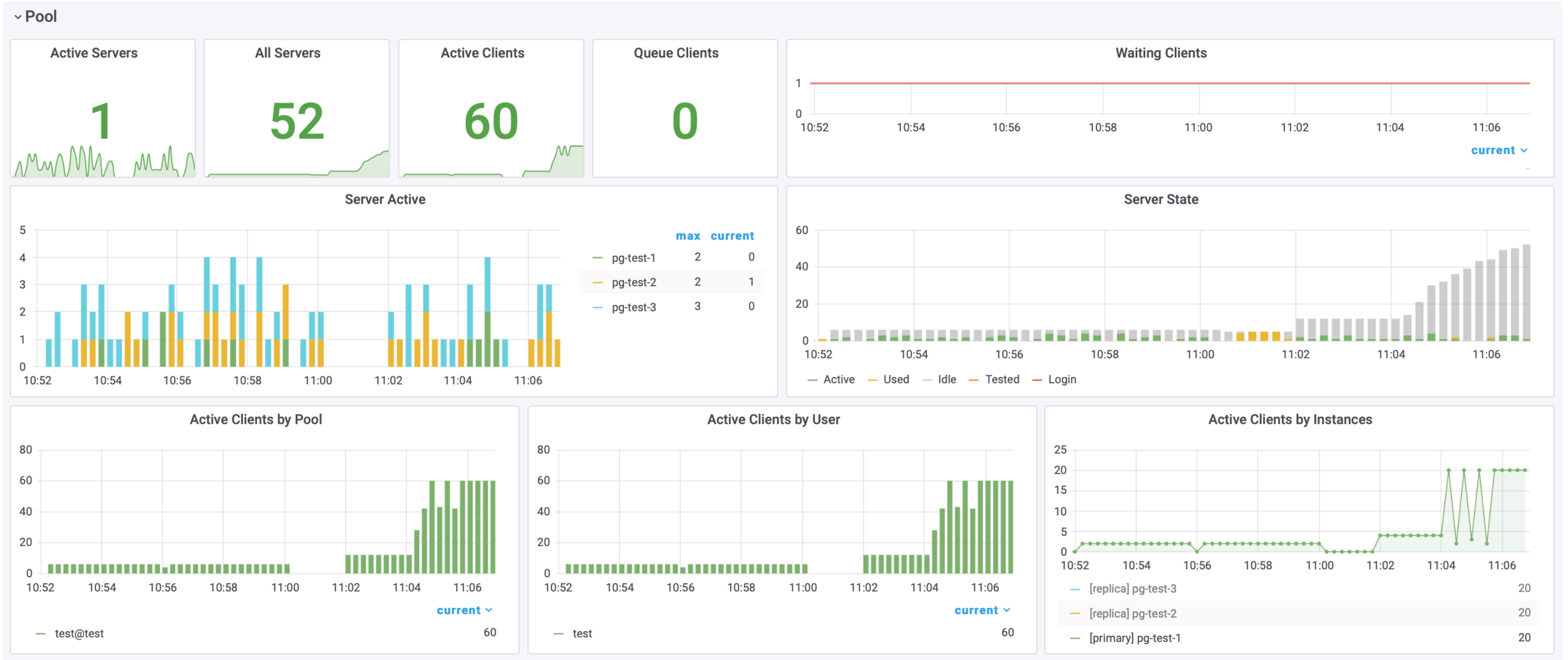


主库活跃客户端连接数 = 20　　从库活跃客户端连接数 = 40

集群活跃服务端连接数 < 5　　每个从库活跃客户端连接数 = 20

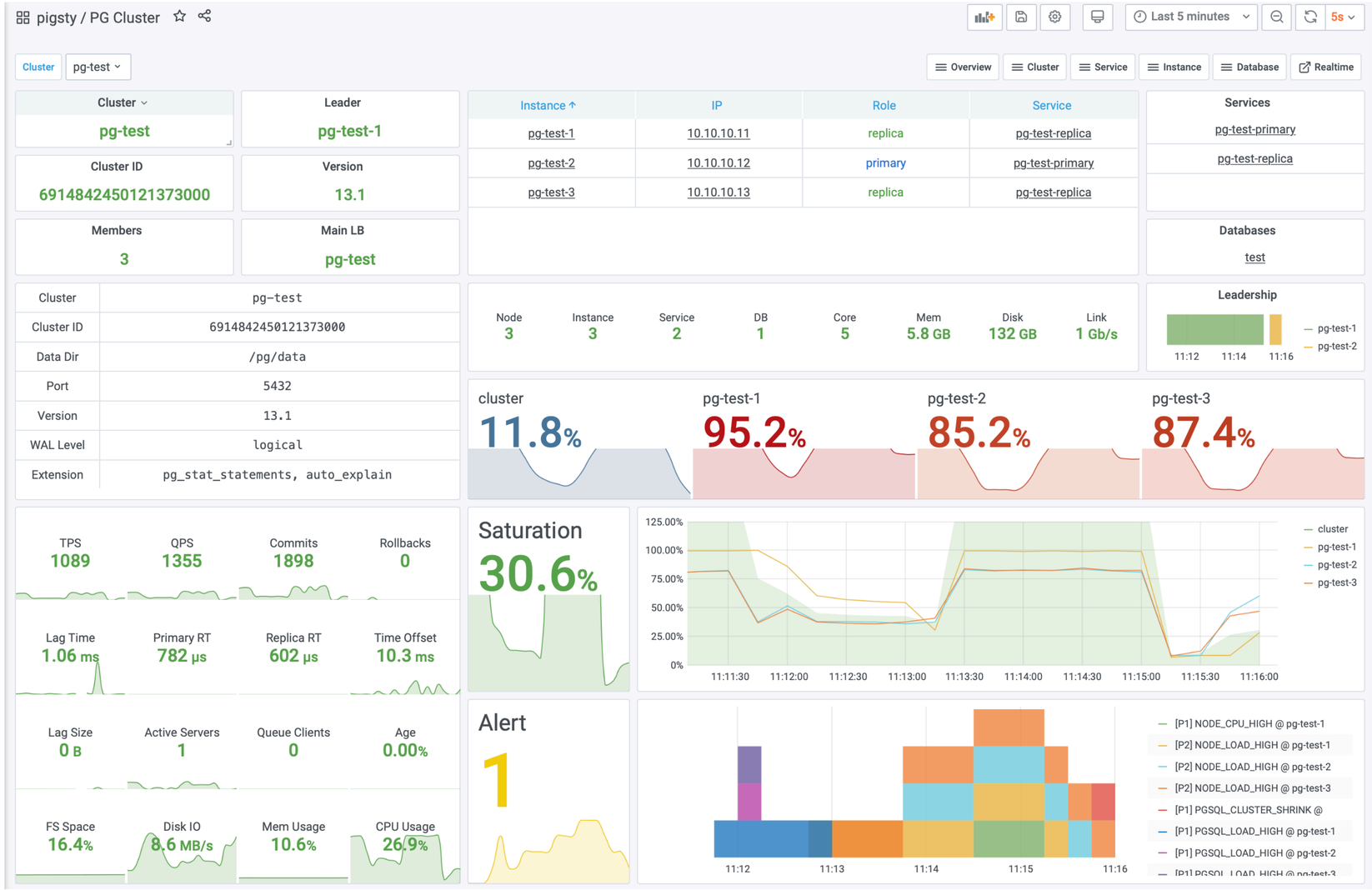
使用尽可能大的负载，观察系统在过载下的表现

```
while true; do pgbench -nv -P1 -c40 --select-only -T10 postgres://test:test@pg-test:5434/test; done
while true; do pgbench -nv -P1 -c20 -T10 postgres://test:test@pg-test:5433/test; done
```



实验2 重启主库，观察集群领导权交接

```
ssh -t node-1 sudo reboot
```



主库宕机
主库被移出集群
从库被提升为新主库
其他主库从新主库开始重建复制
旧主库重启完成
旧主库降级为从库
新从库重新加入集群

02 解决方案

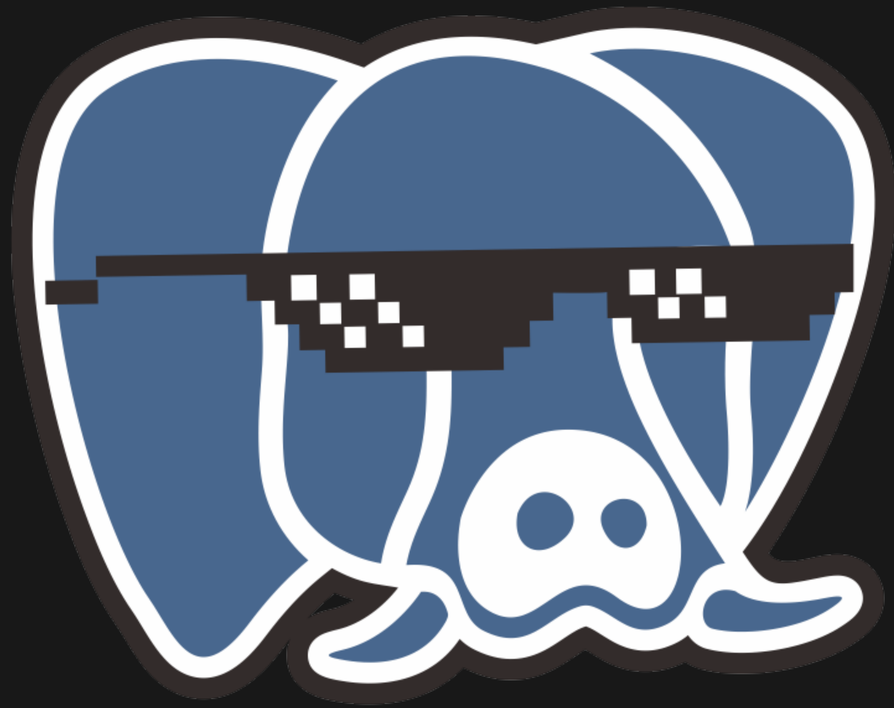
Pigsty

阿里云

PIGSTY /ˈpɪgˌstaɪ/

Postgres **I**n **G**raphic **STY**le

开源免费
PgSQL
监控系统
供给方案



Pigsty

https://pigsty.cc


PIGSTY
关于
文档
博客
中文
站内搜索...
Pigsty: 图形化PostgreSQL
公共示例
源码仓库
参考文档
世界上最好的PostgreSQL监控系统
https://pigsty.cc

Summary

Pigsty

Alerts

39

Nodes	Clusters	Instances	Replicas	Databases
224	92	225	127	129

Core	Mem	Disk	PCIE-SSD	Link
11664	88.6 TB	1.5 PB	680 TB	2 Tb/s

CPU Usage 2.3%

Disk Traffic 5.1 GiB/s

Net Traffic 1.6 GiB/s

Mem Usage 5.0%

Disk IOPS 677.5K iops

Max NTP Offset 2.47 ms

Dashboards

- Node
- PG Alert
- PG Cluster
- PG Cluster Activity
- PG Cluster All
- PG Cluster Database
- PG Cluster Node
- PG Cluster Persist
- PG Cluster Query
- PG Cluster Replication
- PG Cluster Session
- PG Cluster Stat
- PG Database
- PG Instance
- PG KPI
- PG Overivew
- PG Pgbouncer
- PG Proxy
- PG Query

Cluster	Size
pg-	3
pg-	2
pg-	2
pg-	2
pg-	3
pg-	2
pg-	9
pg-	2
pg-	7
pg-	2
pg-	2
pg-	2
pg-	2
pg-	2
pg-	2
pg-	2
pg-	6
pg-	2
pg-	2
pg-	3

Instance ↑	IP	Role
pg- t-0	10.189	primary
pg- t-1	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -2	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -2	10.	replica
pg- -0	10.	primary
pg- -2	10.	replica
pg- -3	10.	replica
pg- -0	10.	primary
pg- -1	10.	replica
pg- -0	10.	primary

Max CPU Usage 29.6%

Max PG Load 25.5%

Max Lag 9.6 MiB

Max RT 918 ms

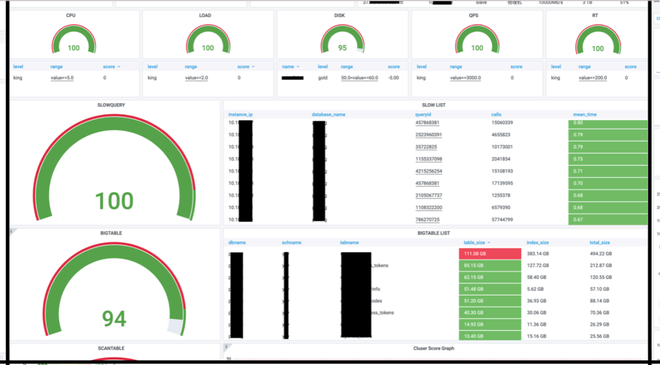
Max Active Server 19

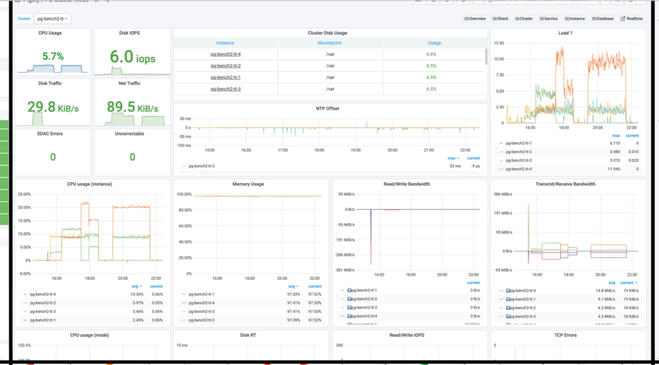
Max Queue Clients 0

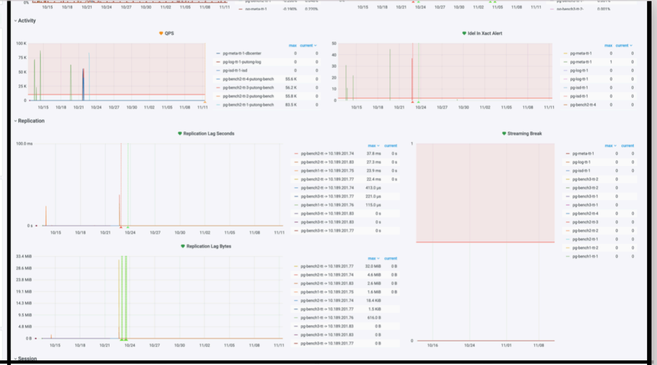
Max Disk Usage 81.9%

Max Age 596 Mil


Core
504
Mem
3.7 TB
Disk
63.06 TB
Link
10.0 Gbps
Alerts
6
Instances
9
Primary
1
Replicas
7
Timeline
2
Leadership
Cluster Topo
TPS
44358
Load
Lag
Queue
CPU Usage
Disk Usage
Xid Usage
Conn Usage
Loads


CPU
LOAD
DISK
QPS
RT
SLOWQUERY
SLOW LIST
BIGTABLE
BIGTABLE LIST
SCANTABLE
Cluster Score Graph


CPU Usage
Disk IOPS
Disk Traffic
Net Traffic
EDAC Errors
Uncorrectable
Cluster Disk Usage
Load 1
NTP Offset
CPU usage (instance)
Memory Usage
Read/Write Bandwidth
Transmit/Receive Bandwidth
CPU usage (node)
Disk RT
Read/Write IOPS
TCP Errors


Activity
QPS
Idle In Xact Alert
Replication
Replication Lag Seconds
Streaming Break
Replication Lag Bytes
Session

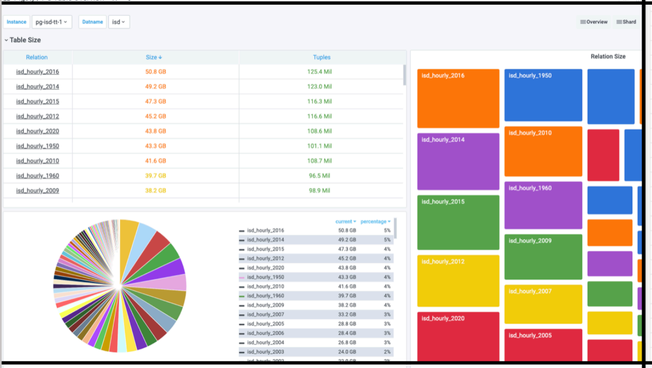
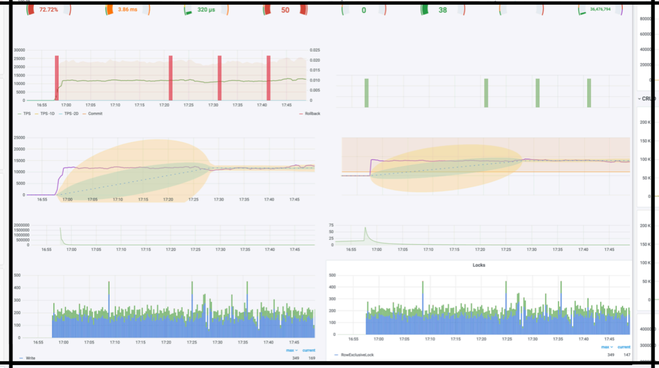
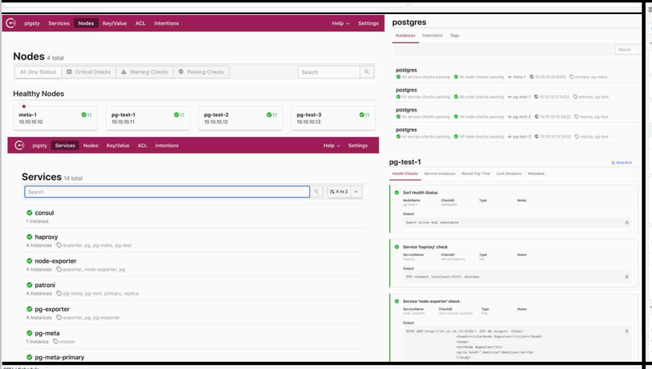


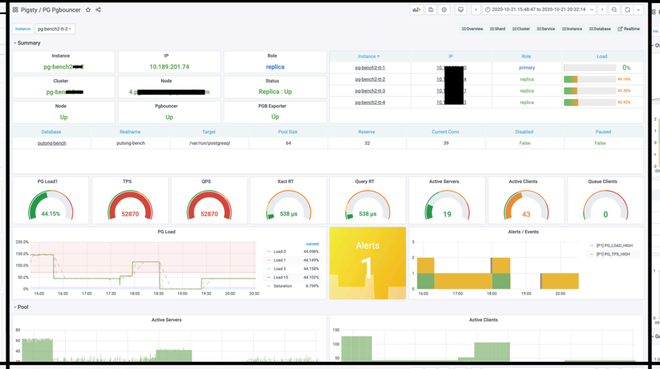
Table Size
Relation
Size
Tuples
isd_hourly_2016
isd_hourly_2014
isd_hourly_2015
isd_hourly_2012
isd_hourly_2020
isd_hourly_1950
isd_hourly_2010
isd_hourly_1960
isd_hourly_2009
Relation Size

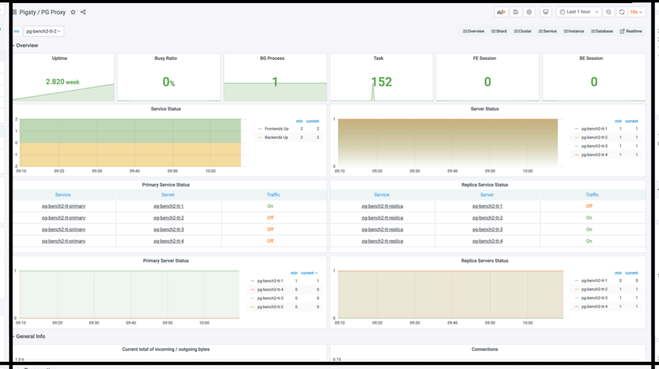

Load
Memory
Memory Active/Anonymous Structure
CPU usage (core)
Schedule
Process State / Thread


72.72%
3.86 ms
320 µs
50
0
38
Locks


Database TPS
Database QPS
CPU
Rows Selected by Instance
Rows Touched by Instance
Rows Selected by Database
Rows Touched by Database
Rows Modified
Rows Modified by Database

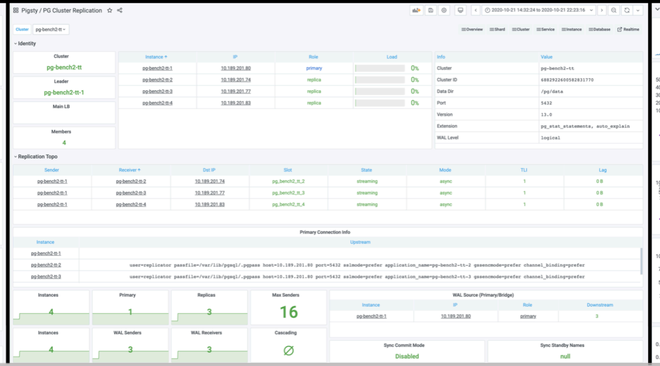

pigsty
Services
Nodes
Key/Value
ACL
Intentions
Nodes
4 total
Healthy Nodes
meta-1
pg-test-1
pg-test-2
pg-test-3
Services
14 total
consul
haproxy
node-exporter
patroni
pg-exporter
pg-meta
pg-meta-primary
postgres
pg-test-1

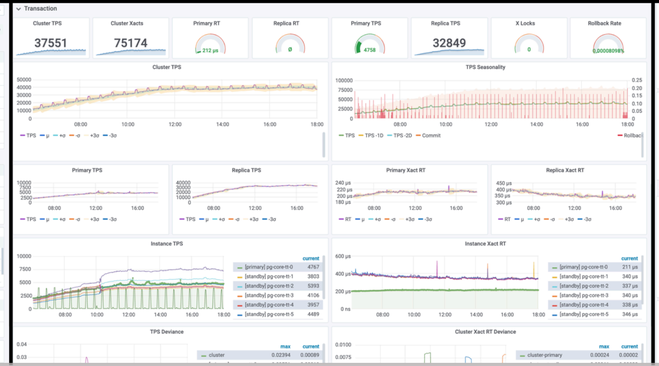

Pigsty / PG Pgbouncer
Summary
Pool
Alerts
1


Pigsty / PG Proxy
Overview
Uptime
Busy Ratio
BG Process
Task
FE Session
BE Session
Service Status
Server Status
Primary Service Status
Replica Service Status
Primary Server Status
Replica Servers Status
General Info

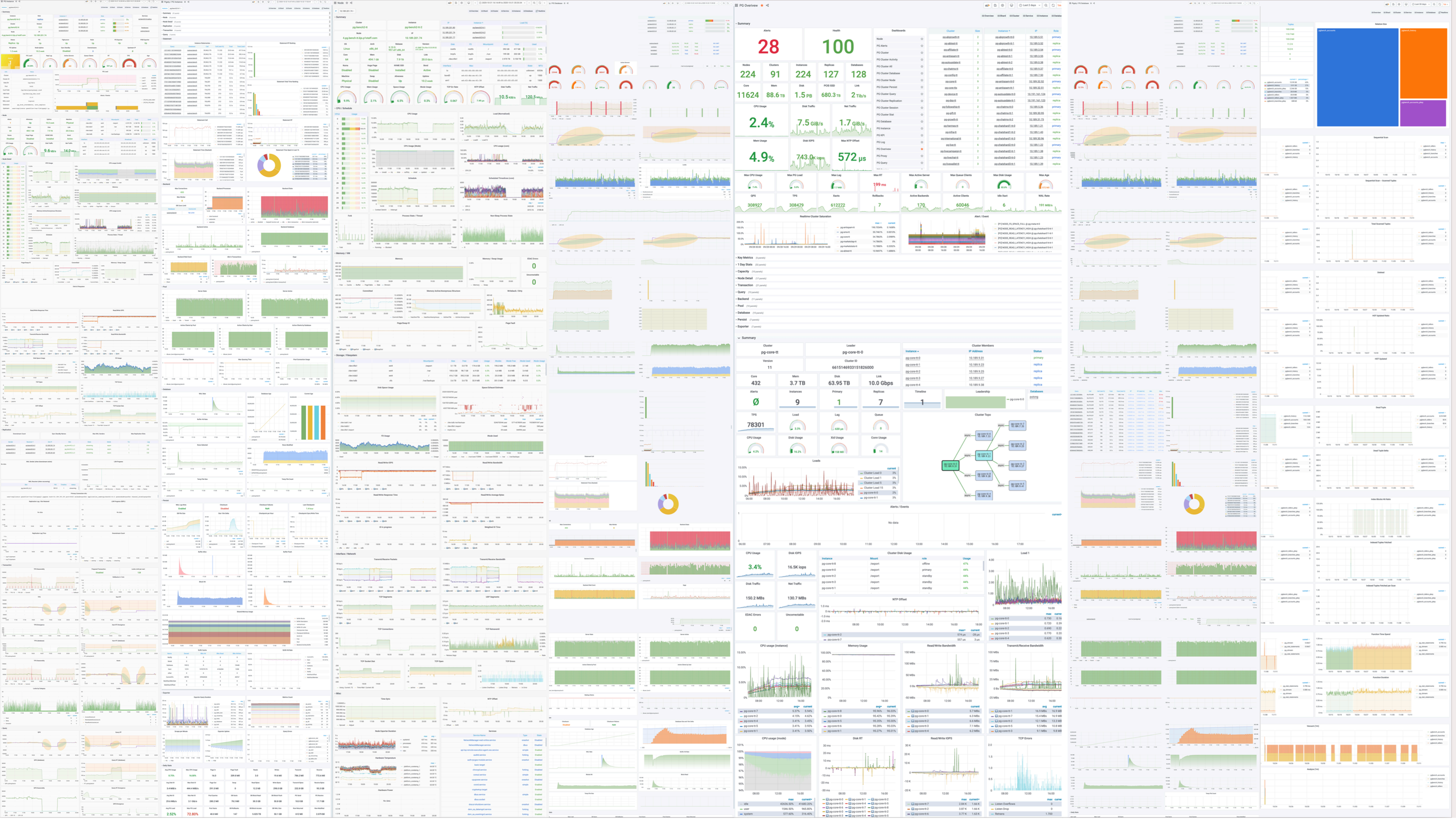

Rows Selected
Rows Modified
Temp File Size
Temp File Count
Persist
WAL Log Hints
Disabled
Checksum
Disabled
Checksum Failures
Last Checkpoint
54.2 min
DB File Size
Size 10m Delta
Checkpoint per Hour
Checkpoint Sync/Write Time
Buffer Alloc
Buffer Flush


CPU / Schedule
CPU Usage
Load (Normalized)
CPU Usage (Mode)
CPU Usage (core)
Schedule
Scheduled Timeslices (core)
Fork
Process State / Thread
Non-Sleep Process State
Memory / VM
Memory
Memory / Swap Usage
EDAC Errors


Pigsty / PG Cluster Replication
Identity
Cluster
pg-bench2-tt
Leader
pg-bench2-tb-1
Main LB
Members
4
Replication Topo
Primary Connection Info
Instances
4
Primary
1
Replicas
3
Max Senders
16
Instances
4
WAL Senders
4
WAL Receivers
3
Cascading
WAL Source (Primary/Bridge)
Sync Commit Mode
Disabled
Sync Standby Names
null


Transaction
Cluster TPS
37551
Cluster Xacts
75174
Primary RT
Replica RT
Primary TPS
4758
Replica TPS
32849
X Locks
Rollback Rate
Cluster TPS
TPS Seasonality
Primary TPS
Replica TPS
Primary Xact RT
Replica Xact RT
Instance TPS
Instance Xact RT
TPS Deviance
Cluster Xact RT Deviance


Backend Active
Backend Database
Backend Wait Event
Idle In Transactions
Sage
Pool
Server State
Server Active
Active Clients by Pool
Active Clients by User
Active Clients by Database



界面

了解Pigsty提供的图形化用户界面

Pigsty提供了专业且易用的PostgreSQL监控系统，浓缩了业界监控的最佳实践。您可以方便地进行修改与定制；复用监控基础设施，或与其他监控系统相集成。下表是每个监控面板介绍页面的快速导航连接。

全局	集群	服务	实例	数据库
Home	**PG Cluster**	**PG Service**	**PG Instance**	**PG Database**
PG Overview	**PG Cluster Replication**	PG DNS	**Node**	**PG Query**
PG Shard	PG Cluster Activity		PG Pgbouncer	**PG Catalog**
PG Alert	PG Cluster Session		PG Proxy	**PG Table**
PG KPI	PG Cluster Node		PG Exporter	**PG Table Detail**
PG Capacity	PG Cluster Persist		PG Setting	
PG Change	PG Cluster Database		PG Stat Activity	
PG Monitor	PG Cluster Stats		PG Stat Statements	
	PG Cluster Table			
	PG Cluster Table Detail			
	PG Cluster Query			
	PG Cluster Health			
	PG Cluster Log			
	PG Cluster All			

全局监控

- Home
- PG Overview
- PG Shard
- PG Alert
- PG KPI
- PG Capacity
- PG Change
- PG Monitor

集群监控

- PG Cluster
- PG Cluster Replication
- PG Cluster Activity
- PG Cluster Session
- PG Cluster Node
- PG Cluster Persist
- PG Cluster Database
- PG Cluster Stat
- PG Cluster Table
- PG Cluster Table Detail
- PG Cluster Query
- PG Cluster Health
- PG Cluster Log
- PG Cluster All

服务监控

- PG Service
- PG DNS

实例监控

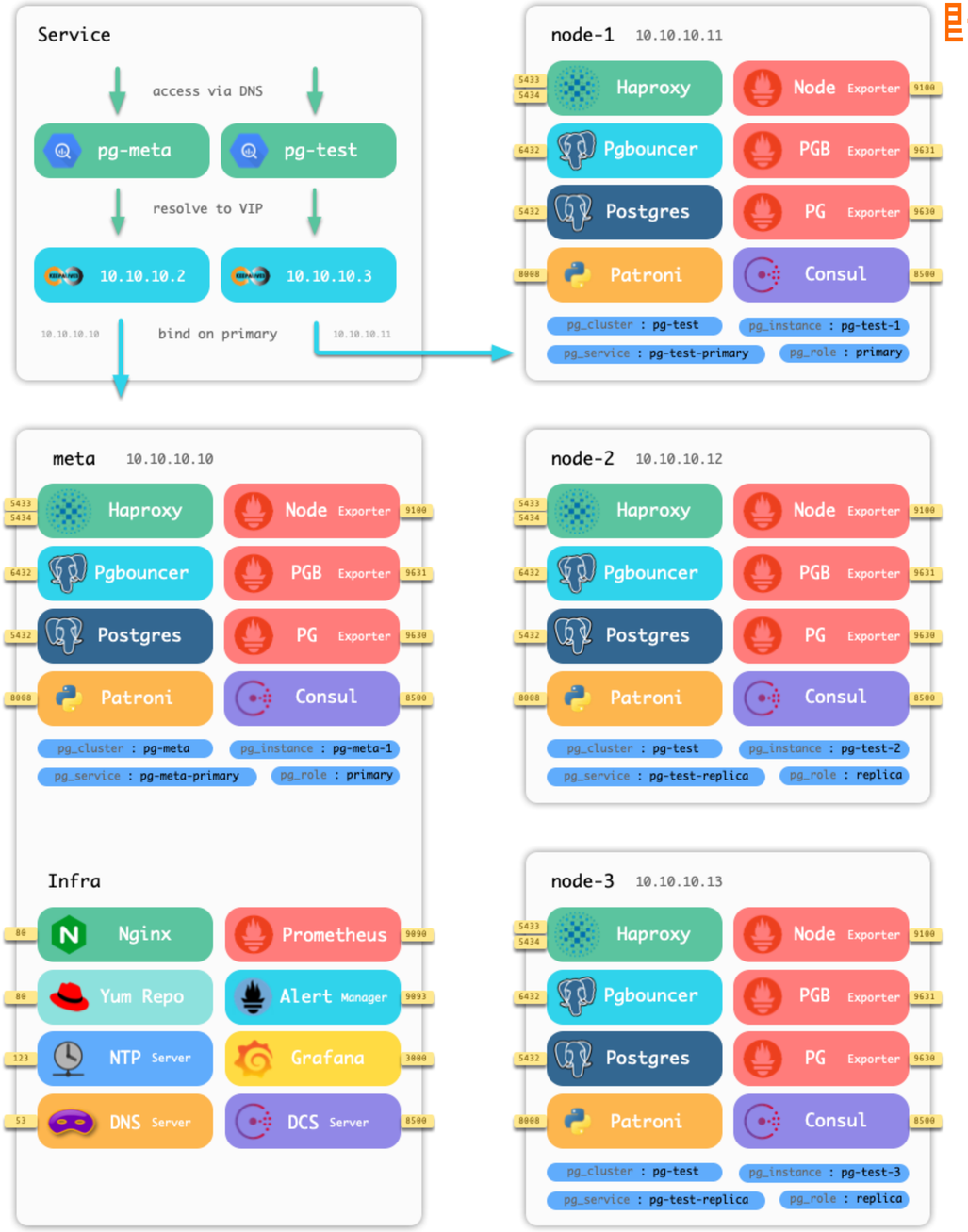
- PG Instance
- Node
- PG Pgbouncer
- PG Proxy
- PG Exporter
- PG Setting
- PG Stat Activity
- PG Stat Statements

数据库监控

- PG Database
- PG Pool
- PG Query
- PG Table Catalog
- PG Table
- PG Table Detail
- PG Query Detail

如何拥有？

Pigsty提供了沙箱环境
在您的笔记本上
迅速拉起PostgreSQL集群
与Pigsty监控系统



上手

基于vagrant，快速在本机拉起演示系统

这篇文档将介绍如何在您手头的**笔记本或PC机**上，基于Vagrant与Virtualbox，一键拉起Pigsty演示沙箱。

本教程着眼于在本地单机创建Pigsty演示环境，如果您已经有可以用于部署的机器实例，可以参考部署教程

太长；不看

如果您的本地计算机上已经安装有vagrant， virtualbox与ansible，那么只需要克隆并进入本项目后，依次执行以下命令即可：

```
make up          # 拉起vagrant虚拟机
make ssh         # 配置虚拟机ssh访问
make init        # 初始化Pigsty
sudo make dns    # 写入Pigsty静态DNS域名（需要sudo，可选）
make mon-view    # 打开Pigsty首页（默认用户密码：admin:admin）
```

正常情况下执行结果详见 参考-标准流程。

如果您希望了解详情，知晓每条命令背后完成的工作，请往下阅读。

欢迎来到Pigsty中文文档

> 【DRAFT】Pigsty English Documentation
>
> Pigsty是 PostgreSQL In Graphic STYle 的缩写，即 “图形化Postgres”。
>
> pigsty 一词的本意是猪圈，读作 Pig Style (/ˈpɪgˌstaɪ/) 。

Pigsty

Pigsty 针对大规模数据库集群监控与管理而设计，提供业界顶尖的监控系统与开箱即用的高可用数据库供给方案。Pigsty基于开源生态构建，旨在降低PostgreSQL使用管理的门槛，为用户带来极致的可观测性与丝滑的数据库使用体验。

Pigsty经过长期迭代演进，久经实际生产环境考验，可免费用于测试生产。如需专业支持，Pigsty亦提供可选商业支持与企业版订阅方案。

概览
快速了解Pigsty所解决的问题，采用的技术，适用的场景。

上手
基于vagrant，快速在本机拉起演示系统

概念
在使用Pigsty时需要了解的一些信息

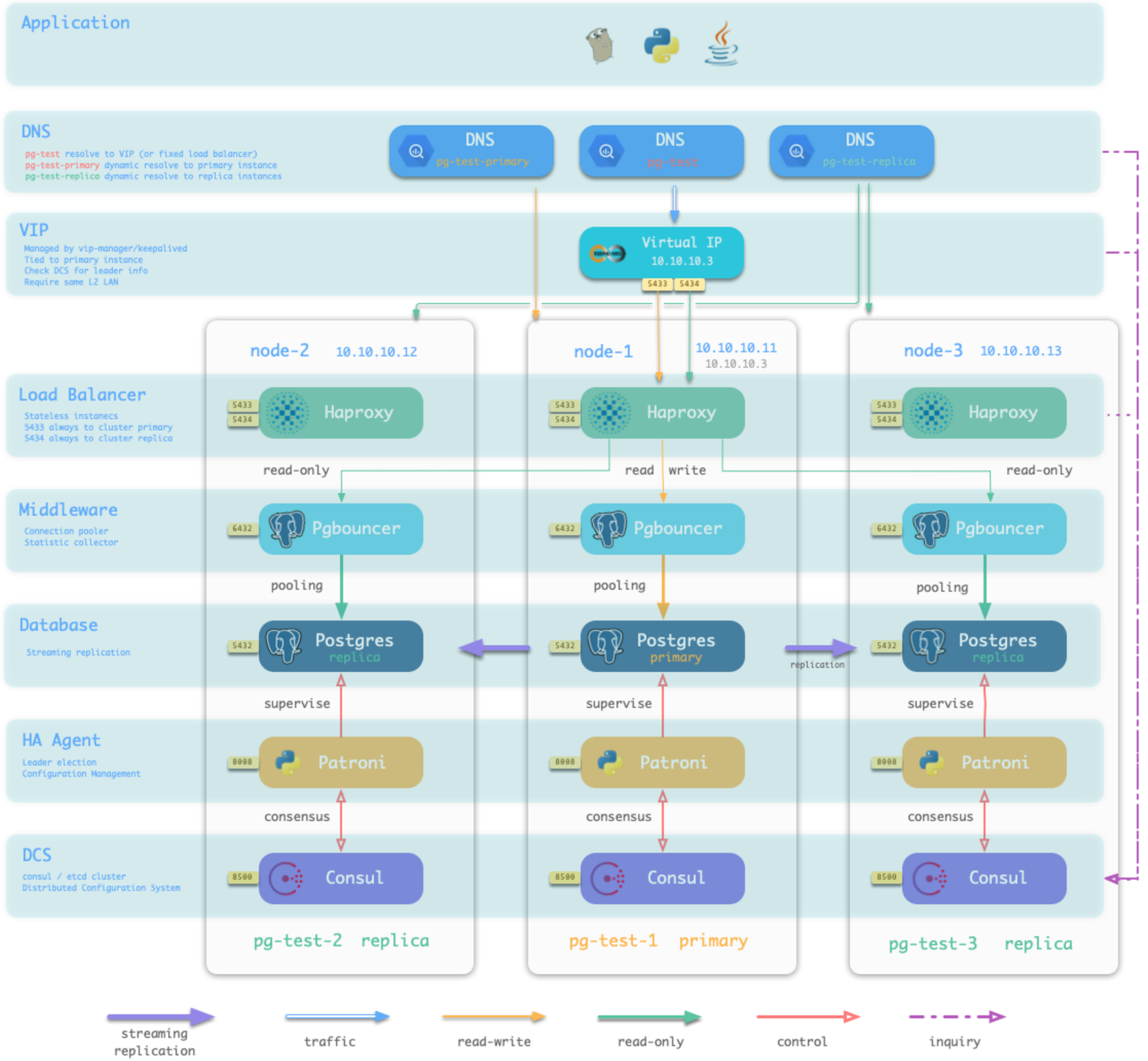
界面
了解Pigsty提供的图形化用户界面

教程
如何配置与定制Pigsty

任务
高可用演练，数据库试用，一些可以在Pigsty中探索的任务

示例
在不同的环境中部署Pigsty

参考
Pigsty配置项细节参考



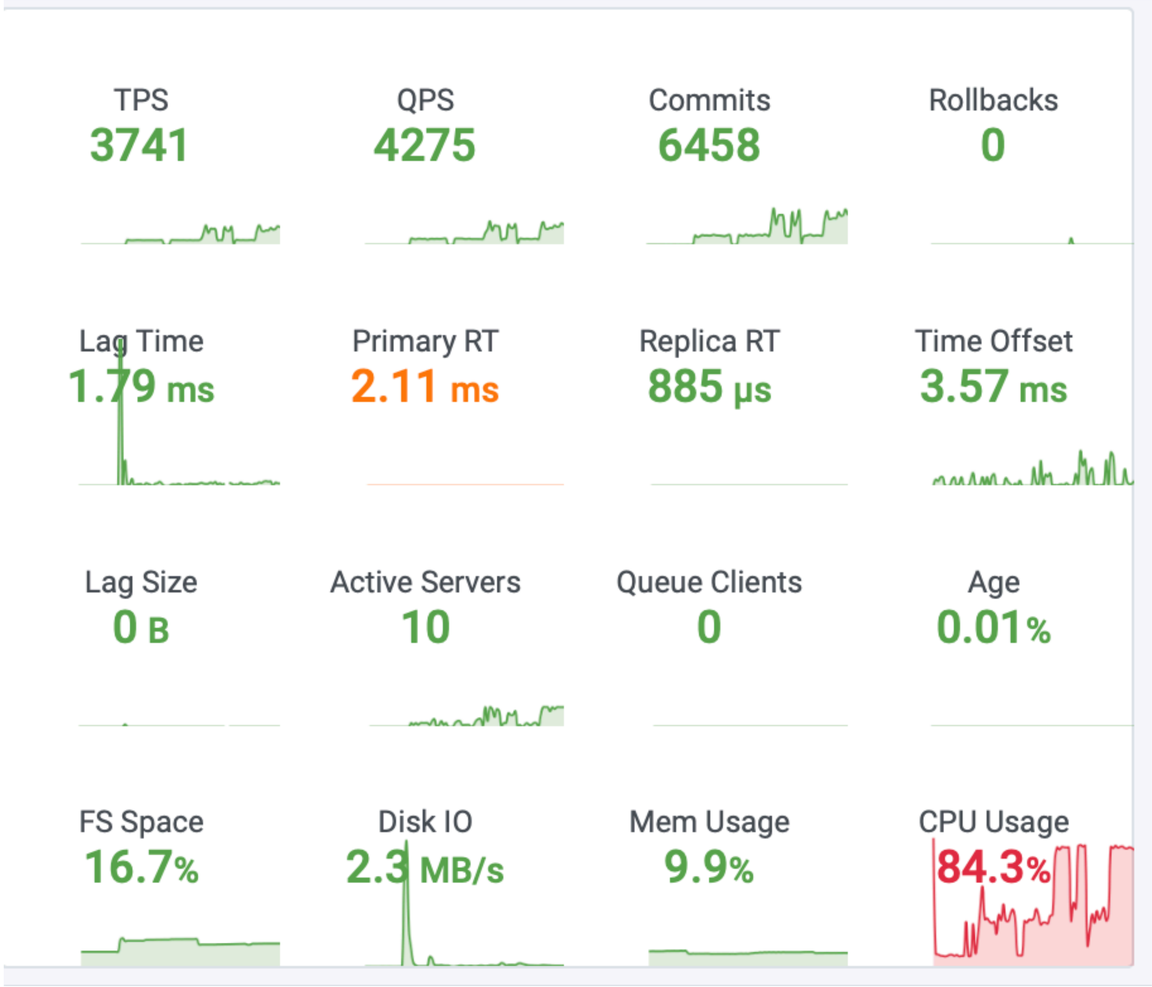
03 解决问题

如何使用监控系统解决现实问题？

常用指标

指标	缩写	层次	来源	种类
数据库负载	PG Load	DB	连接池	饱和度
数据库饱和度	PG Saturation	DB	连接池/节点	饱和度
主从复制延迟	Repl Lag	DB	数据库	延迟
平均查询响应时间	Query RT	DB	连接池	延迟
活跃后端进程数	Backends	DB	数据库	饱和度
数据库年龄	Age	DB	数据库	饱和度
CPU使用率	CPU Usage	SYS	机器节点	饱和度
每秒查询数	QPS	**APP**	连接池	流量
连接池排队数	Queue Clients	DB	连接池	错误
错误日志条数	Error Count	SYS/DB/APP	日志系统	错误

Pigsty 提供了约1200 个指标，但最重要的便是这10个



核心指标 PG Load　　衡量数据库的负载程度

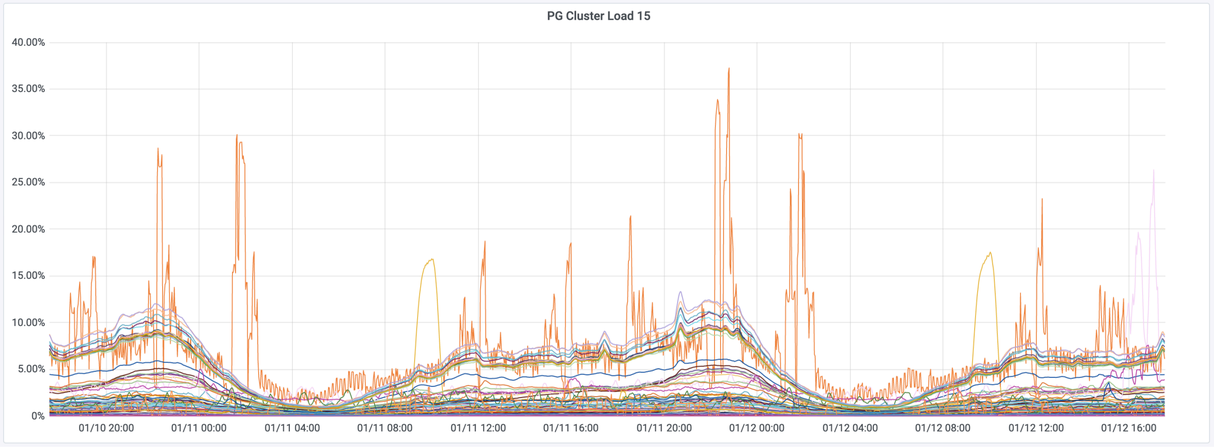
归一化指标，**PG活跃状态事务时长占CPU总可用时间之比**

```
PG Load = TPS x AverageTransactionResponseTime / CPU Count
```

```
  - record: pg:ins:xact_time_realtime
    expr: sum without (datname) (irate(pgbouncer_stat_total_xact_time{}[1m]))
  - record: pg:ins:load0
    expr:  pg:ins:xact_time_realtime / on (ip) group_left()  node:ins:cpu_count
```

http://pigsty.cc/zh/blog/2020/05/29/postgresql%E7%9A%84kpi/

0%　　零负载
40%　　中负载（黄线）
70%　　高负载（红线）
100%+ 满载　　（死线）



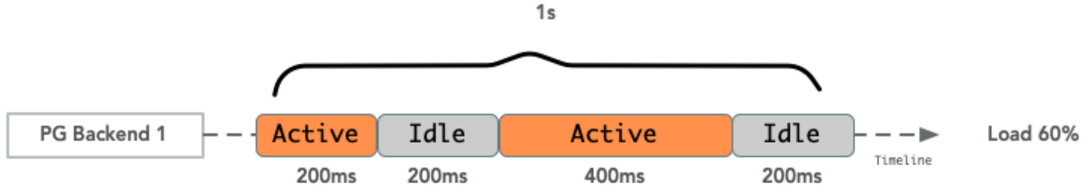


Fig.1 Single Process PG Load

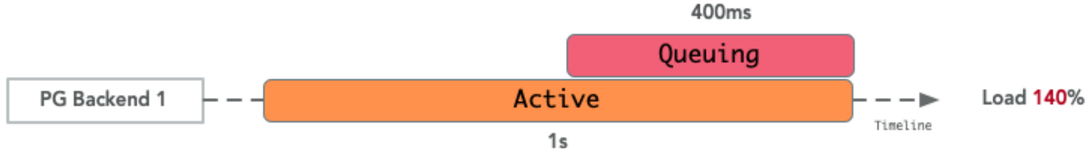


Fig.2 Load Overflow

Overall Load :
(200 + 400 + 800) / 2000 = 70%

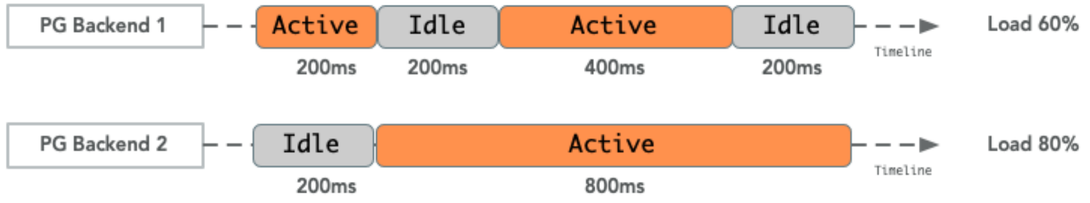


Fig.3 Parallel PG Load

Fig.4 Xact or Query

核心指标 PG Saturation

衡量数据库的整体饱和度

阿里云

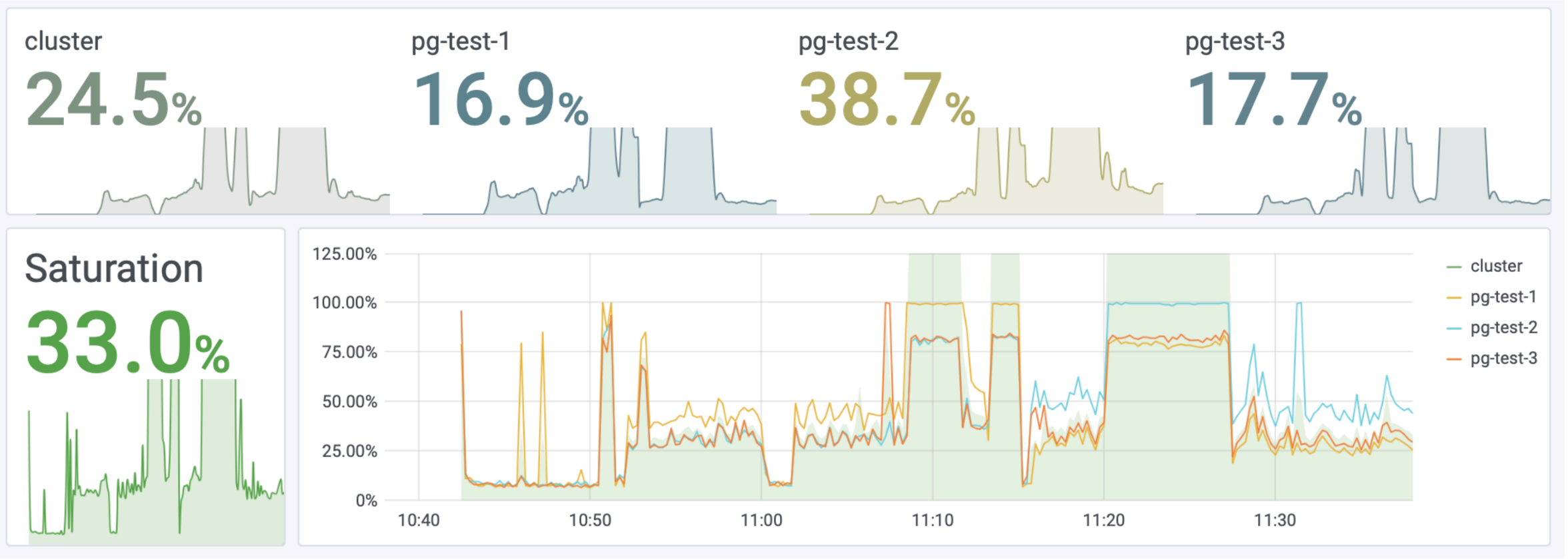
饱和度用于反映数据库整体资源利用率
理论上应当取所有饱和度指标的最大值（PG，CPU，内存，网络，磁盘……）

```
PG Saturation = max(PG Load, CPU Usage, xxxUsage…)
```

实际应用中，饱和度指标取 PG Load与 CPU Usage的最大值作为饱和度

当数据库使用非独占式部署，其他应用占用CPU资源时，
PG Saturation比单纯的PG Load更能反映数据库整体负载水位

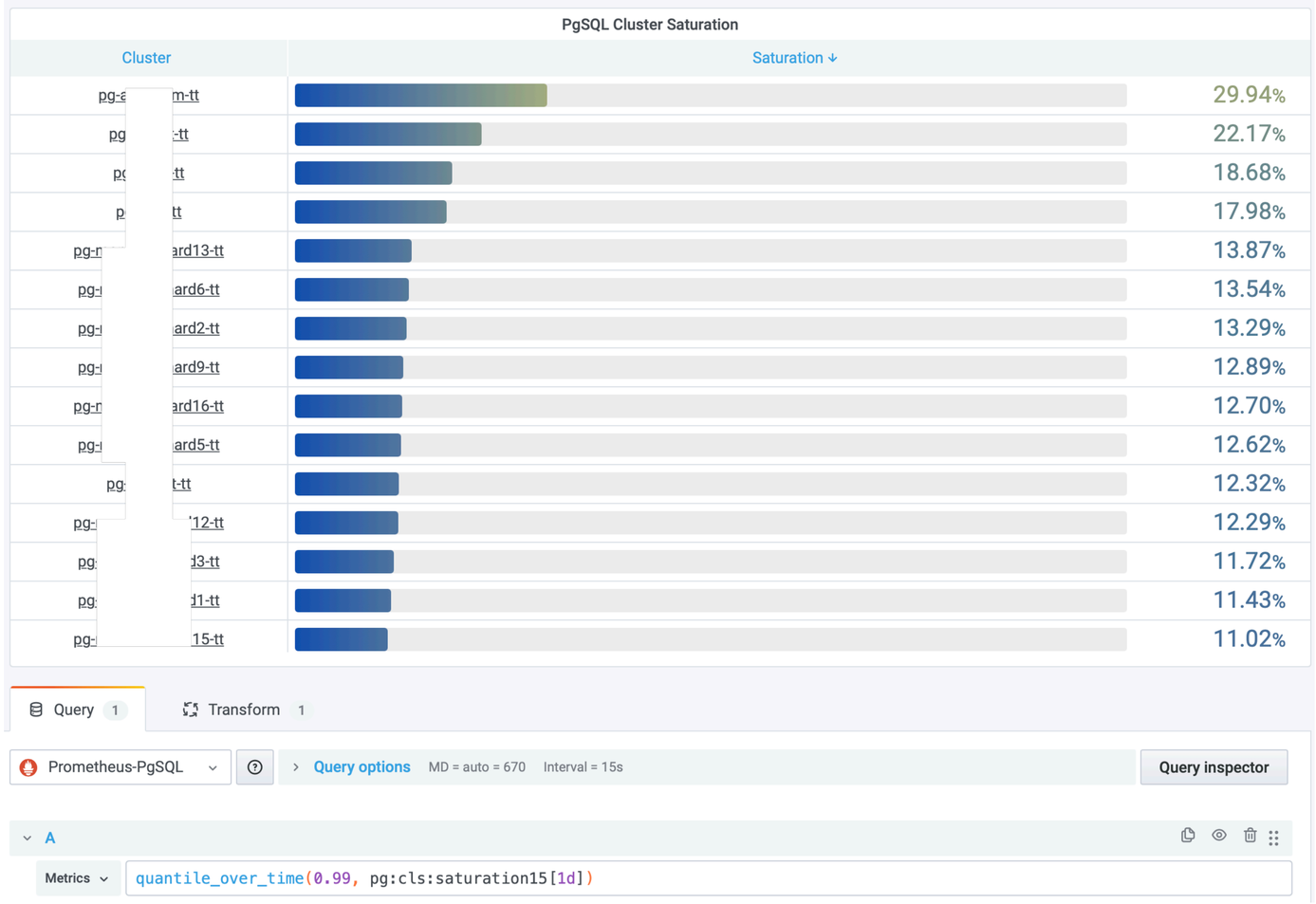
```
- record: pg:ins:saturation0
  expr: pg:ins:load0 > node:ins:cpu_usage or node:ins:cpu_usage
- record: pg:ins:saturation1
  expr: pg:ins:load1 > node:ins:cpu_usage or node:ins:cpu_usage
- record: pg:ins:saturation5
  expr: pg:ins:load5 > node:ins:cpu_usage or node:ins:cpu_usage
- record: pg:ins:saturation15
  expr: pg:ins:load15 > node:ins:cpu_usage or node:ins:cpu_usage
```



水位评估： 饱和度的历史分位点

过去一天中饱和度的99分位点

```
quantile_over_time(0.99, pg:cls:saturation15[1d])
```



系统在99%的时间内
整体资源使用水平低于该值

根据您的具体场景与需求修改参数
例如：

过去一天 ➡️ 过去15天
99分位点 ➡️ 99.99分位点

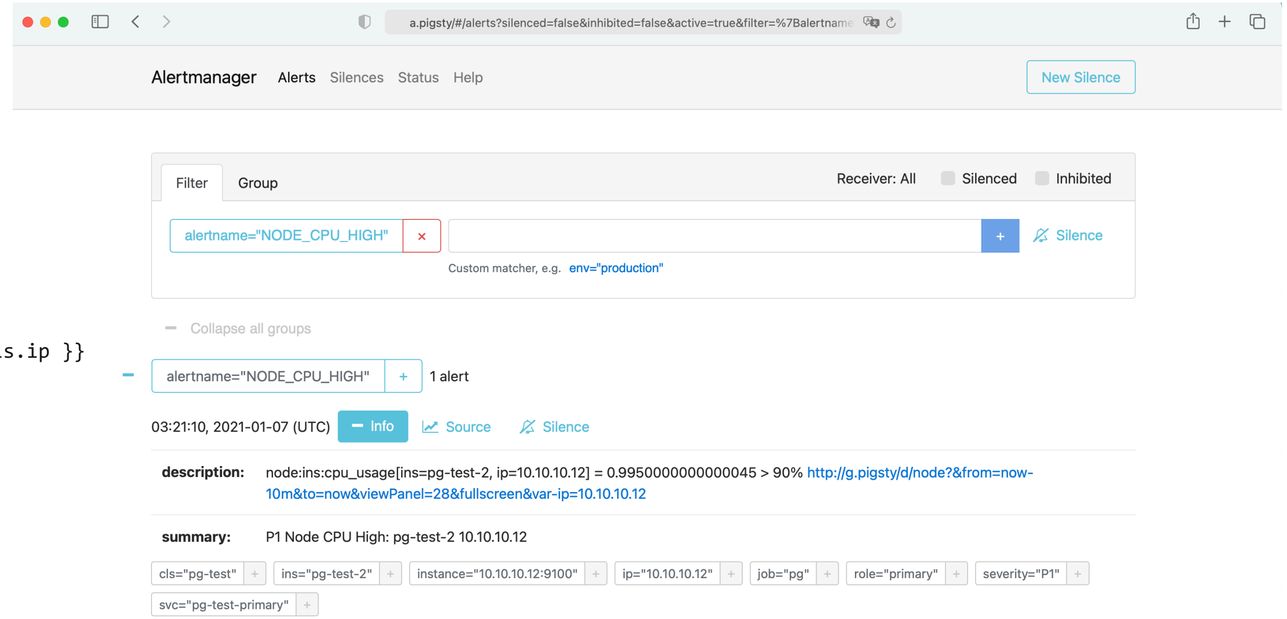
用于水位评估
作为扩缩容，资源分配的依据

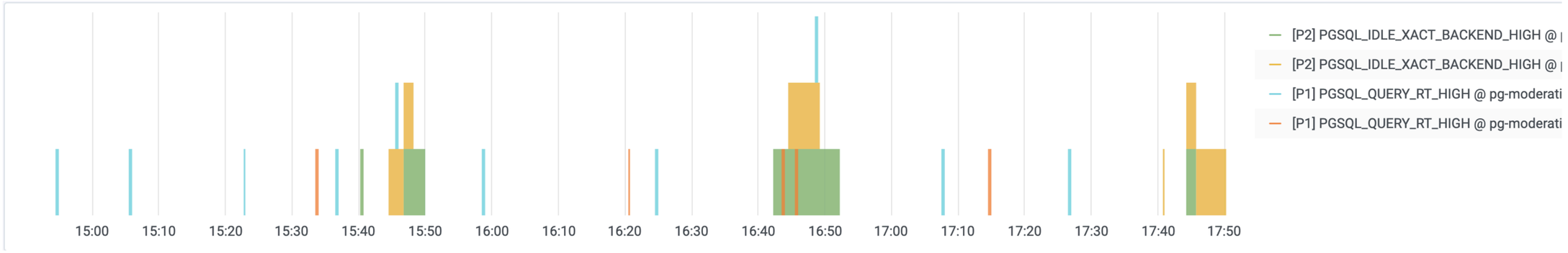
报警通知

使用报警系统自动跟踪指标异常

```yaml
# node avg CPU usage > 90% for 1m
- alert: NODE_CPU_HIGH
  expr: node:ins:cpu_usage > 0.90
  for: 1m
  labels:
    severity: P1
  annotations:
    summary: "P1 Node CPU High: {{ $labels.ins }} {{ $labels.ip }}"
    description: |
      node:ins:cpu_usage[ins={{ $labels.ins }}, ip={{ $labels.ip }}] = {{ $value }} > 90%
      http://g.pigsty/d/node?&from=now-10m&to=now&viewPanel=28&fullscreen&var-ip={{ $labels.ip }}


# pg load1 high than 70% for 3m triggers a P1 alert
- alert: PGSQL_LOAD_HIGH
  expr: pg:ins:load1{} > 0.70
  for: 3m
  labels:
    severity: P1
  annotations:
    summary: "P1 PG Load High: {{ $labels.ins }} {{ $value }}"
    description: |
      pg:ins:load1[ins={{ $labels.ins }}] = {{ $value }} > 70%
      http://g.pigsty/d/pg-instance?from=now-10m&to=now&viewPanel=210&fullscreen&var-ins={{ $labels.ins }}
```





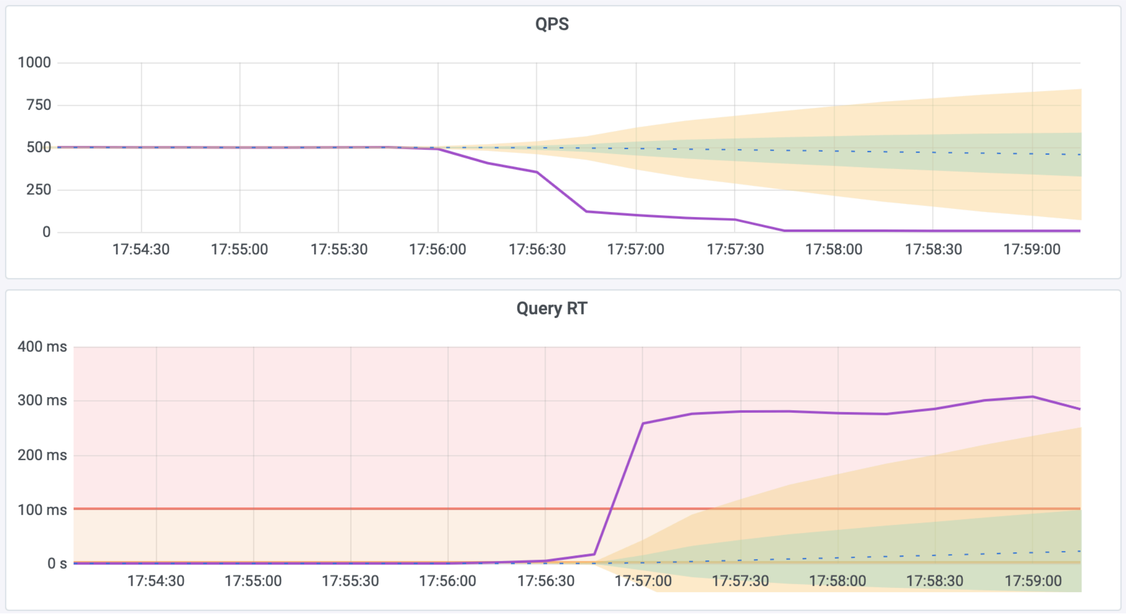
慢查询：模拟

慢查询是数据库的大敌，这里我们使用 pgbench 用例模拟一个慢查询

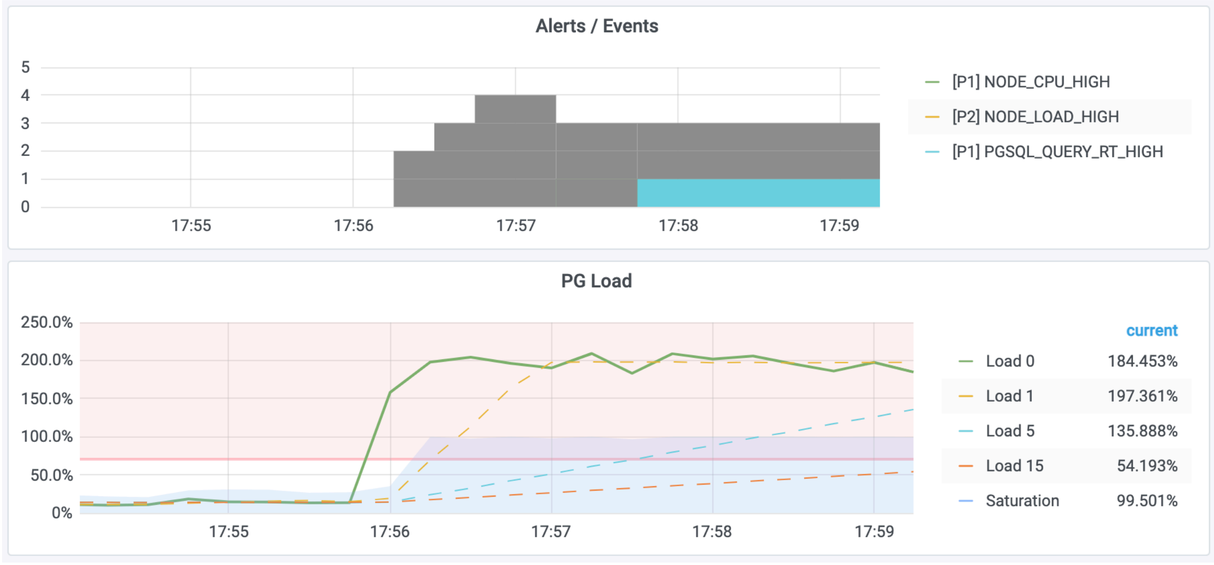
```
ALTER TABLE pgbench_accounts DROP CONSTRAINT pgbench_accounts_pkey ;
```

该命令会移除 pgbench_accounts 表上的主键，导致相关查询变慢，系统瞬间雪崩过载。

单个从库实例的QPS从500下降至7，Query RT下降至300ms



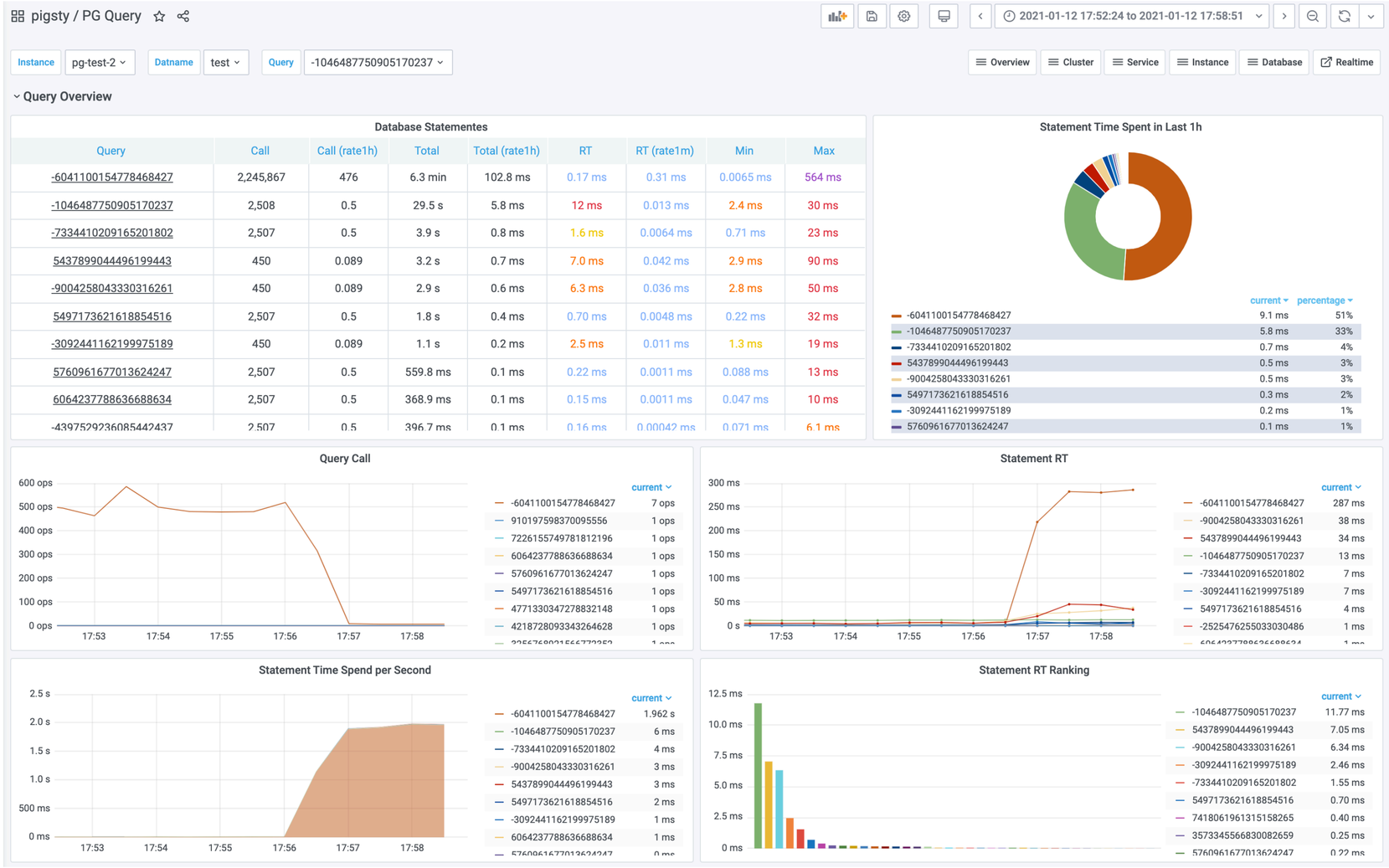
系统负载达到200%，触发机器负载过大，与查询响应时间过长的报警规则。



慢查询：定位

首先，使用PG Cluster面板定位慢查询所在的具体实例，这里以 pg-test-2为例

然后，使用PG Query面板定位具体的慢查询：编号为 **-6041100154778468427**



该查询表现出：

响应时间显著上升： **17us** 升至 **280ms**
QPS 显著下降： 从**500**下降到 **7**
花费在该查询上的时间占比**显著增加**

可以确定：

就是这个查询变慢了！

慢查询：猜想

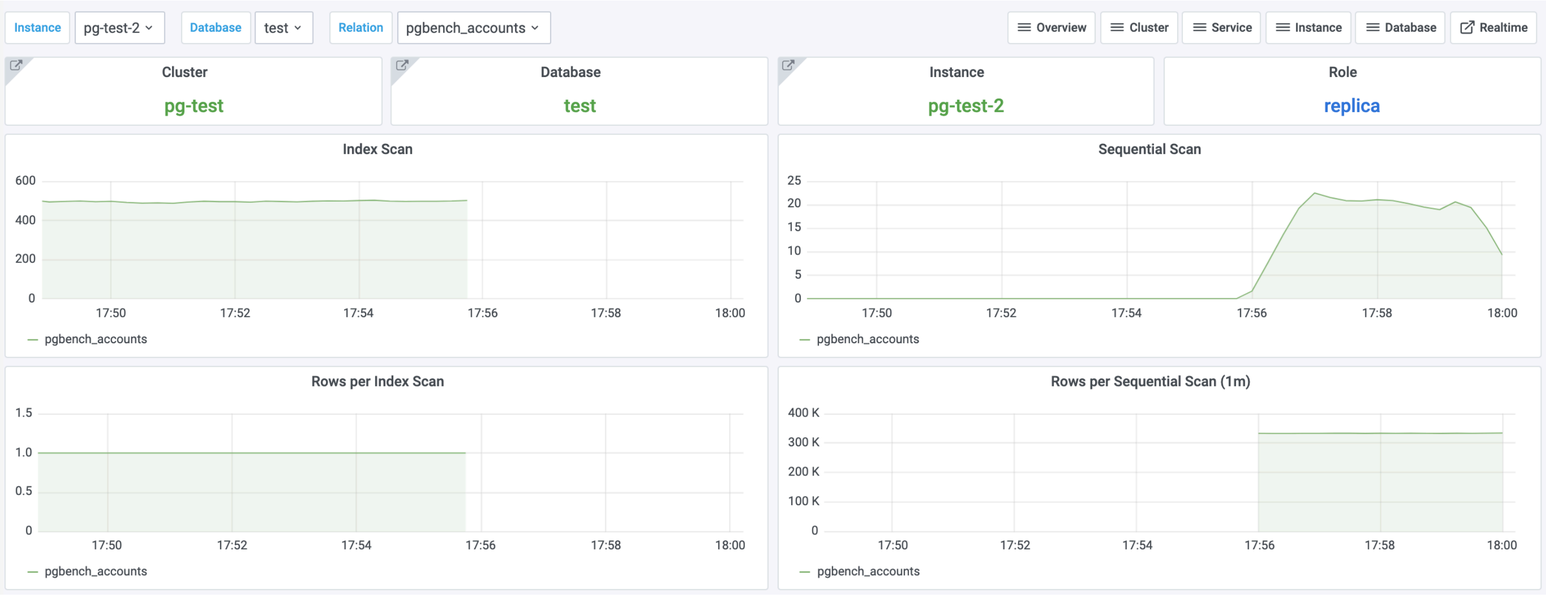
接下来，利用PG Stat Statements面板，根据查询ID定位慢查询的具体语句。

-6041100154778468427	test	19353	SELECT abalance FROM pgbench_accounts WHERE aid = $1
-2525476255033030486	test	16385	SE
2951481090153650440	test	16385	SE

```
"SELECT abalance FROM pgbench_accounts WHERE aid = $1"
```

分析查询后**提出猜想**： 该查询变慢是pgbench_accounts表上aid列缺少索引

下一步，查阅 PG Table Detail面板，检查 pgbench_accounts 表上的访问



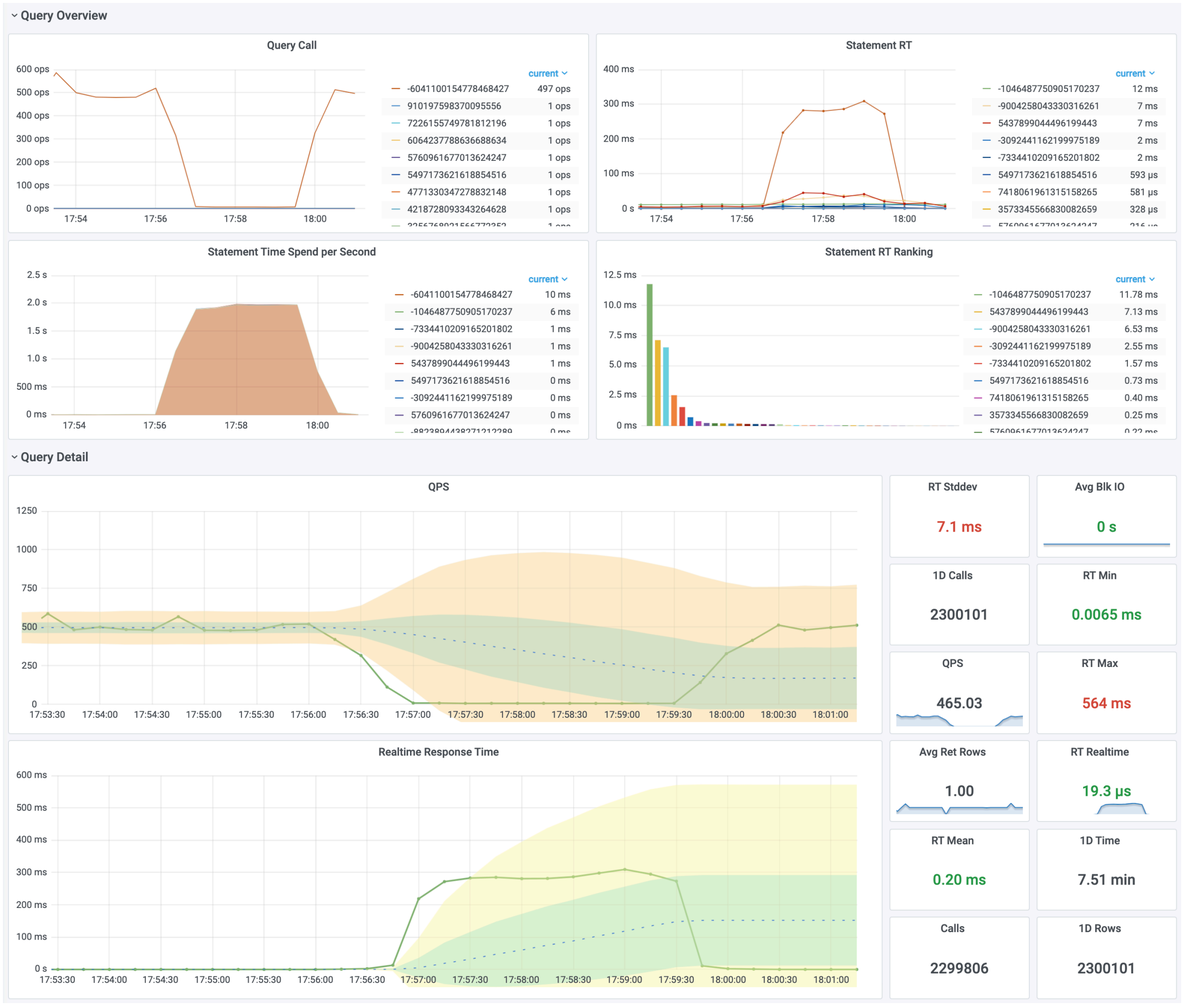
通过观察，我们发现表上的**索引扫描**归零，与此同时**顺序扫描**却有相应增长。这印证了我们的猜想！

查询以 aid 作为过滤条件查询 pgbench_accounts 表查询变慢，大概率是这张表上的索引出了问题。

慢查询：处理

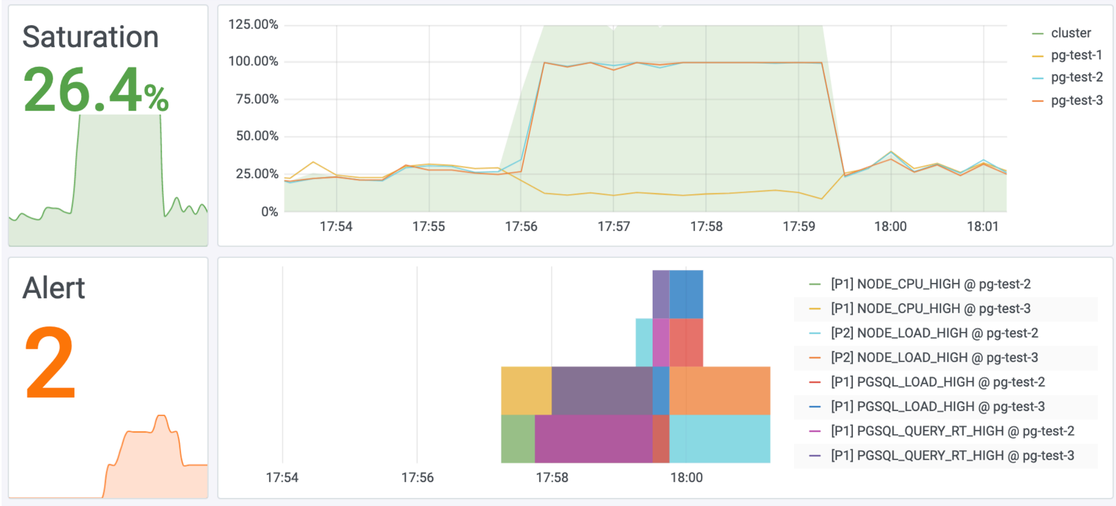
我们尝试在 pgbench_accounts 表上为 aid 列添加索引，看看能否解决这个问题。

```
CREATE UNIQUE INDEX ON pgbench_accounts (aid);
```



可以看到，查询的响应时间与QPS已经恢复正常。

整个集群的负载水平也恢复正常，报警也平息下来。



慢查询：评估

精准衡量优化效果，直观展示工作成绩，真正做到数据驱动。