

# pgnodemx

## Overview

SQL functions that allow capture of node OS metrics from PostgreSQL

## Security

Executing role must have been granted pg\_monitor membership (pgmonitor for PostgreSQL version 9.6 and below - see Compatibility section below).

## cgroup Related Functions

For detailed information about the various virtual files available on the cgroup file system, see: \* cgroup v1: <https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/index.html> \* cgroup v2: <https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html>

## General Access Functions

cgroup virtual files fall into (at least) the following general categories, each with a generic SQL access function:

- BIGINT single line scalar values - `SELECT cgroup_scalar_bigint(filename);`
- cgroup v1 examples: `blkio.leaf_weight`, `blkio.weight`, `cpuacct.usage`, `cpuacct.usage_percpu`, `cpuacct.usage_percpu_sys`, `cpuacct.usage_percpu_user`, `cpuacct.usage_sys`, `cpuacct.usage_user`, `cpu.cfs_period_us`, `cpu.cfs_quota_us`, `cpu.rt_period_us`, `cpu.rt_runtime_us`, `cpu.shares`, `cpuacct.usage`, `memory.failcnt`, `memory.kmem.failcnt`, `memory.kmem.limit_in_bytes`, `memory.kmem.max_usage_in_bytes`, `memory.kmem.tcp.failcnt`, `memory.kmem.tcp.limit_in_bytes`, `memory.kmem.tcp.max_usage_in_bytes`, `memory.kmem.tcp.usage_in_bytes`, `memory.kmem.usage_in_bytes`, `memory.limit_in_bytes`, `memory.max_usage_in_bytes`, `memory.memsw.failcnt`, `memory.memsw.limit_in_bytes`, `memory.memsw.max_usage_in_bytes`, `memory.memsw.usage_in_bytes`, `memory.move_charge_at_immigrate`, `memory.soft_limit_in_bytes`, `memory.usage_in_bytes`, `net_cls.classid`, `net_prio.prioidx`
- cgroup v2 examples: `cgroup.freeze`, `cgroup.max.depth`, `cgroup.max.descendants`, `cpu.weight`, `cpu.weight.nice`, `memory.current`, `memory.high`, `memory.low`, `memory.max`, `memory.min`, `memory.oom.group`, `memory.swap.current`, `memory.swap.max`, `pids.current`, `pids.max`

- FLOAT8 single line scalar values - `SELECT cgroup_scalar_float8(filename);`
- cgroup v1 examples: (none known)
- cgroup v2 examples: `cpu.uclamp.max`, `cpu.uclamp.min`
- TEXT single line scalar values - `SELECT cgroup_scalar_text(filename);`
- cgroup v1 examples: (none known)
- cgroup v2 examples: `cgroup.type`
- SETOF(BIGINT) multiline scalar values - `SELECT * FROM cgroup_setof_bigint(filename);`
- cgroup v1 examples: `cgroup.procs`
- cgroup v2 examples: `cgroup.procs`, `cgroup.threads`
- SETOF(TEXT) multiline scalar values - `SELECT * FROM cgroup_setof_text(filename);`
- cgroup v1 examples: (none known)
- cgroup v2 examples: (none known)
- ARRAY[BIGINT] space separated values - `SELECT cgroup_array_bigint(filename);`
- cgroup v1 examples: (none known)
- cgroup v2 examples: `cpu.max`
- ARRAY[TEXT] space separated values - `SELECT cgroup_array_text(filename)`
- cgroup v1 examples: `cpuacct.usage_all` (sort of)
- cgroup v2 examples: `cgroup.controllers`, `cgroup.subtree_control`
- SETOF(TEXT, BIGINT) flat keyed - `SELECT * FROM cgroup_setof_kv(filename);`
- cgroup v1 examples: `cpuacct.stat`, `cpu.stat`, `cpuacct.stat`, `memory.oom_control`, `memory.stat`, `net_prio.ifpriomap`, `blkio.io_merged`, `blkio.io_merged_recursive`, `blkio.io_queued`, `blkio.io_queued_recursive`, `blkio.io_service_bytes`, `blkio.io_service_bytes_recursive`, `blkio.io_serviced`, `blkio.io_serviced_recursive`, `blkio.io_service_time`, `blkio.io_service_time_recursive`, `blkio.io_wait_time`, `blkio.io_wait_time_recursive`
- cgroup v2 examples: `cgroup.events`, `cgroup.stat`, `cpu.stat`, `io.pressure`, `io.weight`, `memory.events`, `memory.events.local`, `memory.stat`, `memory.swap.events`, `pids.events`
- SETOF(TEXT, TEXT, BIGINT) key/subkey/value space separated - `SELECT * FROM cgroup_setof_ksv(filename);`
- cgroup v1 examples: `blkio.throttle.io_service_bytes`, `blkio.throttle.io_serviced`

- cgroup v2 examples: (none known)
- SETOF(TEXT, TEXT, FLOAT8) nested keyed - `SELECT * FROM cgroup_setof_nkv(filename);`
- cgroup v1 examples: (none known)
- cgroup v2 examples: `memory.pressure`, `cpu.pressure`, `io.max`, `io.stat`

In each case, the filename must be in the form `<controller>.<metric>`, e.g. `memory.stat`.

### Get status of cgroup support

```
SELECT current_setting('pgnodemx.cgroup_enabled');
```

- Returns boolean result (“on”/“off”).
- This value may be explicitly set in `postgresql.conf`
- However the extension will disable it at runtime if the location pointed to by `pgnodemx.cgrouproot` does not exist or is not a valid cgroup v1 or v2 mount.

### Get current cgroup mode

```
SELECT cgroup_mode();
```

- Returns the current cgroup mode. Possible values are “legacy”, “unified”, “hybrid”, and “disabled”. These correspond to cgroup v1, cgroup v2, mixed, and disabled, respectively.
- Currently “hybrid” mode is not supported; it might be in the future.

### Determine if Running Containerized

```
SELECT current_setting('pgnodemx.containerized');
```

- Returns boolean result (“on”/“off”). The extension attempts to heuristically determine whether PostgreSQL is running under a container, but this value may be explicitly set in `postgresql.conf` to override the heuristically determined value. The value of this setting influences the cgroup paths which are used to read the cgroup controller files.

### Get cgroup Paths

```
SELECT controller, path FROM cgroup_path();
```

- Returns the path to each supported cgroup controller.

### Get cgroup process count

```
SELECT cgroup_process_count();
```

- Returns the number of processes assigned to the cgroup
- For cgroup v1, based on the “memory” controller cgroup.procs file. For cgroup v2, based on the unified cgroup.procs file.

## Environment Variable Related Functions

### Get Environment Variable as TEXT

```
SELECT envvar_text('PGDATA');
```

- Returns the value of requested environment variable as TEXT

### Get Environment Variable as BIGINT

```
SELECT envvar_bigint('PGPORT');
```

- Returns the value of requested environment variable as BIGINT

## /proc Related Functions

For more detailed information about the /proc file system virtual files, please see: <https://www.kernel.org/doc/html/latest/filesystems/proc.html>

### Get “/proc/diskstats” as a virtual table

```
SELECT * FROM proc_diskstats();
```

### Get “/proc/self/mountinfo” as a virtual table

```
SELECT * FROM proc_mountinfo();
```

Get “/proc/meminfo” as a virtual table

```
SELECT * FROM proc_meminfo();
```

Get “/proc/self/net/dev” as a virtual table

```
SELECT * FROM proc_network_stats();
```

## System Information Related Functions

Get file system information as a virtual table

```
SELECT * FROM fsinfo(path text);
```

- Returns `major_number`, `minor_number`, `type`, `block_size`, `blocks`, `total_bytes`, `free_blocks`, `free_bytes`, `available_blocks`, `available_bytes`, `total_file_nodes`, `free_file_nodes`, and `mount_flags` for the file system on which `path` is mounted.

## Kubernetes DownwardAPI Related Functions

For more detailed information about the Kubernetes DownwardAPI please see: <https://kubernetes.io/docs/tasks/inject-data-application/downward-api-volume-expose-pod-information/>

Get status of `kdapi_enabled`

```
SELECT current_setting('pgnodemx.kdapi_enabled');
```

- Returns boolean result (“on”/“off”).
- This value may be explicitly set in `postgresql.conf`
- However the extension will disable it at runtime if the location pointed to by `pgnodemx.kdapi_path` does not exist.

Access “key equals quoted value” files

```
SELECT * FROM kdapi_setof_kv('filename');
```

Get scalar `BIGINT` from file

```
SELECT kdapi_scalar_bigint('filename text');
```

## Configuration

- Add `pgnodemx` to `shared_preload_libraries` in `postgresql.conf`.

```
shared_preload_libraries = 'pgnodemx'
```

- The following custom parameters may be set. The values shown are defaults. If the default values work for you, there is no need to add these to `postgresql.conf`.

```
# enable or disable the cgroup facility
pgnodemx.cgroup_enabled = on
# force use of "containerized" assumptions for cgroup file paths
pgnodemx.containerized = off
# specify location of cgroup mount
pgnodemx.cgrouproot = '/sys/fs/cgroup'
# enable cgroup functions
pgnodemx.cgroup_enabled = on
# enable or disable the Kubernetes DownwardAPI facility
pgnodemx.kdapi_enabled = on
# specify location of Kubernetes DownwardAPI files
pgnodemx.kdapi_path = '/etc/podinfo'
```

Notes:

- If `pgnodemx.cgroup_enabled` is defined in `postgresql.conf`, and set to `off` (or `false`), then all `cgroup*` functions will return `NULL`, or zero rows, except `cgroup_mode()` which will return “disabled”.
- If `pgnodemx.containerized` is defined in `postgresql.conf`, that value will override `pgnodemx` heuristics. When not specified, `pgnodemx` heuristics will determine if the value should be `on` or `off` at runtime.
- If the location specified by `pgnodemx.cgrouproot`, default or as set in `postgresql.conf`, is not accessible (does not exist, or otherwise causes an error when accessed), then `pgnodemx.cgroup_enabled` is forced to `off` at runtime and all `cgroup*` functions will return `NULL`, or zero rows, except `cgroup_mode()` which will return “disabled”.
- If the location specified by `pgnodemx.kdapi_path`, default or as set in `postgresql.conf`, is not accessible (does not exist, or otherwise causes an error when accessed), then `pgnodemx.kdapi_enabled` is forced to `off` at runtime and all `kdapi*` functions will return `NULL`, or zero rows.

## Installation

### Compatibility

- PostgreSQL version 9.5 or newer is required.
- On PostgreSQL version 9.6 or earlier, a role called pgmonitor must be created, and the user calling these functions must be granted that role.

### Compile and Install

Clone PostgreSQL repository:

```
$> git clone https://github.com/postgres/postgres.git
```

Checkout REL\_12\_STABLE (for example) branch:

```
$> git checkout REL_12_STABLE
```

Make PostgreSQL:

```
$> ./configure  
$> make install -s
```

Change to the contrib directory:

```
$> cd contrib
```

Clone pgnodemx extension:

```
$> git clone https://github.com/crunchydata/pgnodemx
```

Change to pgnodemx directory:

```
$> cd pgnodemx
```

Build pgnodemx:

```
$> make
```

Install pgnodemx:

```
$> make install
```

**Using PGXS** If an instance of PostgreSQL is already installed, then PGXS can be utilized to build and install `pgnodemx`. Ensure that PostgreSQL binaries are available via the `$PATH` environment variable then use the following commands.

```
$> make USE_PGXS=1
$> make USE_PGXS=1 install
```

## Configure

The following bash commands should configure your system to utilize `pgnodemx`. Replace all paths as appropriate. It may be prudent to visually inspect the files afterward to ensure the changes took place.

Initialize PostgreSQL (if needed):

```
$> initdb -D /path/to/data/directory
```

Create Target Database (if needed):

```
$> createdb <database>
```

Install `pgnodemx` functions:

Edit `postgresql.conf` and add `pgnodemx` to the `shared_preload_libraries` line, and change custom settings as mentioned above.

Finally, restart PostgreSQL (method may vary):

```
$> service postgresql restart
```

Install the extension into your database:

```
psql <database>
CREATE EXTENSION pgnodemx;
```

## TODO

- Map more `/proc` files to virtual tables
- Add support for “hybrid” cgroup mode