Setting up Exporters

Contents

pgMonitor

pgMonitor is your all-in-one tool to easily create an environment to visualize the health and performance of your PostgreSQL cluster.

pgMonitor combines a suite of tools to facilitate the collection and visualization of important metrics that you need be aware of in your PostgreSQL database and your host environment, including:

- Connection counts: how busy is your system being accessed and if connections are hanging
- Database size: how much disk your cluster is using
- Replication lag: know if your replicas are falling behind in loading data from your primary
- Transaction wraparound: don't let your PostgreSQL database stop working
- Bloat: how much extra space are your tables and indexes using
- System metrics: CPU, Memory, I/O, uptime

pgMonitor is also highly configurable, and advanced users can design their own metrics, visualizations, and add in other features such as alerting.

Running pgMonitor will give you confidence in understanding how well your PostgreSQL cluster is performing, and will provide you the information to make calculated adjustments to your environment.

Contents

- Purpose
- Supported Platforms
- Operating Systems
- PostgreSQL
- Installation
- Roadmap
- Version History
- Sponsors
- Legal Notices

Purpose

pgMonitor is an open-source monitoring solution for PostgreSQL and the systems that it runs on. pgMonitor came from the need to provide a way to easily create a visual environment to monitor all the metrics a database administrator needs to proactively ensure the health of the system.

pgMonitor combines multiple open-source software packages and necessary configuration to create a robust PostgreSQL monitoring environment. These include:

- Prometheus - an open-source metrics collector that is highly customizable.
- Grafana - an open-source data visualizer that allows you to generate many different kinds of charts and graphs.
- PostgreSQL Exporter - an open-source data export to Prometheus that supports collecting metrics from any PostgreSQL server version 9.1 and above.

Supported Platforms

Operating Systems

- Prometheus/Alertmanager & Grafana: CentOS/RHEL 7 or greater, Windows Server 2012R2 or later
- Exporters (node, wmi, postgres): CentOS/RHEL 6 or greater (node), Windows Server 2012R2 or later (WMI & postgres)

PostgreSQL

- pgMonitor plans to support all PostgreSQL versions that are actively supported by the PostgreSQL community. Once a major version of PostgreSQL reaches its end-of-life (EOL), pgMonitor will cease supporting that major version.
- 12, 11, 10, 9.6, 9.5, 9.4

Known issues

- PostgreSQL 10+ SCRAM-SHA-256 encrypted passwords are supported on the Linux version of pgMonitor 4.0 or later only.

Installation

Installation instructions for each package are provided in that packages subfolder. Each step in the installation process is listed here, with a link to additional to further installation instructions for each package.

1. **exporter**

2. **Prometheus**

3. **Grafana**

Roadmap

- Additional monitoring metrics out-of-the-box
- Improved visualizations
- Project build testing

Version History

For the full history of pgMonitor, please see the CHANGELOG.

Sponsors

Crunchy Data is pleased to sponsor pgMonitor and many other open-source projects to help promote support the PostgreSQL community and software ecosystem.

Legal Notices

Copyright © 2019 Crunchy Data Solutions, Inc.

CRUNCHY DATA SOLUTIONS, INC. PROVIDES THIS GUIDE "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Crunchy, Crunchy Data Solutions, Inc. and the Crunchy Hippo Logo are trademarks of Crunchy Data Solutions, Inc.

The Linux instructions below use RHEL, but any Linux-based system should work. Crunchy Data customers can obtain Linux packages through the Crunchy Customer Portal; for Windows packages, contact Crunchy Data directly.

- Installation
- RPM installs
- Non-RPMs installs
- Windows installs
- Upgrading
- Setup
- RHEL / CentOS 7 (preferred)
- RHEL / CentOS 6

- Windows Server 2012R2
- Metrics Collected
- PostgreSQL
- System

Installation

RPM installs

The following RPM packages are available to Crunchy Data customers through the Crunchy Customer Portal. *After installing via these packages, continue reading at the Setup section.*

Package Name	Description
node_exporter	Base package for node_exporter
postgres_exporter	Base package for postgres_exporter
pgmonitor-pg##-extras	Crunchy optimized configurations for postgres_exporter. Note that each major version of PostgreSQL
pgmonitor-pg-common	Package containing postgres_exporter items common for all versions of postgres
pgmonitor-node_exporter-extras	Crunchy optimized configurations for node_exporter
pg_bloat_check	Package for pg_bloat_check script
pgbouncer_fdw	Package for the pgbouncer_fdw extension
blackbox_exporter	Package for the blackbox_exporter

Available Packages

Non-RPM installs

For non-package installations on Linux, applications can be downloaded from their respective repositories:

Library	
node_exporter	https://github.com/prometheus/node_exporter/releases
postgres_exporter	https://github.com/wrouesnel/postgres_exporter/releases
pg_bloat_check	https://github.com/keithf4/pg_bloat_check
pgbouncer_fdw	https://github.com/CrunchyData/pgbouncer_fdw
blackbox_exporter	https://github.com/prometheus/blackbox_exporter

User and Configuration Directory Installation You will need to create a user named `ccp_monitoring` which you can do with the following command:

```
sudo useradd -m -d /var/lib/ccp_monitoring ccp_monitoring
```

Configuration File Installation All executables installed via the above releases are expected to be in the `/usr/bin` directory. A base node_exporter systemd file is expected to be in place already. An example one can be found here:

https://github.com/lest/prometheus-rpm/tree/master/node_exporter

A base blackbox_exporter systemd file is also expected to be in place. No examples are currently available.

The files contained in this repository are assumed to be installed in the following locations with the following names. In the instructions below, you should replace a double-hash (`##`) with the two-digit major version of PostgreSQL you are running (ex: 95, 96, 10, etc.).

node_exporter The `node_exporter` data directory should be `/var/lib/ccp_monitoring/node_exporter` and owned by the `ccp_monitoring` user. You can set it up with:

```
sudo install -m 0700 -o ccp_monitoring -g ccp_monitoring -d /var/lib/ccp_monitoring/node_exporter
```

The following pgMonitor configuration files should be placed according to the following mapping:

pgmonitor Configuration File	System Location
node/crunchy-node-exporter-service-el7.conf	`/etc/systemd/system/node_exporter.service.d/crunchy-node-exporter-service-el7`
node/sysconfig.node_exporter	`/etc/sysconfig/node_exporter`

postgres_exporter The following pgMonitor configuration files should be placed according to the following mapping:

pgMonitor Configuration File	System Location
crontab.txt	`/etc/postgres_exporter/##/cront`
postgres/crunchy_postgres_exporter@.service	`/usr/lib/systemd/system/crunchy_postgres_exporter@.service`
postgres/sysconfig.postgres_exporter_pg##	`/etc/sysconfig/postgres_exporter_pg##`
postgres/sysconfig.postgres_exporter_pg##_per_db	`/etc/sysconfig/postgres_exporter_pg##_per_db`
postgres/setup_pg##.sql	`/etc/postgres_exporter/##/setup_pg##.sql`
postgres/queries_*.yml	`/etc/postgres_exporter/##/queries_*.yml`
postgres/pgbackrest-info.sh	`/usr/bin/pgbackrest-info.sh`

blackbox_exporter The following pgMonitor configuration files should be placed according to the following mapping:

pgMonitor Configuration File	System Location
blackbox/blackbox_exporter.sysconfig	`/etc/sysconfig/blackbox_exporter`
blackbox/crunchy-blackbox.yml	`/etc/blackbox_exporter/crunchy-blackbox.yml`

Windows installs

The following Windows Server 2012R2 packages are available to Crunchy Data customers. *After installing via these packages, continue reading at the Windows Server 2012R2 section.*

PACKAGE NAME	DESCRIPTION
pgMonitor_client_#.#*Crunchy.win.x86*64.exe	Contains the needed metric exporters for monitoring the health of a PostgreSQL server.

The client package is run on the PostgreSQL server(s) to be monitored. *This includes the primary and all secondary servers.*

Upgrading

- See the CHANGELOG for full details on both major & minor version upgrades.

Setup

Setup on RHEL/CentOS 7 (preferred)

Service Configuration The following files contain defaults that should enable the exporters to run effectively on your system for the purposes of using pgMonitor. You should take some time to review them.

If you need to modify them, see the notes in the files for more details and recommendations: - `/etc/systemd/system/node_exporter.servic` - `/etc/sysconfig/node_exporter` - `/etc/sysconfig/postgres_exporter_pg##` - `/etc/sysconfig/postgres_exporter_pg##_per_db`

Note that `/etc/sysconfig/postgres_exporter_pg##` & `postgres_exporter_pg##_per_db` are the default sysconfig files for monitoring the database running on the local socket at /var/run/postgresql and connect to the "postgres" database. If you've installed the pgMonitor setup to a different database, modify these files accordingly or make new ones. If you make new ones, ensure the service name you enable references this file (see the Enable Services section below).

Database Configuration

General Configuration First, make sure you have installed the PostgreSQL contrib modules. You can install them with the following command:

```
sudo yum install postgresql##-contrib
```

Where **##** corresponds to your current PostgreSQL version. For PostgreSQL 11 this would be:

```
sudo yum install postgresql11-contrib
```

You will need to modify your `postgresql.conf` configuration file to tell PostgreSQL to load shared libraries. In the default setup, this file can be found at `/var/lib/pgsql/##/data/postgresql.conf`.

Modify your `postgresql.conf` configuration file to add the following shared libraries

```
shared_preload_libraries = 'pg_stat_statements,auto_explain'
```

You will need to restart your PostgreSQL instance for the change to take effect. Neither of the above extensions are used outside of the postgres database itself, but we find they are extremely useful to have loaded and available in the database when further diagnosis of issues is required.

For each database you are planning to monitor, you will need to run the following command as a PostgreSQL superuser:

```
CREATE EXTENSION pg_stat_statements;
```

If you want the `pg_stat_statements` extension to be available in all newly created databases, you can run the following command as a PostgreSQL superuser:

```
psql -d template1 -c "CREATE EXTENSION pg_stat_statements;"
```

Query File	Description
setup_pg##.sql	Creates **Monitoring Setup** with monitoring grants. Creates all necessary database objects (functions, tables
queries_bloat.yml	postgres_exporter query file to allow bloat monitoring.
queries_common.yml	postgres_exporter query file with minimal recommended queries that are common across all PG versions.
queries_per_db.yml	postgres_exporter query file with queries that gather per databse stats. WARNING: If your database has many
queries_pg##.yml	postgres_exporter query file for queries that are specific to the given version of PostgreSQL.
queries_backrest.yml	postgres_exporter query file for monitoring pgBackRest backup status. By default, new backrest data is only co
queries_pgbouncer.yml	postgres_exporter query file for monitoring pgbouncer.

By default, there are two postgres_exporter services expected to be running as of pgMonitor 4.0 and higher. One connects to the default `postgres` database that most postgresql instances come with and is meant for collecting global metrics that are the same on all databases in the instance, for example connection and replication statistics. This service uses the sysconfig file postgres_exporter_pg##. Connect to this database and run the setup_pg##.sql script to install the required database objects for pgMonitor.

The second postgres_exporter service is used to collect per-database metrics and uses the sysconfig file postgres_exporter_pg##*per*db. By default it is set to also connect to the `postgres` database, but you can add as many additional connection strings to this service for each individual database that you want metrics for. Per-db metrics include things like table/index statistics and bloat. See the section below for monitorig multitple databases for how to do this.

Note that your pg_hba.conf will have to be configured to allow the `ccp_monitoring` system user to connect as the `ccp_monitoring` role to any database in the instance. As of version 4.0 of pg_monitor, the postgres_exporter service is set to connect via local socket, so passwordless local peer authentication is the expected default. If password-based authentication is required, we recommend using SCRAM authentication, which is supported as of version 0.7.x of postgres_exporter. See our blog post for more information on SCRAM - https://info.crunchydata.com/blog/how-to-upgrade-postgresql-passwords-to-scram

The common queries to all postgres versions are contained in `queries_common.yml`. Major version specific queries are contained in a relevantly named file. Queries for more specialized monitoring are contained in additional files.

postgres_exporter only takes a single yaml file as an argument for custom queries, so this requires concatinating the relevant files together. The sysconfig files for the service help with this concatenation task and define the variable `QUERY_FILE_LIST`. Set this variable to a space delimited list of the full path names to all files that contain queries you want to be in the single file that postgres_exporter uses.

For example, to use just the common queries for PostgreSQL 9.6 modify the relevant sysconfig file and update `QUERY_FILE_LIST`.

```
QUERY_FILE_LIST="/etc/postgres_exporter/96/queries_common.yml
    /etc/postgres_exporter/96/queries_pg96.yml"
```

As an another example, to include queries for PostgreSQL 10 as well as pgBackRest, modify the relevant sysconfig file and update `QUERY_FILE_LIST`:

```
QUERY_FILE_LIST="/etc/postgres_exporter/10/queries_common.yml
    /etc/postgres_exporter/10/queries_pg10.yml /etc/postgres_exporter/10/queries_backrest.yml"
```

For replica servers, the setup is the same except that the setup_pg##.sql file does not need to be run since writes cannot be done there and it was already run on the primary.

Access Control: GRANT statements

The `ccp_monitoring` database role (created by running the "setup_pg##.sql" file above) must be allowed to connect to all databases in the cluster. To do this, run the following command to generate the necessary GRANT statements:

```
SELECT 'GRANT CONNECT ON DATABASE "' || datname || '" TO ccp_monitoring;'
FROM pg_database
WHERE datallowconn = true;
```

This should generate one or more statements similar to the following:

```
GRANT CONNECT ON DATABASE "postgres" TO ccp_monitoring;
```

Run these grant statements to then allow monitoring to connect.

Bloat setup

Run the script on the specific database(s) you will be for monitoring bloat in the cluster. See special note in crontab.txt concerning a superuser requirement for using this script

```
psql -d postgres -c "CREATE EXTENSION pgstattuple;"
/usr/bin/pg_bloat_check.py -c "host=localhost dbname=postgres user=postgres" --create_stats_table
psql -d postgres -c "GRANT SELECT,INSERT,UPDATE,DELETE,TRUNCATE ON bloat_indexes, bloat_stats,
    bloat_tables TO ccp_monitoring;"
```

The `/etc/postgres_exporter/##/crontab.txt` file is meant to be a guide for how you setup the `ccp_monitoring` *crontab*. You should modify crontab entries to schedule your bloat check for off-peak hours. This script is meant to be run at most, once a week. Once a month is usually good enough for most databases as long as the results are acted upon quickly.

The script requires being run by a database superuser by default since it must be able to run a scan on every table. If you'd like to not run it as a superuser, you will have to create a new role that has read permissions on all tables in all schemas that are to be monitored for bloat. You can then change the user in the connection string option to the script.

Blackbox Exporter   The configuration file for the blackbox_exporter provided by pgMonitor (`/etc/blackbox_exporter/crunchy-black` provides a probe for monitoring any IPv4 TCP port status. The actual target and port being monitored are controlled via the Prometheus target configuration system. Please see the pgMonitor Prometheus documentation for further details. If any additional Blackbox probes are desired, please see the upstream documentation.

PGBouncer   In order to monitor pgbouncer with pgMonitor, the pgbouncer_fdw maintained by CrunchyData is required. Please see its repository for full installation instructions. A package for this is available for Crunchy customers.

https://github.com/CrunchyData/pgbouncer_fdw

Once that is working, you should be able to add the queries_pgbouncer.yml file to the QUERY_FILE_LIST for the exporter that is monitoring the database where the FDW was installed.

```
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
sudo systemctl status node_exporter
```

If you've installed the blackbox exporter:

```
sudo systemctl enable blackbox_exporter
sudo systemctl start blackbox_exporter
sudo systemctl status blackbox_exporter
```

To most easily allow the use of multiple postgres exporters, running multiple major versions of PostgreSQL, and to avoid maintaining many similar service files, a systemd template service file is used. The name of the sysconfig EnvironmentFile to be used by the service is passed as the value after the "@" and before ".service" in the service name. The default exporter's sysconfig file is named "postgres_exporter_pg##" and tied to the major version of postgres that it was installed for. A similar EnvironmentFile exists for the per-db service. Be sure to replace the ## in the below commands first!

```
sudo systemctl enable crunchy-postgres-exporter@postgres_exporter_pg##
sudo systemctl start crunchy-postgres-exporter@postgres_exporter_pg##
sudo systemctl status crunchy-postgres-exporter@postgres_exporter_pg##

sudo systemctl enable crunchy-postgres-exporter@postgres_exporter_pg##_per_db
sudo systemctl start crunchy-postgres-exporter@postgres_exporter_pg##_per_db
sudo systemctl status crunchy-postgres-exporter@postgres_exporter_pg##_per_db
```

Monitoring multiple databases and/or running multiple postgres exporters (RHEL / CentOS 7)

Certain metrics are not cluster-wide, so multiple exporters must be run to collect all relevant metrics. As of v0.5.x of postgres_exporter, a single service can connect to multiple databases. As long as you're using the same custom query file for all of those databases, only one additional exporter service is required and this comes with pgMonitor 4.0 and above by default. The queries_per_db.yml file contains these queries and the secondary exporter can use this file to collect those metrics and avoid duplicating cluster-wide metrics. Note that some other metrics are per database as well (Ex. bloat). You can then define multiple targets for that one job in Prometheus so that all the metrics are collected together for a single database instance. Note that the "setup_*.sql" file does not need to be run on these additional databases if using the queries that pgMonitor comes with.

pgMonitor provides and recommends an example sysconfig file for this per-db exporter: `sysconfig.postgres_exporter_pg##_per_db`. If you'd like to create additional exporter services for different query files, just copy the existing ones and modify the relevant lines, mainly being the port, database name, and query file. The below example shows connecting to 3 databases in the same instance to collect their per-db metrics: `postgres`, `mydb1`, and `mydb2`.

```
OPT="--web.listen-address=0.0.0.0:9188
    --extend.query-path=/etc/postgres_exporter/11/queries_per_db.yml"
DATA_SOURCE_NAME="postgresql:///postgres?host=/var/run/postgresql/&user=ccp_monitoring&sslmode=disabl
```

As was done with the exporter service that is collecting the global metrics, also modify the `QUERY_LIST_FILE` in the new sysconfig file to only collect per-db metrics

```
QUERY_FILE_LIST="/etc/postgres_exporter/11/queries_per_db.yml"
```

Since a systemd template is used for the postgres_exporter services, all you need to do is pass the sysconfig file name as part of the new service name.

```
sudo systemctl enable crunchy-postgres-exporter@postgres_exporter_pg11_per_db
sudo systemctl start cruncy-postgres-exporter@postgres_exporter_pg11_per_db
sudo systemctl status crunchy-postgres-exporter@postgres_exporter_pg11_per_db
```

Lastly, update the Prometheus auto.d target file to include the new exporter in the same job you already had running for this system

Installation / Setup on RHEL/CentOS 6

The node_exporter and postgres_exporter services on RHEL6 require the "daemonize" package that is part of the EPEL repository. This can be turned on by running:

```
sudo yum install epel-release
```

All setup for the exporters is the same on RHEL6 as it was for 7 with the exception of the base service files. Whereas RHEL7 uses systemd, RHEL6 uses init.d. The Crunchy RHEL6 packages will create the base service files for you

```
/etc/init.d/crunchy-node-exporter
/etc/init.d/crunchy-postgres-exporter
```

Note that these service files are managed by the package and any changes you make to them could be overwritten by future updates. If you need to customize the service files for RHEL6, it's recommended making a copy and editing/using those.

Or if you are setting this up manually, the repository file locations and expected directories are:

```
node/crunchy-node-exporter-el6.service -> /etc/init.d/crunchy-postgres-exporter
postgres/crunchy-postgres-exporter-el6.service -> /etc/init.d/crunchy-postgres-exporter

/var/run/postgres_exporter/
/var/log/postgres_exporter/ (owned by postgres_exporter service user)
```

```
/var/run/node_exporter/
/var/log/node_exporter/ (owned by node_exporter service user)
```

The same /etc/sysconfig files that are used in RHEL7 above are also used in RHEL6, so follow guidance above concerning them and the notes that are contained in the files themselves.

Once the files are in place, set the service to start on boot, then manually start it

```
sudo chkconfig crunchy-node-exporter on
sudo service crunchy-node-exporter start
sudo service crunchy-node-exporter status

sudo chkconfig crunchy-postgres-exporter on
sudo service crunchy-postgres-exporter start
sudo service crunchy-postgres-exporter status
```

Running multiple postgres exporters (RHEL / CentOS 6)   If you need to run multiple postgres_exporter services, follow the same instructions as RHEL / CentOS 7 for making a new queries_XX.yml file to only gather database specific metrics. Then follow the steps below:

```
- Make a copy of the /etc/sysconfig file with a new name. If you need to collect per-db metrics,
    you can use the same per-db sysconfig file that CentOS7 uses.
- Update --web.listen-address in the new sysconfig file to use a new port number
- Update --extend.query-path in the new sysconfig file to point to the new query file generated
- Update the DATA_SOURCE_NAME in the new sysconfig file to point to the name of the database to be
    monitored
- Update the QUERY_FILE_LIST in the new sysconfig file to list all the name of yaml files used for
    metric collection
- Make a copy of the /etc/init.d/crunchy-postgres-exporter with a new name
- Update the SYSCONFIG variable in the new init.d file to match the new sysconfig file
- Update the Prometheus auto.d target file to include the new exporter in the same one you already
    had running for this system
```

Remaining steps to initialize service at boot and start it up should be the same as above for the default service.

Windows Server 2012R2

Currently the Windows installers assume you are logged in as the local Administrator account, so please ensure to do so before attempting the following.

Install the WMI and PostgreSQL exporters by:

1. Find and launch the `pgMonitor_client_#.#_Crunchy.win.x86_64.exe` file previously obtained from Crunchy Data. It will present you with the following screen:

2. Adjust the desired installation path and click 'Install'. The installer will run until you are eventually presented with this screen, where you can click 'Close':

3. The installer will then launch the configuration utility:

4. You will then be prompted to configure the `postgres_exporter`. Choose 'Yes' to do so:

5. The configuration window will open. It first prompts you for a name to be used to identify the services by. Keep the name simple, but informative. We use 'prod' as an example:

6. You will then be asked which exporter you're setting up: the cluster or the per-db. You will need one of each. We start with the global:

7. Choose '1' to configure the cluster exporter, then give it a meaningful name, e.g. payroll or whatever the main app is for this PostgreSQL cluster, enter your PostgreSQL version, and specify the default port of 9187:

8. Enter the PostgreSQL connection info. You will need the name of the database superuser account, its password, you can use 127.0.0.1 to connect, and finally enter the port PostgreSQL is listening on:

9. The script will set up the cluster exporter service and bring you back to the main menu. Choose '1' to add a service, name it the same you used in the previous step but append 'db' to the name, e.g. payrolldb, and choose '2' for the exporter type:
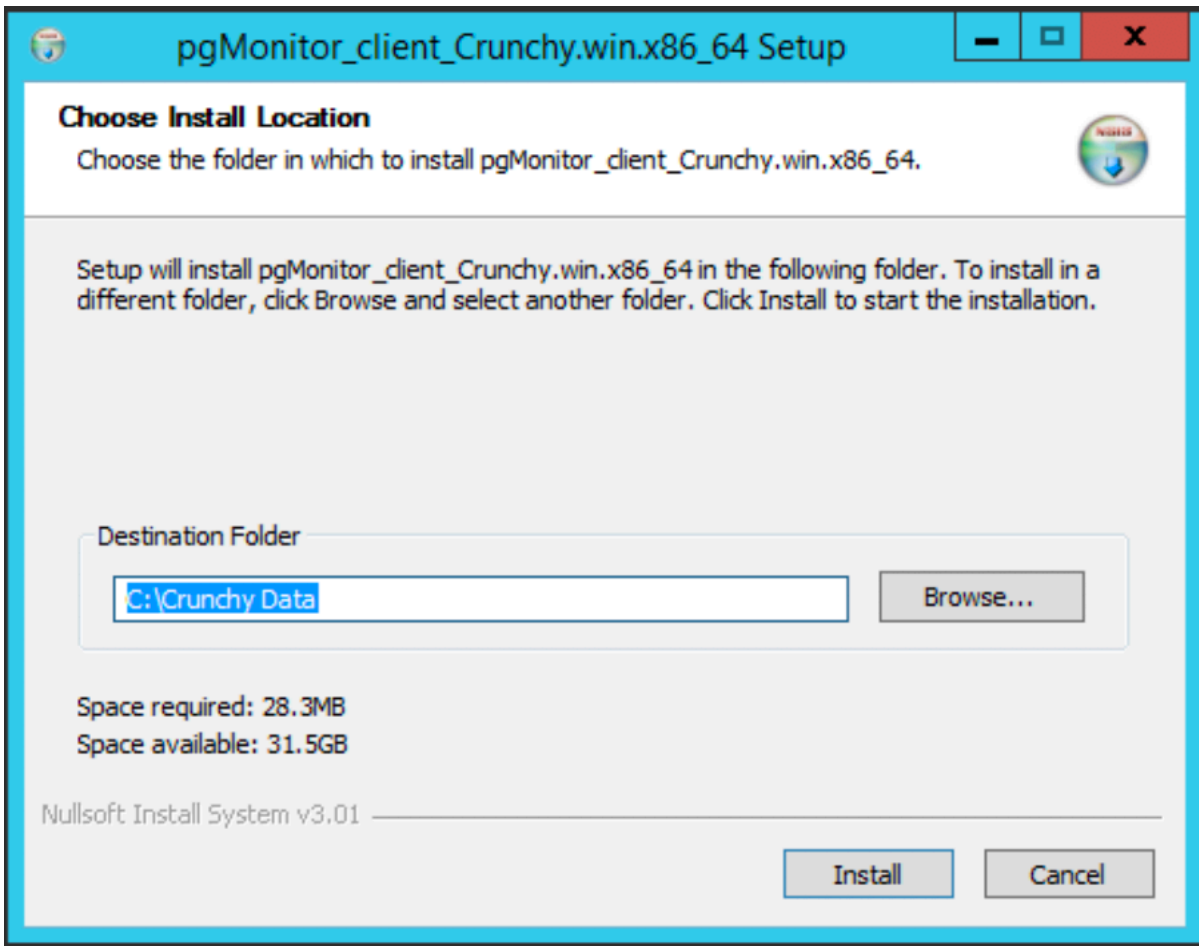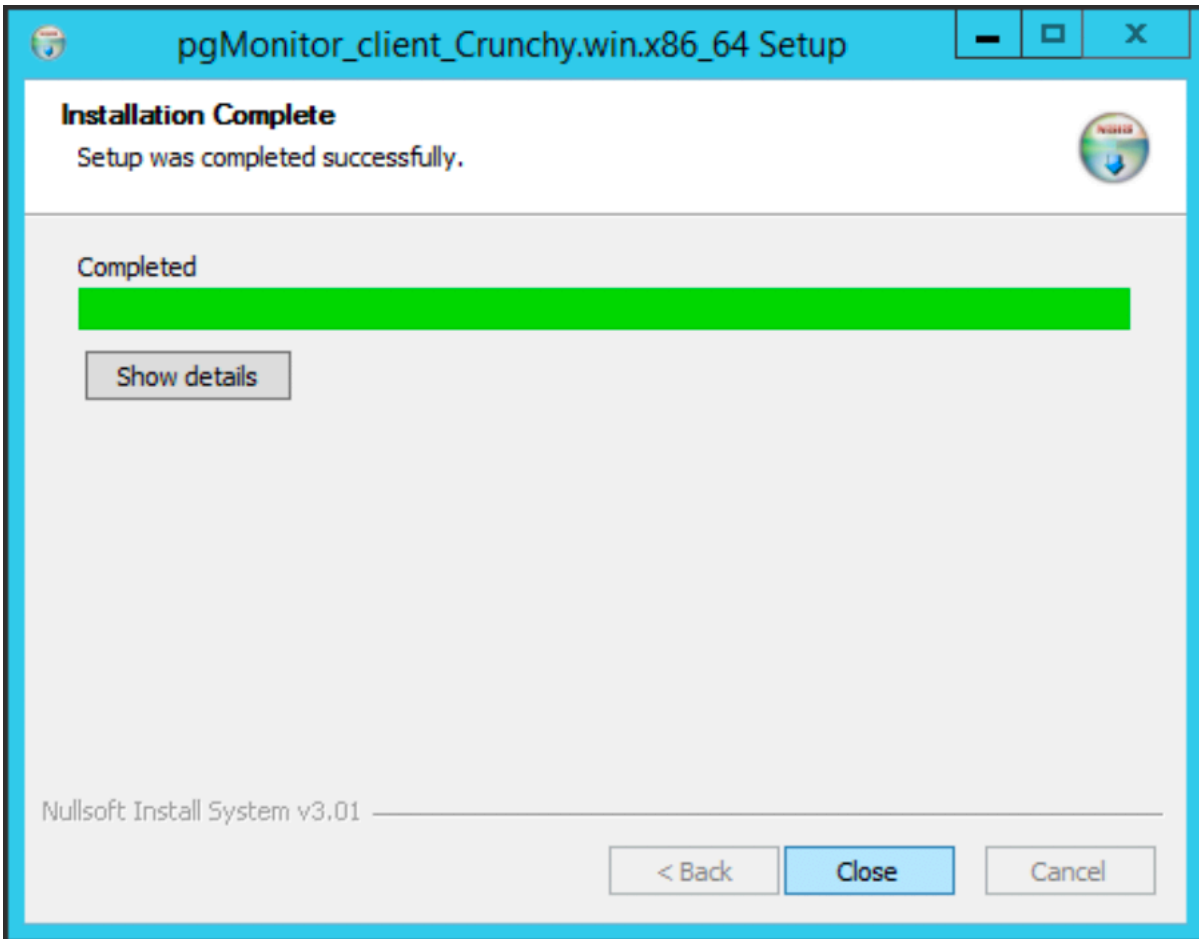
Figure 1: client installer 1
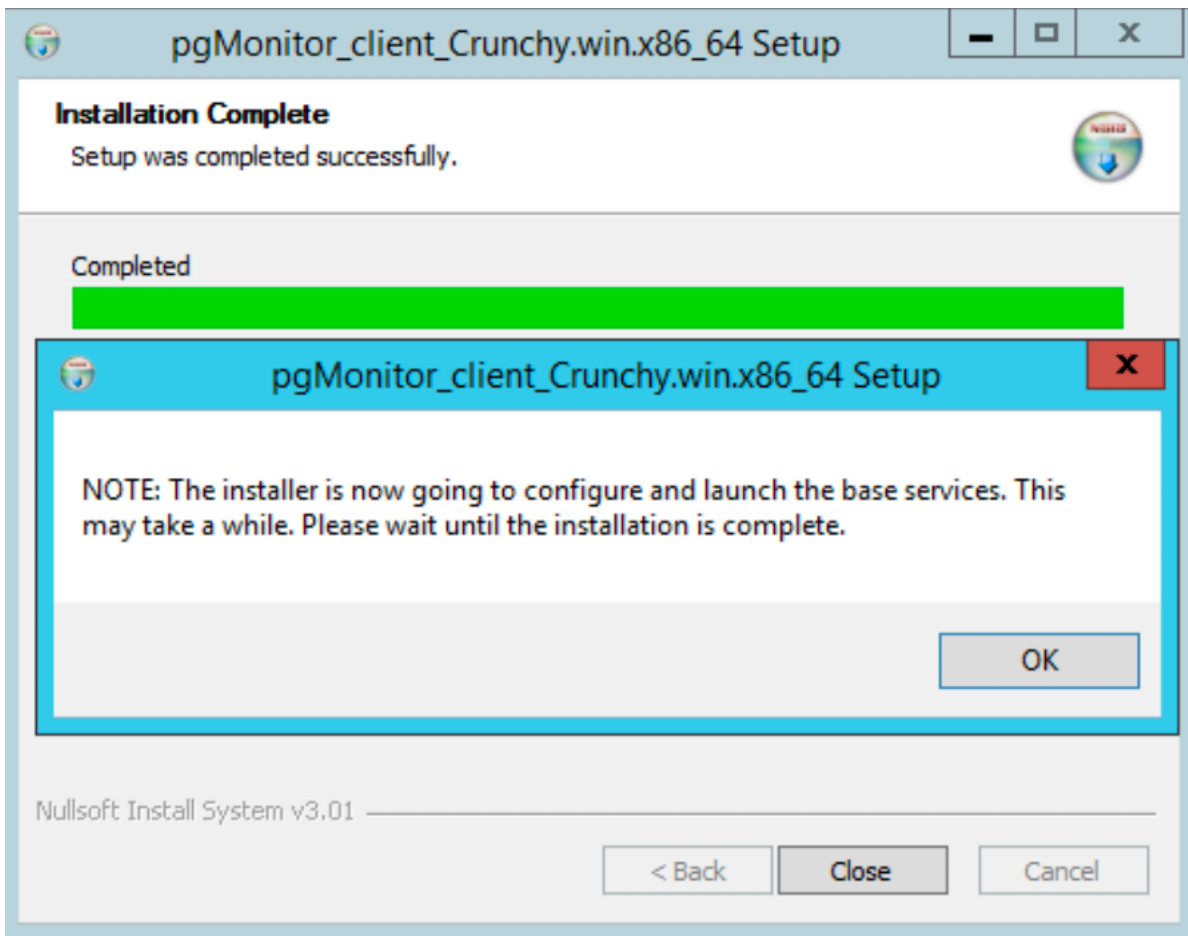

Figure 2: client installer 2
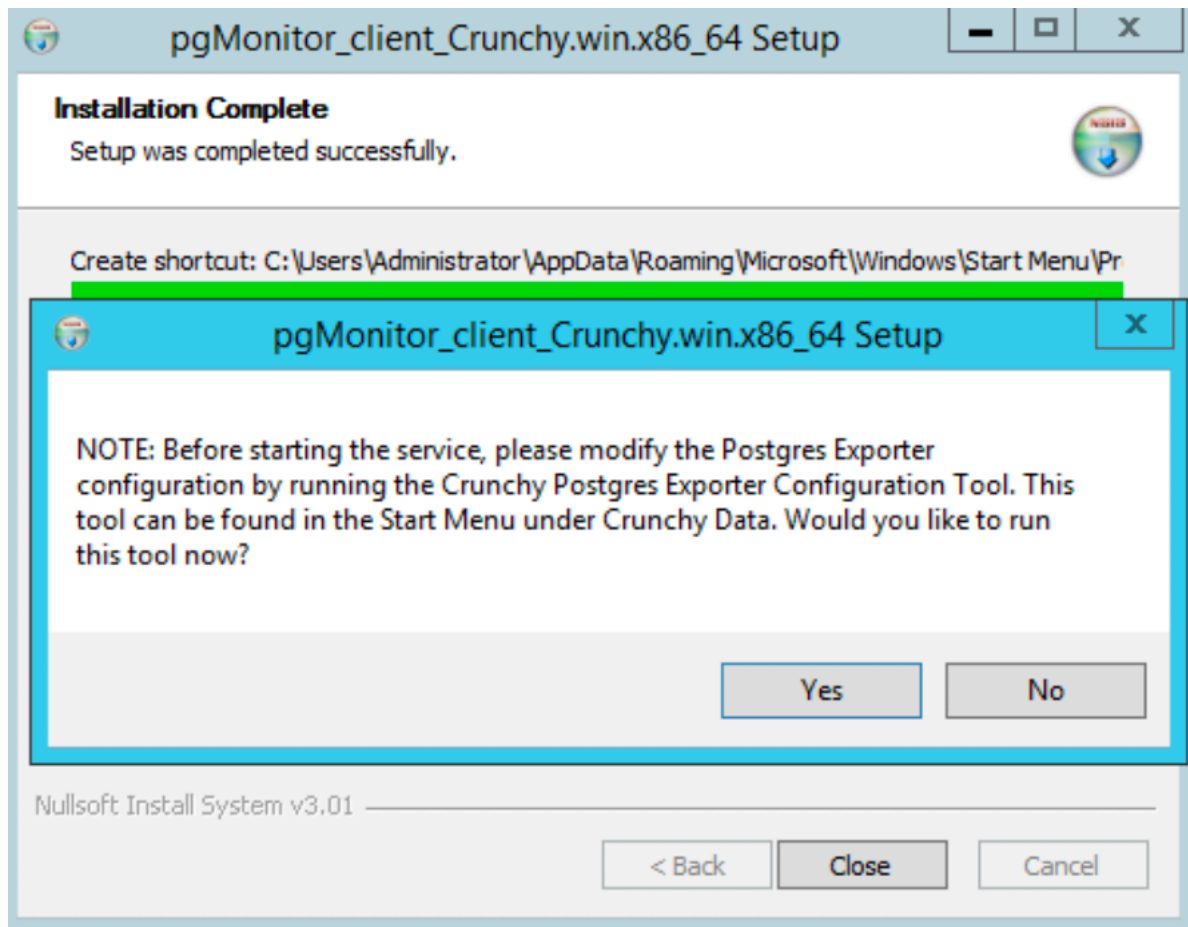
Figure 3: client installer 3
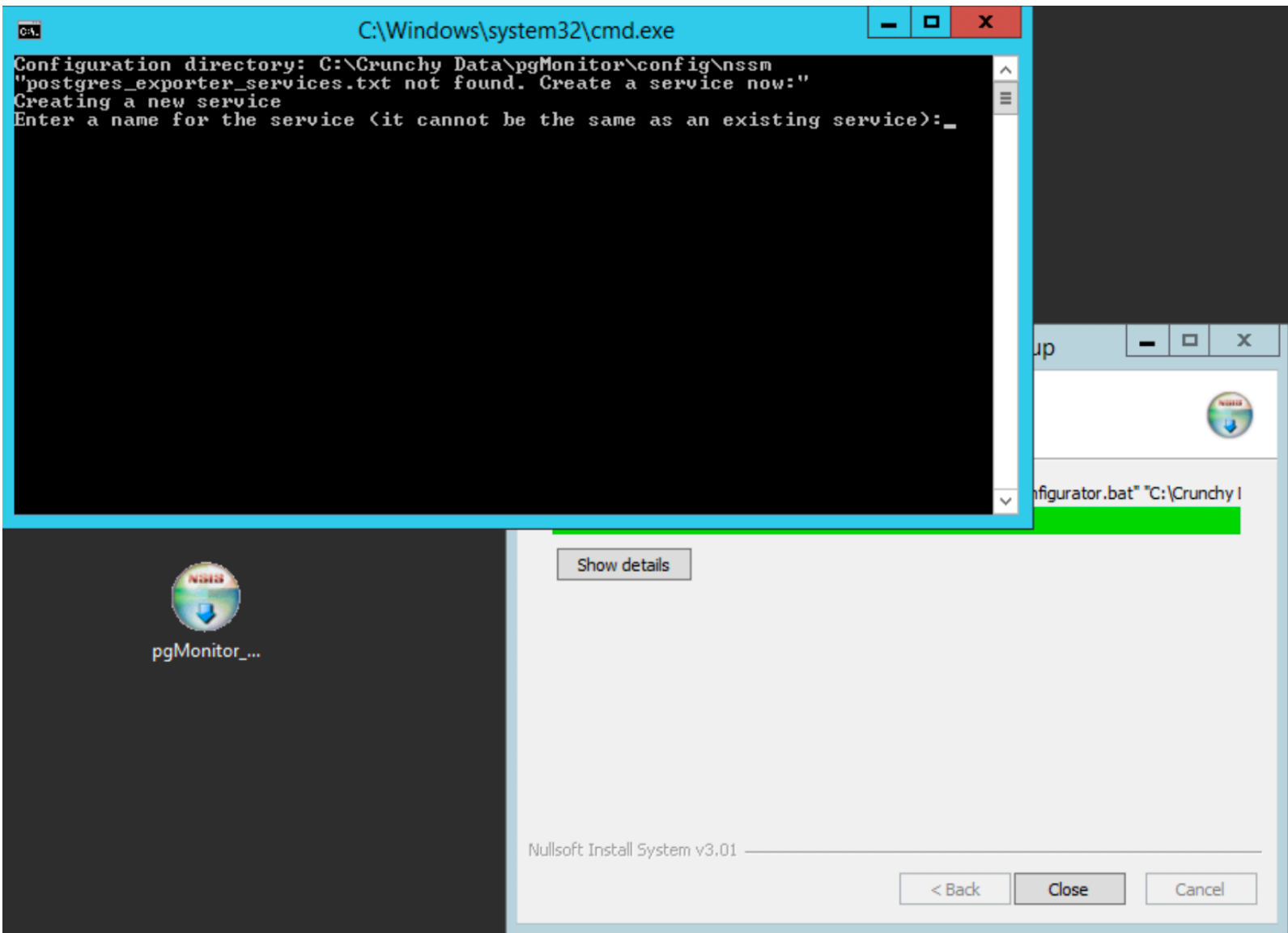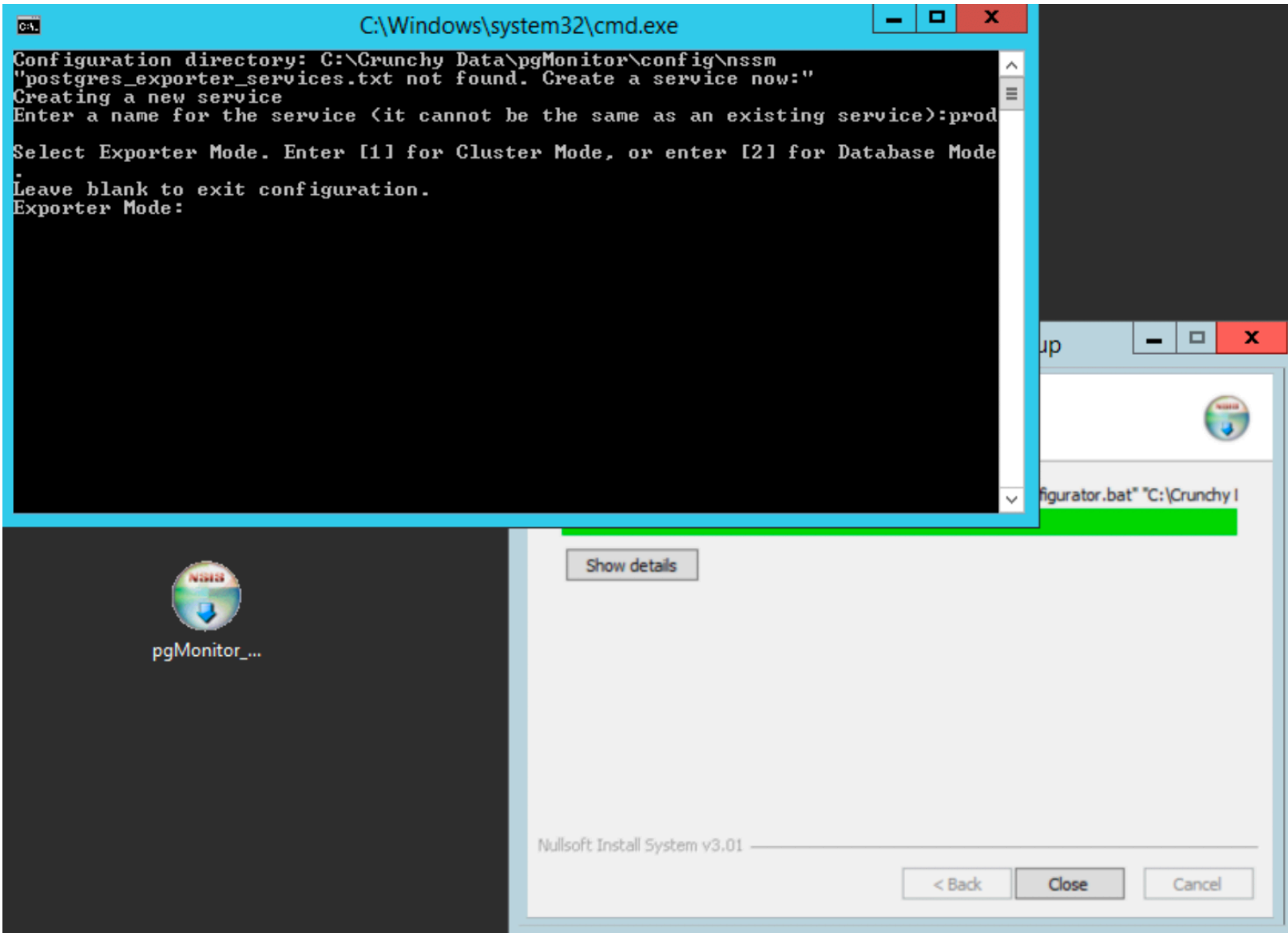

Figure 4: client installer 4

Figure 5: client installer 5

Figure 6: client installer 6
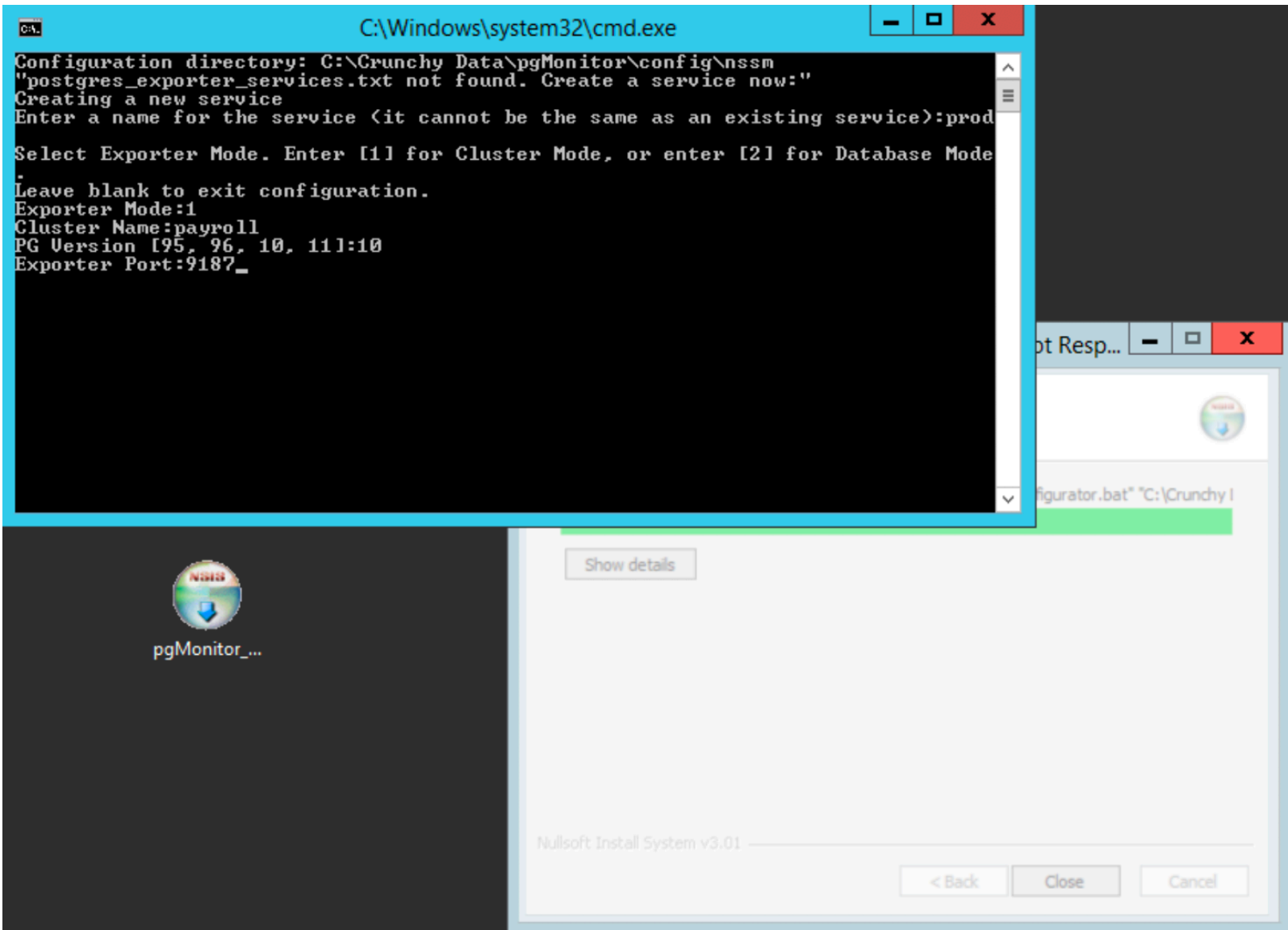
Figure 7: client installer 7

C:\Windows\system32\cmd.exe

```
Configuration directory: C:\Crunchy Data\pgMonitor\config\nssm
"postgres_exporter_services.txt not found. Create a service now:"
Creating a new service
Enter a name for the service (it cannot be the same as an existing service):prod

Select Exporter Mode. Enter [1] for Cluster Mode, or enter [2] for Database Mode
.
Leave blank to exit configuration.
Exporter Mode:1
Cluster Name:payroll
PG Version [95, 96, 10, 11]:10
Exporter Port:9187
DB User:Administrator
DB Password:secret
DB Host:127.0.0.1
DB Port:5432
```

pgMonitor_...

figurator.bat" "C:\Crunchy I

Show details

Nullsoft Install System v3.01

< Back   Close   Cancel

Figure 8: client installer 8
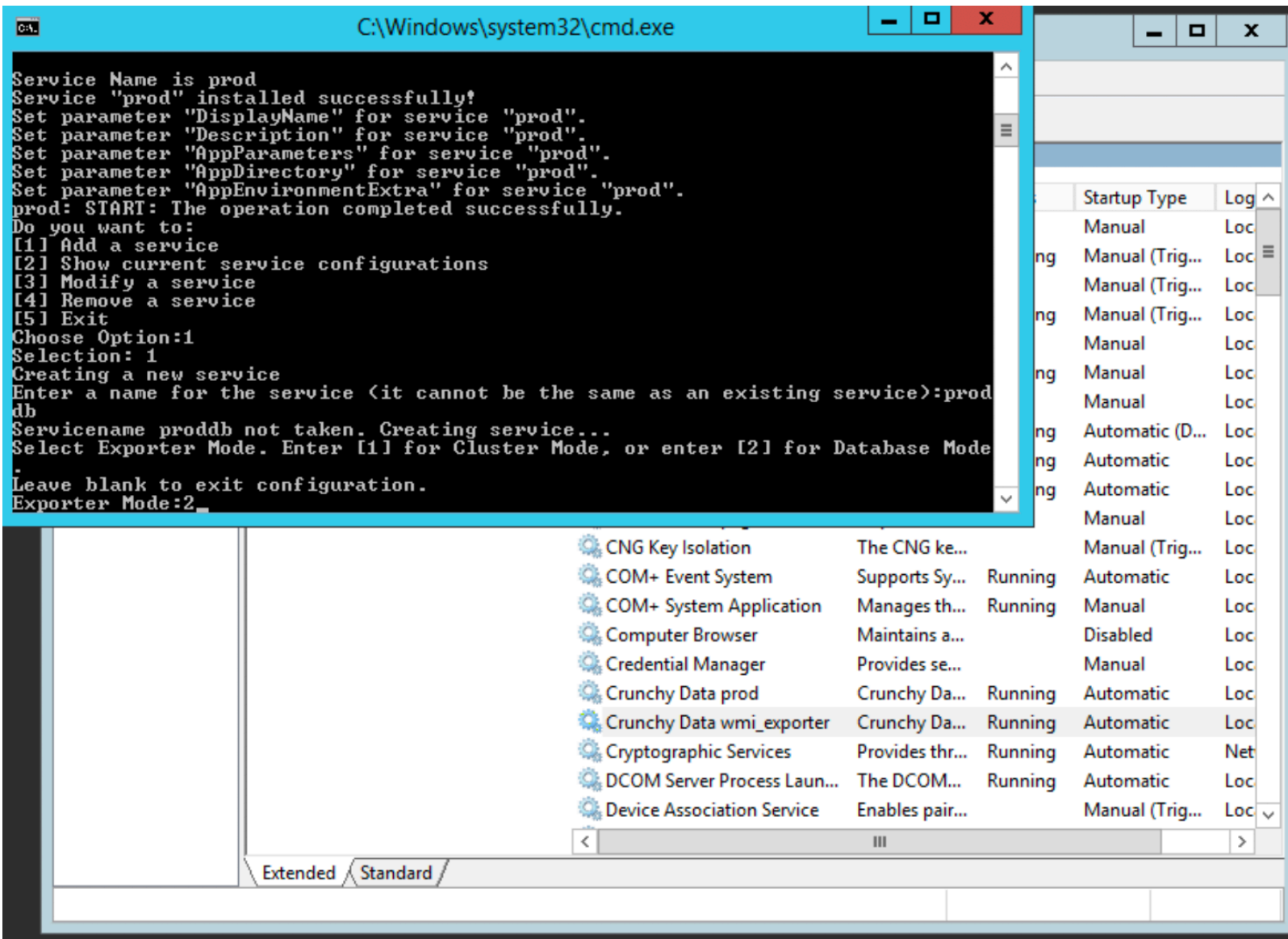
Figure 9: client installer 9

Figure 10: client installer 10

10. Enter your PostgreSQL version again, then enter '9188' as the port (two exporters cannot share the same port). Enter the same PostgreSQL connection info again. The script will setup the per-db exporter. You may now choose option '5' to exit the script:

11. Run `C:\Crunchy Data\pgMonitor\postgres_exporter\##\setup_pg##.sql` against your `postgres` database as your PostgreSQL super user replacing `##` with the major version of your PostgreSQL install (e.g. 96, 10, 11).

12. Confirm that the WMI Exporter is functional by loading http://localhost:9182/metrics in your browser:

13. Verify the cluster exporter is functional by loading http://localhost:9187/metrics in your browser. You should see multiple metrics that begin with `ccp_`:

14. Finally, confirm the per-db eporter is functional by loading http://localhost:9188/metrics in your browser:

Metrics Collected

The metrics collected by our exporters are outlined below.

PostgreSQL

PostgreSQL metrics are collected by the postgres_exporter. pgMonitor uses custom queries for its PG metrics. The default metrics that postgres_exporter comes with are all disabled except for the `pg_up` metric.

General Metrics *pg_up* - Database is up and connectable by metric collector. This is the only metrics that comes with postgres_exporter that is currently used

Common Metrics Metrics contained in the `queries_common.yml` file. These metrics are common to all versions of PostgreSQL and are recommended as a minimum default for the global exporter.
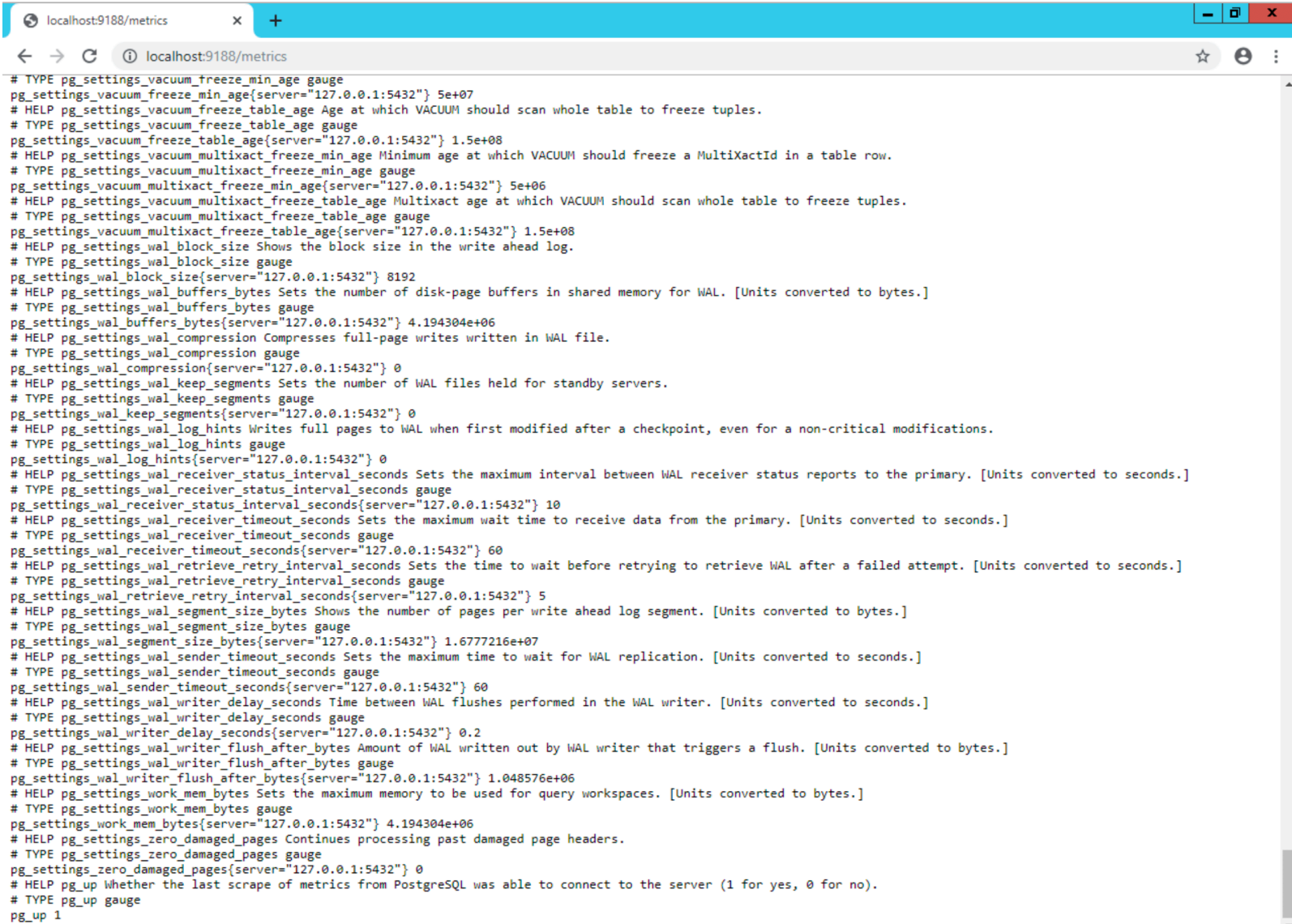
- *ccp_archive_command_status_seconds_since_last_fail* - Seconds since the last `archive_command` run failed. If zero, the `archive_command` is succeeding without error.

- *ccp_database_size_bytes* - Total size of each database in PostgreSQL instance

- *ccp_is_in_recovery_status* - Current value of the pg_is_in_recovery() function expressed as 2 for true (instance is a replica) and 1 for false (instance is a primary)

Figure 11: client installer 11

Figure 12: client installer 12

```
# TYPE pg_settings_vacuum_freeze_min_age gauge
pg_settings_vacuum_freeze_min_age{server="127.0.0.1:5432"} 5e+07
# HELP pg_settings_vacuum_freeze_table_age Age at which VACUUM should scan whole table to freeze tuples.
# TYPE pg_settings_vacuum_freeze_table_age gauge
pg_settings_vacuum_freeze_table_age{server="127.0.0.1:5432"} 1.5e+08
# HELP pg_settings_vacuum_multixact_freeze_min_age Minimum age at which VACUUM should freeze a MultiXactId in a table row.
# TYPE pg_settings_vacuum_multixact_freeze_min_age gauge
pg_settings_vacuum_multixact_freeze_min_age{server="127.0.0.1:5432"} 5e+06
# HELP pg_settings_vacuum_multixact_freeze_table_age Multixact age at which VACUUM should scan whole table to freeze tuples.
# TYPE pg_settings_vacuum_multixact_freeze_table_age gauge
pg_settings_vacuum_multixact_freeze_table_age{server="127.0.0.1:5432"} 1.5e+08
# HELP pg_settings_wal_block_size Shows the block size in the write ahead log.
# TYPE pg_settings_wal_block_size gauge
pg_settings_wal_block_size{server="127.0.0.1:5432"} 8192
# HELP pg_settings_wal_buffers_bytes Sets the number of disk-page buffers in shared memory for WAL. [Units converted to bytes.]
# TYPE pg_settings_wal_buffers_bytes gauge
pg_settings_wal_buffers_bytes{server="127.0.0.1:5432"} 4.194304e+06
# HELP pg_settings_wal_compression Compresses full-page writes written in WAL file.
# TYPE pg_settings_wal_compression gauge
pg_settings_wal_compression{server="127.0.0.1:5432"} 0
# HELP pg_settings_wal_keep_segments Sets the number of WAL files held for standby servers.
# TYPE pg_settings_wal_keep_segments gauge
pg_settings_wal_keep_segments{server="127.0.0.1:5432"} 0
# HELP pg_settings_wal_log_hints Writes full pages to WAL when first modified after a checkpoint, even for a non-critical modifications.
# TYPE pg_settings_wal_log_hints gauge
pg_settings_wal_log_hints{server="127.0.0.1:5432"} 0
# HELP pg_settings_wal_receiver_status_interval_seconds Sets the maximum interval between WAL receiver status reports to the primary. [Units converted to seconds.]
# TYPE pg_settings_wal_receiver_status_interval_seconds gauge
pg_settings_wal_receiver_status_interval_seconds{server="127.0.0.1:5432"} 10
# HELP pg_settings_wal_receiver_timeout_seconds Sets the maximum wait time to receive data from the primary. [Units converted to seconds.]
# TYPE pg_settings_wal_receiver_timeout_seconds gauge
pg_settings_wal_receiver_timeout_seconds{server="127.0.0.1:5432"} 60
# HELP pg_settings_wal_retrieve_retry_interval_seconds Sets the time to wait before retrying to retrieve WAL after a failed attempt. [Units converted to seconds.]
# TYPE pg_settings_wal_retrieve_retry_interval_seconds gauge
pg_settings_wal_retrieve_retry_interval_seconds{server="127.0.0.1:5432"} 5
# HELP pg_settings_wal_segment_size_bytes Shows the number of pages per write ahead log segment. [Units converted to bytes.]
# TYPE pg_settings_wal_segment_size_bytes gauge
pg_settings_wal_segment_size_bytes{server="127.0.0.1:5432"} 1.6777216e+07
# HELP pg_settings_wal_sender_timeout_seconds Sets the maximum time to wait for WAL replication. [Units converted to seconds.]
# TYPE pg_settings_wal_sender_timeout_seconds gauge
pg_settings_wal_sender_timeout_seconds{server="127.0.0.1:5432"} 60
# HELP pg_settings_wal_writer_delay_seconds Time between WAL flushes performed in the WAL writer. [Units converted to seconds.]
# TYPE pg_settings_wal_writer_delay_seconds gauge
pg_settings_wal_writer_delay_seconds{server="127.0.0.1:5432"} 0.2
# HELP pg_settings_wal_writer_flush_after_bytes Amount of WAL written out by WAL writer that triggers a flush. [Units converted to bytes.]
# TYPE pg_settings_wal_writer_flush_after_bytes gauge
pg_settings_wal_writer_flush_after_bytes{server="127.0.0.1:5432"} 1.048576e+06
# HELP pg_settings_work_mem_bytes Sets the maximum memory to be used for query workspaces. [Units converted to bytes.]
# TYPE pg_settings_work_mem_bytes gauge
pg_settings_work_mem_bytes{server="127.0.0.1:5432"} 4.194304e+06
# HELP pg_settings_zero_damaged_pages Continues processing past damaged page headers.
# TYPE pg_settings_zero_damaged_pages gauge
pg_settings_zero_damaged_pages{server="127.0.0.1:5432"} 0
# HELP pg_up Whether the last scrape of metrics from PostgreSQL was able to connect to the server (1 for yes, 0 for no).
# TYPE pg_up gauge
pg_up 1
```

Figure 13: client installer 13

- *ccp_locks_count* - Count of active lock types per database

- *ccp_pg_settings_checksum_status* - Value of checksum monitioring status for pg_catalog.pg_settings (postgresql.conf). 0 = valid config. 1 = settings changed. Settings history is available for review in the table `monitor.pg_settings_checksum`. To reset current config to valid after alert, run monitor.pg_settings_checksum_set_valid(). Note this will clear the history table.

- *ccp_postmaster_uptime_seconds* - Time interval in seconds since PostgreSQL database was last restarted

- *ccp_postgresql_version_current* - Version of PostgreSQL that this exporter is monitoring. Value is the 6 digit integer returned by the `server_version_num` PostgreSQL configuration variable to allow easy monitoring for version changes.

- *ccp_sequence_exhaustion_count* - Checks for any sequences that may be close to exhaustion (by default greater than 75% usage). Note this checks the sequences themselves, not the values contained in the columns that use said sequences. Function `monitor.sequence_status()` can provide more details if run directly on database instance.

- *ccp_settings_pending_restart_count* - Number of settings from pg_settings catalog in a pending_restart state. This value is from the similarly named column found in pg_catalog.pg_settings.

The meaning of the following `ccp_transaction_wraparound` metrics, and how to manage when they are triggered, is covered more extensively in this blog post: https://info.crunchydata.com/blog/managing-transaction-id-wraparound-in-postgresql

- *ccp_transaction_wraparound_percent_towards_emergency_autovac* - Recommended thresholds set to 75%/95% when first evaluating vacuum settings on new systems. Once those have been reviewed and at least one instance-wide vacuum has been run, recommend thresholds of 110%/125%. Alerting above 100% for extended periods of time means that autovacuum is not able to keep up with current transaction rate and needs further tuning.

- *ccp_transaction_wraparound_percent_towards_wraparound* - Recommend thresholds set to 50%/75%. If any of these thresholds is tripped, current vacuum settings must be evaluated and tuned ASAP. If critical threshold is reached, it is vitally important that vacuum be run on tables with old transaction IDs to avoid the cluster being forced to shut down and only be able to run in single user mode.

The following `ccp_stat_bgwriter` metrics are statistics collected from the [pg_stat_bgwriter](#) view for monitoring performance. These metrics cover important performance information about flushing data out to disk. Please see the documentation for further details on these metrics.

- *ccp_stat_bgwriter_buffers_alloc*

- *ccp_stat_bgwriter_buffers_backend*

- *ccp_stat_bgwriter_buffers_backend_fsync*

- *ccp_stat_bgwriter_buffers_checkpoint*

- *ccp_stat_bgwriter_buffers_clean*

The following `ccp_stat_database_*` metrics are statistics collected from the [pg_stat_database](#) view.

- *ccp_stat_database_blks_hit*

- *ccp_stat_database_blks_read*

- *ccp_stat_database_conflicts*

- *ccp_stat_database_deadlocks*

- *ccp_stat_database_tup_deleted*

- *ccp_stat_database_tup_fetched*

- *ccp_stat_database_tup_inserted*

- *ccp_stat_database_tup_returned*

- *ccp_stat_database_tup_updated*

- *ccp_stat_database_xact_commit*

- *ccp_stat_database_xact_rollback*

PostgreSQL Version Specific Metrics   The following metrics either require special considerations when monitoring specific versions of PostgreSQL, or are only available for specific versions. These metrics are found in the `queries_pg##.yml` files, where $\#\#$ is the major version of PG. Unless otherwise noted, the below metrics are available for all versions of PG. These metrics are recommend as a minimum default for the global exporter.

- *ccp_connection_stats_active* - Count of active connections

- *ccp_connection_stats_idle* - Count of idle connections

- *ccp_connection_stats_idle_in_txn* - Count of idle in transaction connections

- *ccp_connection_stats_max_connections* - Current value of max_connections for reference

- *ccp_connection_stats_max_idle_in_txn_time* - Runtime of longest idle in transaction (IIT) session.

- *ccp_connection_stats_max_query_time* - Runtime of longest general query (inclusive of IIT).

- *ccp_replication_lag_replay_time* - Only provides values on replica instances. Time since replica received and replayed a WAL file. Note this is not the main way to determine if a replica is behind its primary. It only monitors the time the replica replayed the WAL vs what it has received. It is a secondary metric for monitoring WAL replay on the replica itself.

- *ccp_replication_lag_size_bytes* - Only provides values on instances that have attached replicas (primary, cascading replica). Tracks byte lag of every streaming replica connected to this database instance. This is the main way that replication lag is monitored. Note that if you have WAL replay only replicas, this will not be reflected here.

- *ccp_replication_slots_active* - Active state of given replication slot. 1 = true. 0 = false.

- *ccp_replication_slots_retained_bytes* - The amount of WAL (in bytes) being retained for given slot.

- *ccp_wal_activity_total_size_bytes* - Current size in bytes of the WAL directory

- *ccp_wal_activity_last_5_min_size_bytes* - PostgreSQL 10 and later only. Current size in bytes of the last 5 minutes of WAL generation. Includes recycled WALs.

- *ccp_pg_hba_checksum_status* - PostgreSQL 10 and later only. Value of checksum monitioring status for pg_catalog.pg_hba_file_rules (pg_hba.conf). 0 = valid config. 1 = settings changed. Settings history is available for review in the table `monitor.pg_hba_checksum`. To reset current config to valid after alert, run monitor.pg_hba_checksum_set_valid(). Note this will clear the history table.

- *ccp_data_checksum_failure_count* - PostgreSQL 12 and later only. Total number of checksum failures on this database.

- *ccp_data_checksum_failure_time_since_last_failure_seconds* - PostgreSQL 12 and later only. Time interval in seconds since the last checksum failure was encountered.

Backup Metrics   Backup monitoring only covers pgBackRest at this time. These metrics are found in the `queries_backrest.yml` file. These metrics only need to be collected once per database instance so should be collected by the global postgres_exporter.

- *ccp_backrest_last_full_backup_time_since_completion_seconds* - Time since completion of last pgBackRest FULL backup

- *ccp_backrest_last_diff_backup_time_since_completion_seconds* - Time since completion of last pgBackRest DIFFERENTIAL backup. Note that FULL backup counts as a successful DIFFERENTIAL for the given stanza.

- *ccp_backrest_last_incr_backup_time_since_completion_seconds* - Time since completion of last pgBackRest INCREMENTAL backup. Note that both FULL and DIFFERENTIAL backups count as a successful INCREMENTAL for the given stanza.

- *ccp_backrest_last_info_runtime_backup_runtime_seconds* - Last successful runtime of each backup type (full/diff/incr).

- *ccp_backrest_last_info_repo_backup_size_bytes* - Actual size of only this individual backup in the pgbackrest repository

- *ccp_backrest_last_info_repo_total_size_bytes* - Total size of this backup in the pgbackrest repository, including all required previous backups and WAL

Per-Database Metrics These are metrics that are only available on a per-database level. These metrics are found in the `queries_per_db.yml` file. These metrics are optional and recommended for the non-global, per-db postgres_exporter. They can be included in the global exporter as well if the global database needs per-database metrics monitored. Please note that depending on the number of objects in your database, collecting these metrics can greatly increase the storage requirements for Prometheus since all of these metrics are being collected for each individual object.

- *ccp_table_size_size_bytes* - Table size inclusive of all indexes in that table

The following `ccp_stat_user_tables_*` metrics are statistics collected from the [pg_stat_user_tables](). Please see the PG documentation for descriptions of these metrics.

- *ccp_stat_user_tables_analyze_count*

- *ccp_stat_user_tables_autoanalyze_count*

- *ccp_stat_user_tables_autovacuum_count*

- *ccp_stat_user_tables_n_tup_del*

- *ccp_stat_user_tables_n_tup_ins*

- *ccp_stat_user_tables_n_tup_upd*

- *ccp_stat_user_tables_vacuum_count*

Bloat Metrics Bloat metrics are only available if the `pg_bloat_check` script has been setup to run. See instructions above. These metrics are found in the `queries_bloat.yml` file. These metrics are per-database so, should be used by the per-db postgres_exporter.

- *ccp_bloat_check_size_bytes* - Size of object in bytes

- *ccp_bloat_check_total_wasted_space_bytes* - Total wasted space in bytes of given object

pgBouncer Metrics The following metric prefixes correspond to the SHOW command views found in the [pgBouncer documentation](). Each column found in the SHOW view is a separate metric under the respective prefix. Ex: `ccp_pgbouncer_pools_client_active` corresponds to the `SHOW POOLS` view's `client_active` column. These metrics are found in the `queries_bouncer.yml` file. These metrics only need to be collected once per database instance so should be collected by the global postgres_exporter.

- *ccp_pgbouncer_pools* - SHOW POOLS

- *ccp_pgbouncer_databases* - SHOW DATABASES

- *ccp_pgbouncer_clients* - SHOW CLIENTS

- *ccp_pgbouncer_servers* - SHOW SERVERS

- *ccp_pgbouncer_lists* - SHOW LISTS

System

*NIX Operating System metrics (Linux, BSD, etc) are collected using the [node_exporter]() provided by the Prometheus team. pgMonitor only collects the default metrics provided by node_exporter, but many additional metrics are available if needed.

Windows Operating System metrics are collected by the [wmi_exporter]().

Prometheus can be set up on any Linux-based system, but pgMonitor currently only supports running it on RHEL/CentOS 7. Crunchy Data additionally makes Prometheus available on Windows Server 2012R2 for their customers.

- [Installation]()
  - [RHEL / CentOS 7]()
  - [Windows Server 2012R2]()
- [Upgrading]()
- [Setup]()
  - RHEL / CentOS 7
  - [Windows Server 2012R2]()

Installation

RHEL / CentOS 7

With RPM Packages   There are RPM packages available to Crunchy Data customers through the Crunchy Customer Portal.

After installing via these RPMs, you can continue reading at the Setup section.

Package Name	Description
alertmanager	Base package for the Alertmanager
prometheus2	Base package for Prometheus 2.x
pgmonitor-alertmanager-extras	Custom Crunchy configurations for Alertmanager
pgmonitor-prometheus-extras	Custom Crunchy configurations for Prometheus

Available Packages

Without Crunchy Data Packages   For installations without using packages provided by Crunchy Data, we recommend using the repository maintained at https://github.com/lest/prometheus-rpm. Instructions for setup and installation are contained there. Note this only sets up the base service. The additional files and steps for pgMonitor still need to be set up as instructed below.

Or you can also download Prometheus and Alertmanager from the original site at https://prometheus.io/download. Note that no base service setup is provided here, just the binaries.

Minimum Versions   pgMonitor assumes to be using at least Prometheus 2.9.x. We recommend to always use the latest minor version of Prometheus.

User and Configuration Directory Installation   You will need to create a system user named `ccp_monitoring` which you can do with the following command:

```
sudo useradd -d /var/lib/ccp_monitoring ccp_monitoring
```

Configuration File Installation   The files contained in this repository are assumed to be installed in the following locations with the following names:

Prometheus

The Prometheus data directory should be `/var/lib/ccp_monitoring/prometheus` and owned by the `ccp_monitoring` user. You can set it up with:

```
sudo install -d -m 0700 -u ccp_monitoring -g ccp_monitoring /var/lib/ccp_monitoring/prometheus
```

The following pgmonitor configuration files should be placed according to the following mapping:

pgMonitor Configuration File	System Location
crunchy-prometheus-service-el7.conf	/etc/systemd/system/prometheus.service.d/crunchy-prometheus-service-el7.con
sysconfig.prometheus	/etc/sysconfig/prometheus
crunchy-prometheus.yml	/etc/prometheus/crunchy-prometheus.yml
auto.d/*.yml.example	/etc/prometheus/auto.d/*.yml.example
crunchy-alertmanager.yml	/etc/prometheus/crunchy-alertmanager.yml
alert-rules.d/crunchy-alert-rules*.yml.example	/etc/prometheus/alert-rules.d/crunchy-alert-rules-*.yml.example

Alertmanager

The Alertmanager data directory should be `/var/lib/ccp_monitoring/alertmanager` and owned by the `ccp_monitoring` user. You can set it up with:

```
sudo install -d -m 0700 -o ccp_monitoring -g ccp_monitoring /var/lib/ccp_monitoring/alertmanager
```

The following pgMonitor configuration files should be placed according to the following mapping:

pgMonitor Configuration File	System Location
crunchy-alertmanager-service-el7.conf	/etc/systemd/system/alertmanager.service.d/crunchy-alertmanager-service-el7.conf
sysconfig.alertmanager	/etc/sysconfig/alertmanager

Windows Server 2012R2   There are Windows Servfer 2012R2 packages available to Crunchy Data customers who contact Crunchy Data directly.

If you install the below available packages, you can continue reading at the Setup section.

Package Name	Description
pgMonitor_server_1.0_Crunchy.win.x86_64.exe	Installer package for the Prometheus, Alertmanager, and Grafana servers

Available Packages

Upgrading

When upgrading from pgmonitor 1.x to 2.x, note that the alerting rules for node_exporter metrics have had many of their names changed.   If you've changed the provided alerting rules file, installing the new package should create a file called `/etc/prometheus/crunchy-alert-rules.yml.rpmnew` and not overwrite your current file.   You should be able to copy the new rules as needed from there.

Setup

Setup on RHEL/CentOS 7

Service Configuration   The following files contain defaults that should enable Prometheus and Alertmanager to run effectively on your system for the purposes of using pgmonitor. You should take some time to review them.

If you need to modify them, see the notes in the files for more details and recommendations:

- `/etc/systemd/system/prometheus.service.d/crunchy-prometheus-service-el7.conf`
- `/etc/systemd/system/alertmanager.service.d/crunchy-alertmanager-service-el7.conf`

The below files contain startup properties for Prometheues and Alertmanager. Please review and modify these files as you see fit:

- `/etc/sysconfig/prometheus`
- `/etc/sysconfig/alertmanager`

The below files dictate how Prometheus and Alertmanager will behave at runtime for the purposes of using pgmonitor. Please review each file below and follow the instructions in order to set things up:

File	Instructions
`/etc/prometheus/crunchy-prometheus.yml`	Modify to set scrape interval if different from the default of
`/etc/prometheus/crunchy-alertmanager.yml`	Setup alert target (e.g., SMTP, SMS, etc.), receiver and rou
`/etc/prometheus/alert-ruled.d/crunchy-alert-rules-\*.yml.example`	Update rules as needed and remove `.example` suffix. Prom
`/etc/prometheus/auto.d/*.yml`	You will need at least one file with a final `.yml` extension.

Blackbox Exporter   By default, the Blackbox exporter probes are commented out in the {{ shell }}crunchy-prometheus.yml{{ /shell }} file; please see the notes in that commented out section. For the default IPv4 TCP port targets that pgMonitor configures the blackbox_exporter with, the desired monitoring targets can be configured under the {{ yaml }}static_configs: targets{{ /yaml }} section of the {{ yaml }}blackbox_tcp_services{{ /yaml }} job; some examples for Grafana & Patroni are given there. It is also possible to

create another auto-scrape target directory similar to `auto.d` and manage your blackbox targets more dynamically.

If you configure additional probes beyond the one that pgMonitor comes with, you will need to create a different Prometheus `job_name` for them for the given `params: module` name.

An example rules file for monitoring Blackbox probes, `crunchy-alert-rules-blackbox.yml.example`, is available in the `alert-rules.d` folder.

Enable Services   To enable and start Prometheus as a service, execute the following commands:

```
sudo systemctl enable prometheus
sudo systemctl start prometheus
sudo systemctl status prometheus
```

To enable and start Alertmanager as a service, execute the following commands:

```
sudo systemctl enable alertmanager
sudo systemctl start alertmanager
sudo systemctl status alertmanager
```

Setup Windows Server 2012R2

Currently the Windows installers assume you are logged in as the local Administrator account, so please ensure to do so before attempting the following.

Install the Prometheus, AlertManager, and Grafana servers by:

1. Find and launch the `pgMonitor_server_1.0_Crunchy.win.x86_64.exe` file previously obtained from Crunchy Data. It will present you with the following screen. Choose your install path and click 'Install'.

Figure 14: server_installer_1.png

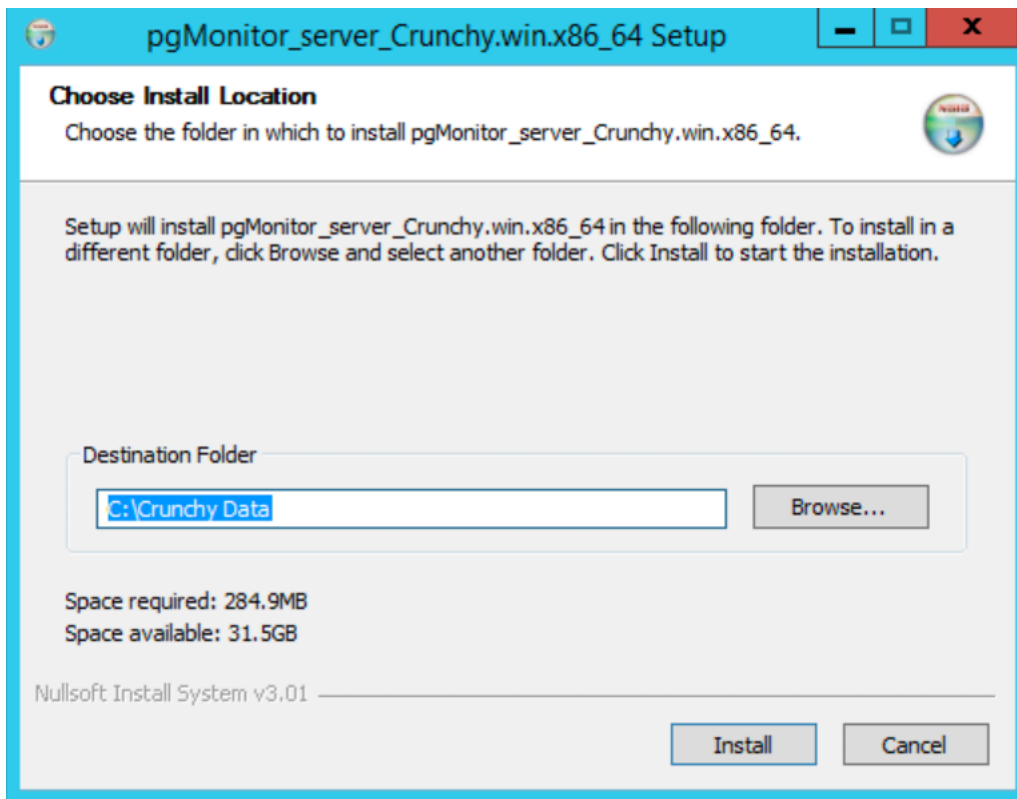2. Once installation has finished, clicked 'Close':

3. The installer will launch the Windows services that were just installed. Click 'OK' to proceed:

4. You will now be prompted to launch the configuration tool. Select 'Yes' to continue:

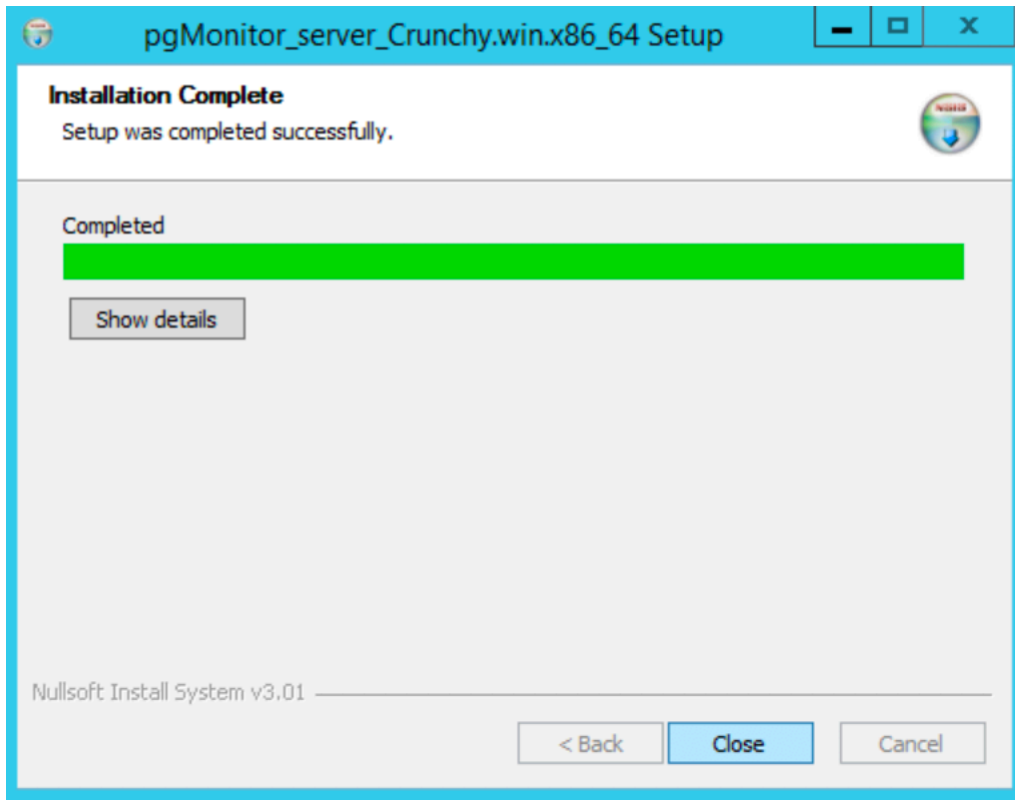5. Select '1' to tell Prometheus about the exporters it should scrape metrics from:

Figure 15: server_installer_2.png
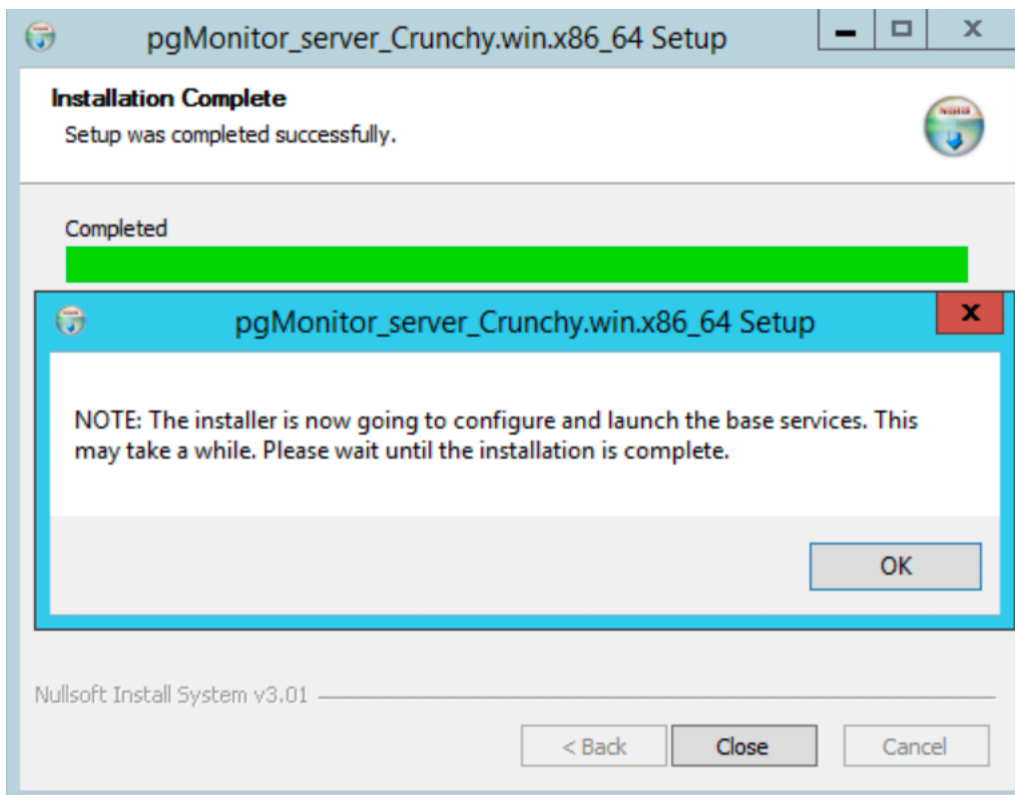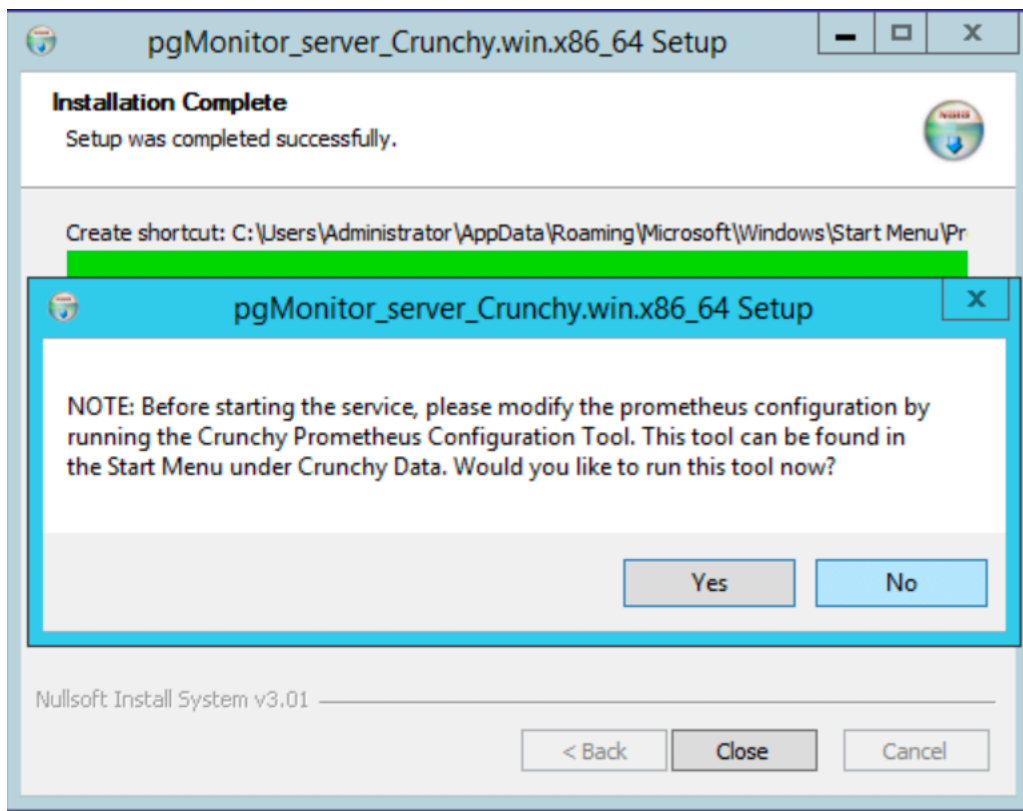

Figure 16: server_installer_3.png

Figure 17: server_installer_4.png

6. Enter the hostname (just the hostname, not the FQDN) of the PostgreSQL server that the exporters are running on. Next, enter the IP address of the PostgreSQL server, and the WMI port (default is 9182):

7. Enter a cluster name. This should be something simple but meaningful to identify the PostgreSQL cluster in question, e.g. payroll. Then enter the port used for both the cluster/global `postgres_exporter` (9187 by default) and the per-db `postgres_exporter` (9188 in our directions):

8. You can now choose '2' to exit the configuration tool:

9. You can now verify that Prometheus is running by loading http://localhost:9090 in your browser:

10. Finally, verify Prometheus can access the exporters by choosing 'Status' and then 'Targets':

11. You should see all configured exporters (1 per PostgreWSQL server, and 2 more per PostgreSQL instance), all with a green 'Up' status:

There are RPM packages available to Crunchy Data customers through the Crunchy Customer Portal. Otherwise the Grafana RPM Package can be downloaded and installed from https://grafana.com/grafana/download. There is no difference between the Crunchy provided package and the one directly from Grafana. Windows Server 2012R2 packges are available directly from Crunchy Data to customers.

- Installation
  - Linux
  - Windows Server 2012R2
- Upgrading
- Setup
  - Linux
  - Windows Server 2012R2

Installation

Linux

With RPM Packages There are RPM packages available to Crunchy Data customers through the Crunchy Customer Portal.

If you install the below available packages with RPM, you can continue reading at the Setup section.
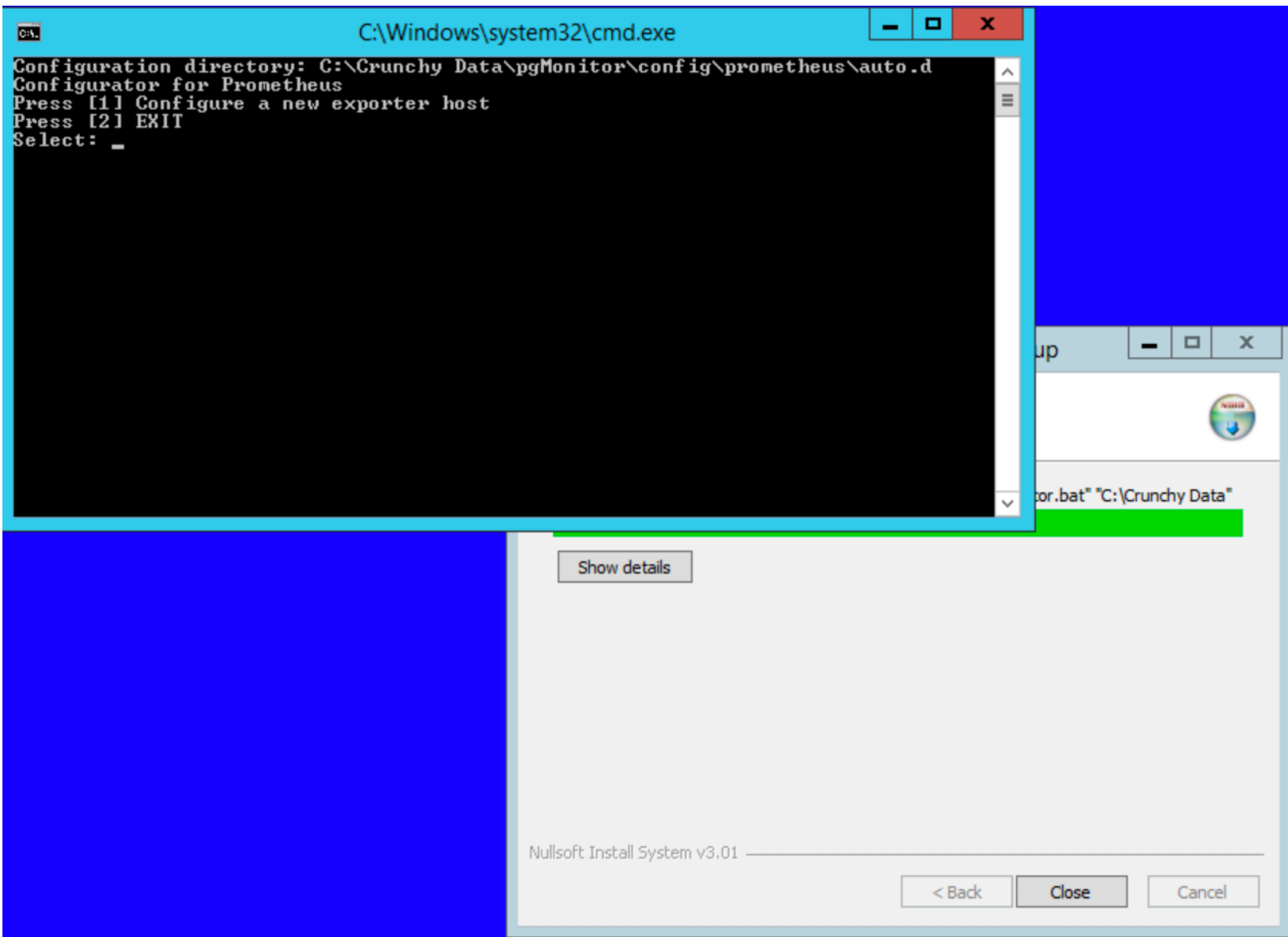
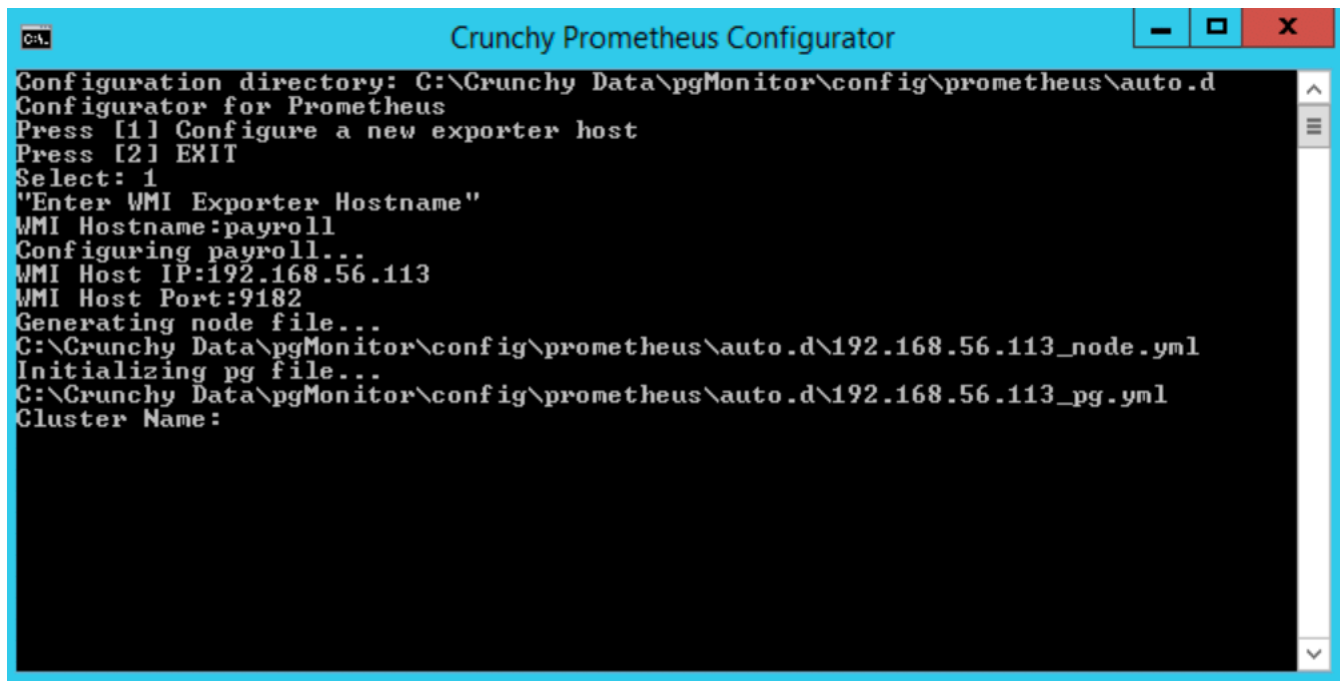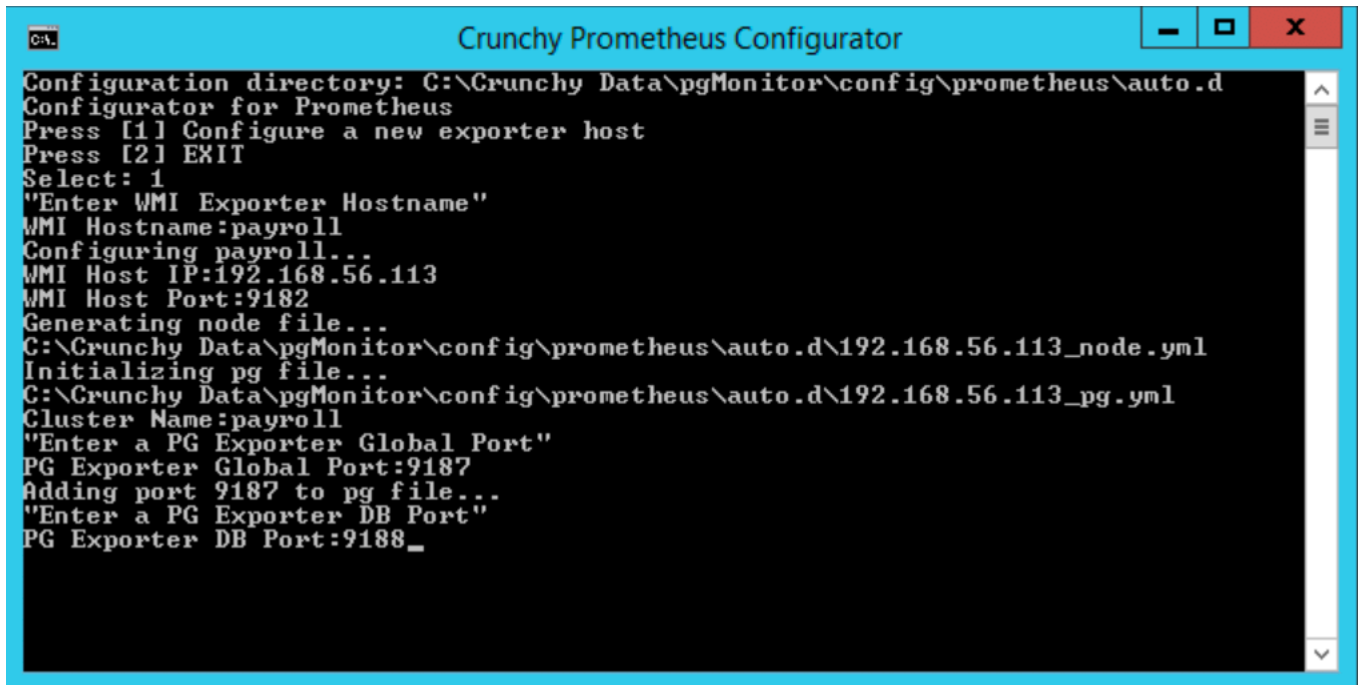Figure 18: server_installer_5.png

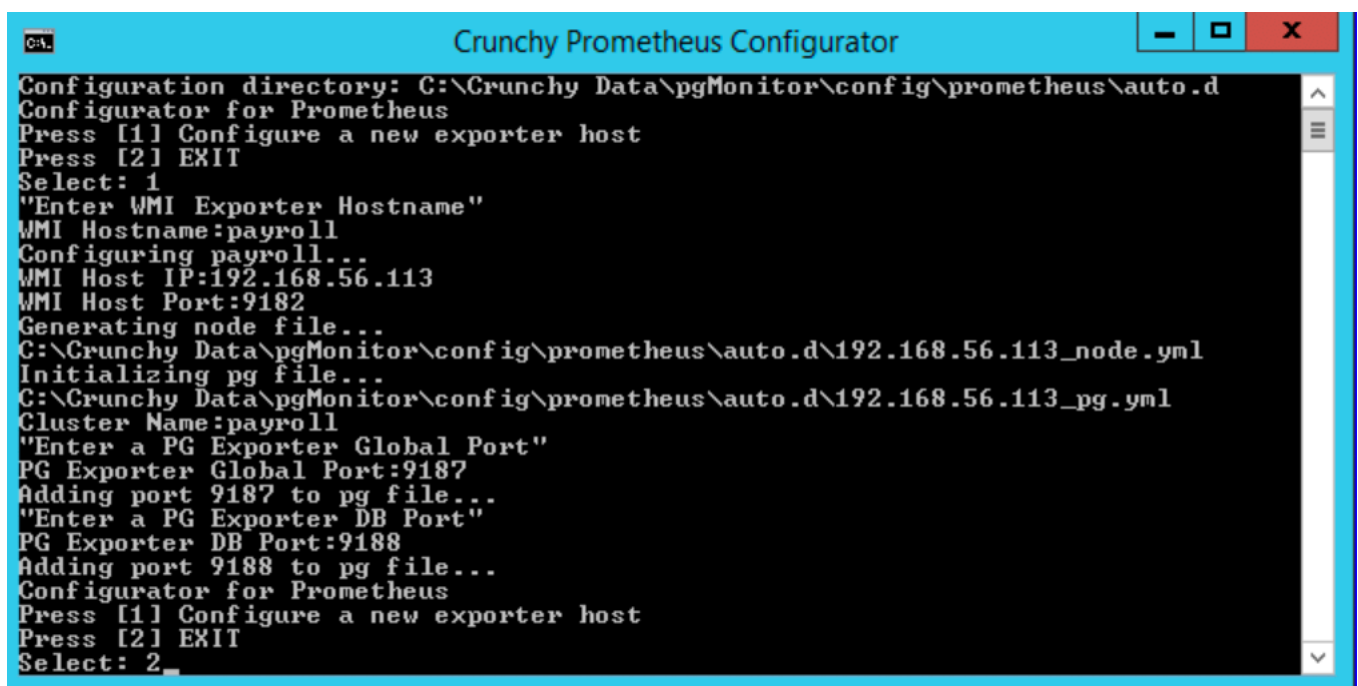Figure 19: server_installer_6.png

Figure 20: server_installer_7.png

Figure 21: server_installer_8.png

Figure 22: server_installer_9.png
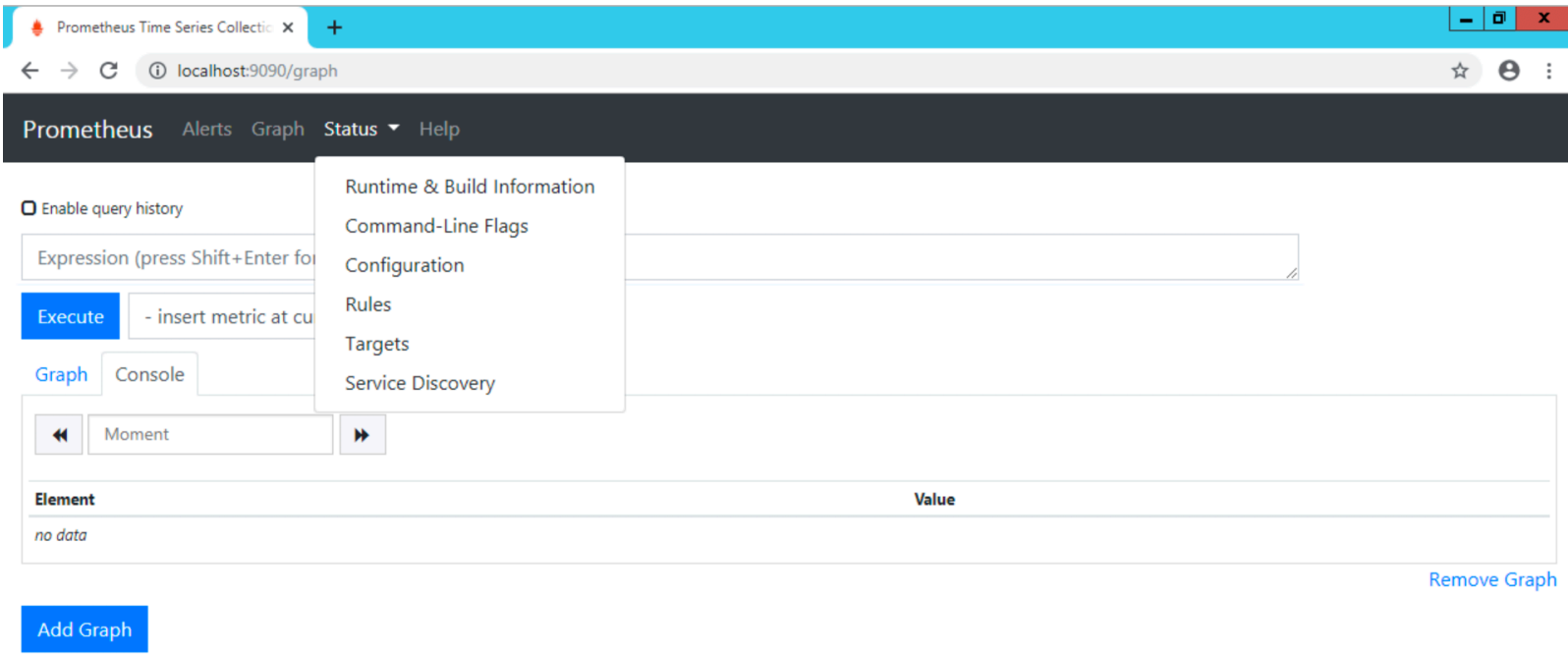
Figure 23: server_installer_10.png

Figure 24: server_installer_11.png

Package Name	Description
grafana	Base package for grafana
pgmonitor-grafana-extras	Crunchy configurations for datasource & dashboard provisioning

Available Packages

Without Packages   Create the following directories on your grafana server if they don't exist:

```
mkdir -p /etc/grafana/provisioning/{datasources,dashboards}
mkdir -p /etc/grafana/crunchy_dashboards
```

pgmonitor Configuration File	System Location
grafana/crunchy_grafana_datasource.yml	/etc/grafana/provisioning/datasources/datasource.yml
grafana/crunchy_grafana_dashboards.yml	/etc/grafana/provisioning/dashboards/dashboards.yml

Review the `crunchy_grafana_datasource.yml` file to ensure it is looking at your Prometheus database. The included file assumes Grafana and Prometheus are running on the same system. DO NOT CHANGE the datasource "name" if you will be using the dashboards provided in this repo. They assume that name and will not work otherwise. Any other options can be changed as needed. Save the `crunchy_grafana_datasource.yml` file and rename it to `/etc/grafana/provisioning/datasources/datasources.yml`. Restart grafana and confirm through the web interface that the datasource was provisioned and working.

Review the `crunchy_grafana_dashboards.yml` file to ensure it's looking at where you stored the provided dashboards. By default it is looking in `/etc/grafana/crunchy_dashboards`. Save this file and rename it to `/etc/grafana/provisioning/dashboards/dashboards.yml`. Restart grafana so it picks up the new config.

Save all of the desired .json dashboard files to the `/etc/grafana/crunchy_dashboards` folder. All of them are not required, so if there is a dashboard you do not need, it can be left out.

Windows Server 2012R2

Grafana and Prometheus are currently both installed together on Windows via the Crunchy Data installer. Please refer to the Prometheus setup guide for installation details. Once installed, follow the configuration steps below.

Upgrading

If you'd like to take advantage of the new provisioning system in Grafana 5 provided by pgmonitor 2.x, we recommend either renaming or deleting your existing datasources and dashboards so there are no issues when the provisioned versions are imported.

When upgrading from pgmonitor 1.x to 2.x, note that many of the system related metric names from node_exporter have had their names changed. The new graphs provided for Grafana 5+ have taken these new names into account. Also, the top level PostgreSQL Overview dashboard no longer uses the ccp_is_ready metric, so you will have to include some new postgres_exporter metrics for that dashboard to work.

Setup

Setup on Linux

Configuration Database   By default Grafana uses an SQLite database to store configuration and dashboard information. We recommend using a PostgreSQL database for better long term scalability. Before doing any further configuration, including changing the default admin password, set the `grafana.ini` to point to a postgresql instance that has a database created for it.

In psql run the following:

```
    CREATE ROLE grafana WITH LOGIN;
    CREATE DATABASE grafana;
    ALTER DATABASE grafana OWNER TO grafana;
    \password grafana
```

You may also need to adjust your `pg_hba.conf` to allow grafana to connect to your database.

In your `grafana.ini`, set the following options at a minimum with relevant values:

```
[database]

type = postgres
host = 127.0.0.1:5432
name = grafana
user = grafana
password = """mypassword"""
```

Now enable and start the grafana service

```
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
sudo systemctl status grafana-server
```

Navigate to the web interface: https://<ip-address>:3000. Log in with admin/admin (be sure to change the admin password) and check settings to ensure the postgres options have been set and are working.

Datasource & Dashboard Provisioning

Grafana 5.x provides the ability to automatically provision datasources and dashboards via configuration files instead of having to manually import them either through the web interface or the API. Note that provisioned dashboards can no longer be directly edited and saved via the web interface. See the Grafana documentation for how to edit/save provisioned dashboards: http://docs.grafana.org/administration/provisioning/#making-changes-to-a-provisioned-dashboard. If you'd like to customize these dashboards, we recommend first adding them via provisioning then saving them with a new name. You can then either manage them via the web interface or add them to the provisioning system.

The extras package takes care of putting all these files in place. If you did not use the crunchy package to install grafana, see the additional instructions above. Once that is done, the only additional setup that needs to be done is to set the "provisioning" option in the `grafana.ini` to point to the top level directory if it hasn't been done already. If you're upgrading from Grafana 4.x to 5.x, you will have to add the "provisioning" option to the `[paths]` section of the `grafana.ini` file. Once that is done, just restart grafana and all datasources and dashboards should appear.

```
[paths]
provisioning = /etc/grafana/provisioning
```

Setup on Windows Server 2012R2

Grafana is installed and configured to use the default Crunchy-provided Prometheus datasource and the Crunchy-provided dashboards automatically. We simply need to set the default admin user's password and verify that the dashboards are functional:

1. Load the Grafana UI in your browser by visiting http://localhost:3000:

2. After entering 'admin' as both the user and password, you are prompted to change the password before continuing:

3. After changing the password, you are logged in and taken to the Home Dashboard. Here you can see the first three items are stricken-through indicating they are already completed. You can now optionally add users or plugins to you installation (we won't cover either of these steps):

4. Click the dashboard selector in the top left of the screen that currently has the four squares, the word Home, and the downward-facing triangle to see the list of installed dashboards:

5. Select the 'Overview' dashboard from the available choices. This is the 'main' dashboard:

6. Click on the 'OS' square to load the 'OS Overview' dashboard:

7. This dashboard shows you whether the host(s) you're monitoring are UP or DOWN. Click on a given host to load the 'OS Details' dashboard for that host:

8. From the 'Overview' dashboard, clicking the 'PostgreSQL' square will load the 'PostgreSQL Overview' dashboard showing which monitored PostgreSQL instances are up or down:

9. And finally, clicking any of the PostgreSQL hosts will take you to the 'PostgreSQL Details' dashboard for that PostgreSQL instance:
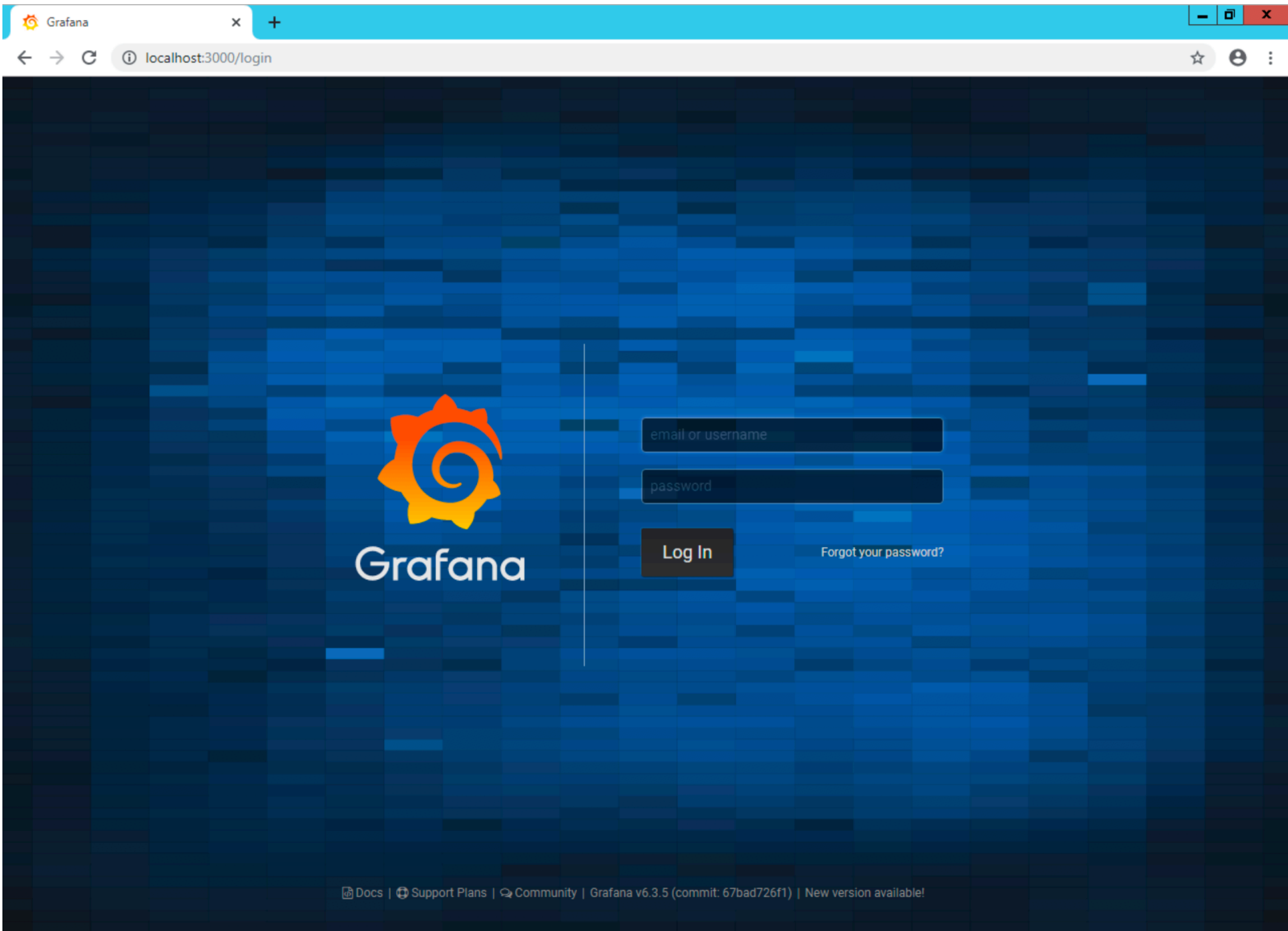
Figure 25: server_installer_12
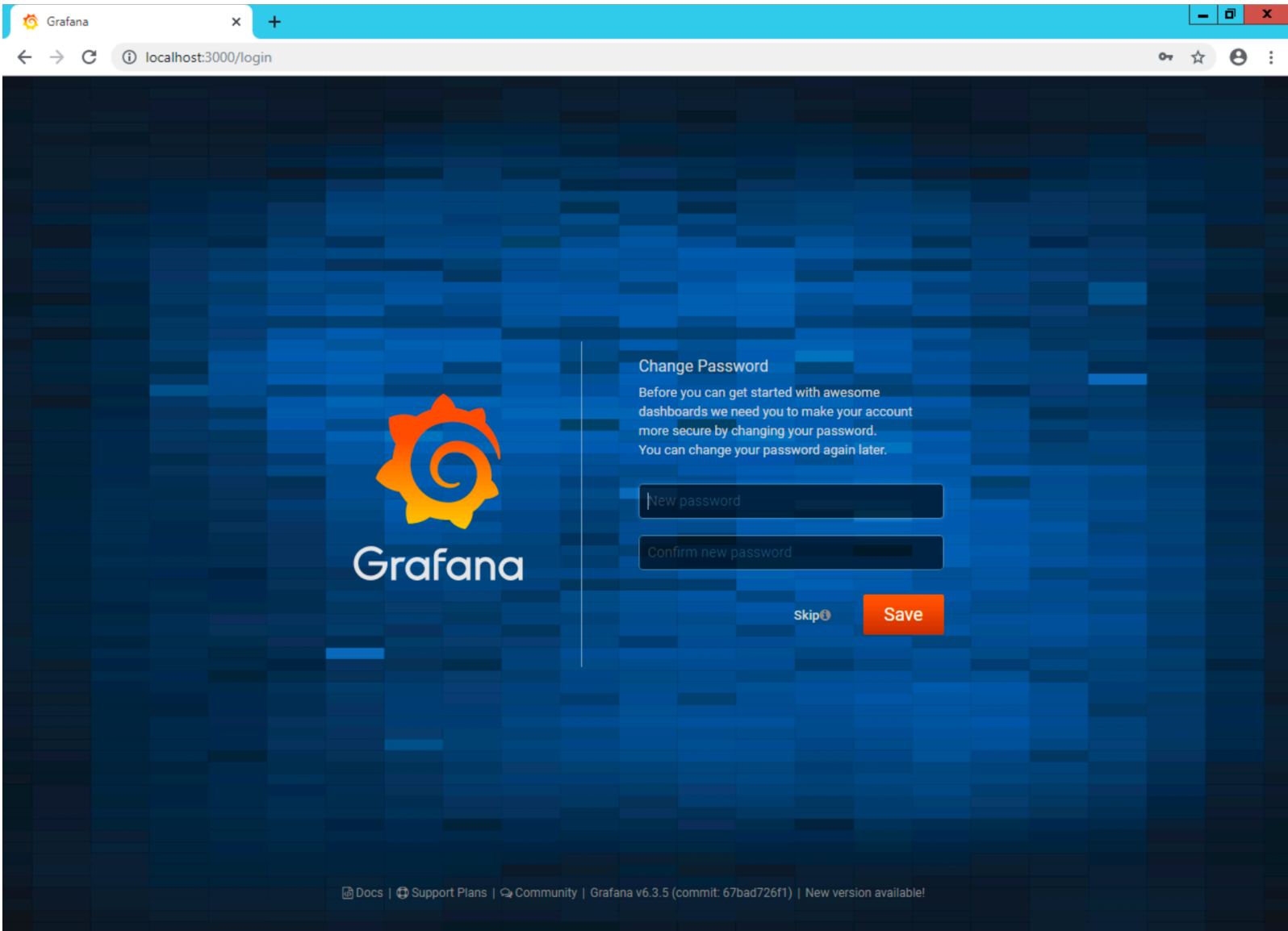
Figure 26: server_installer_13
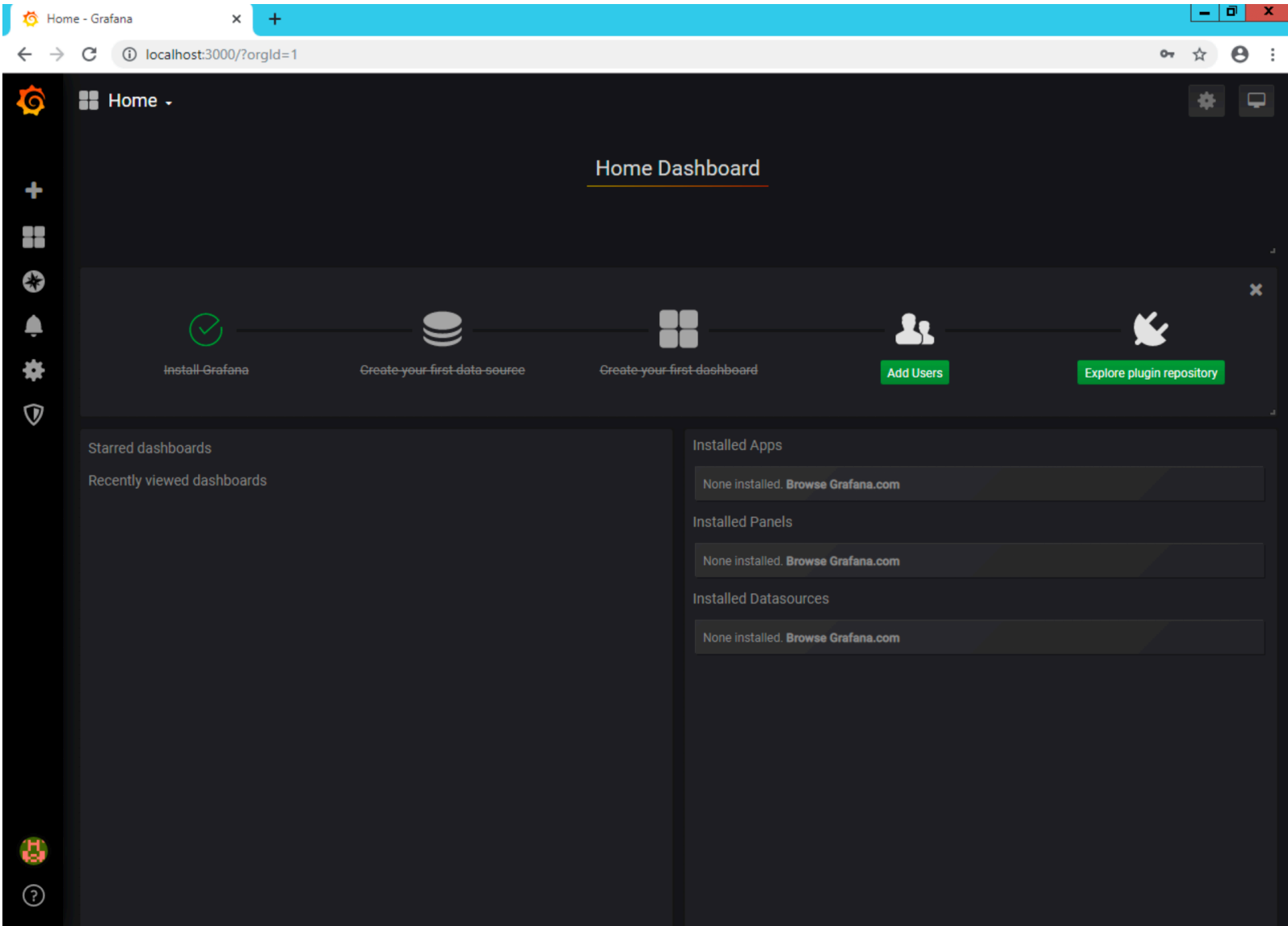
Figure 27: server_installer_14
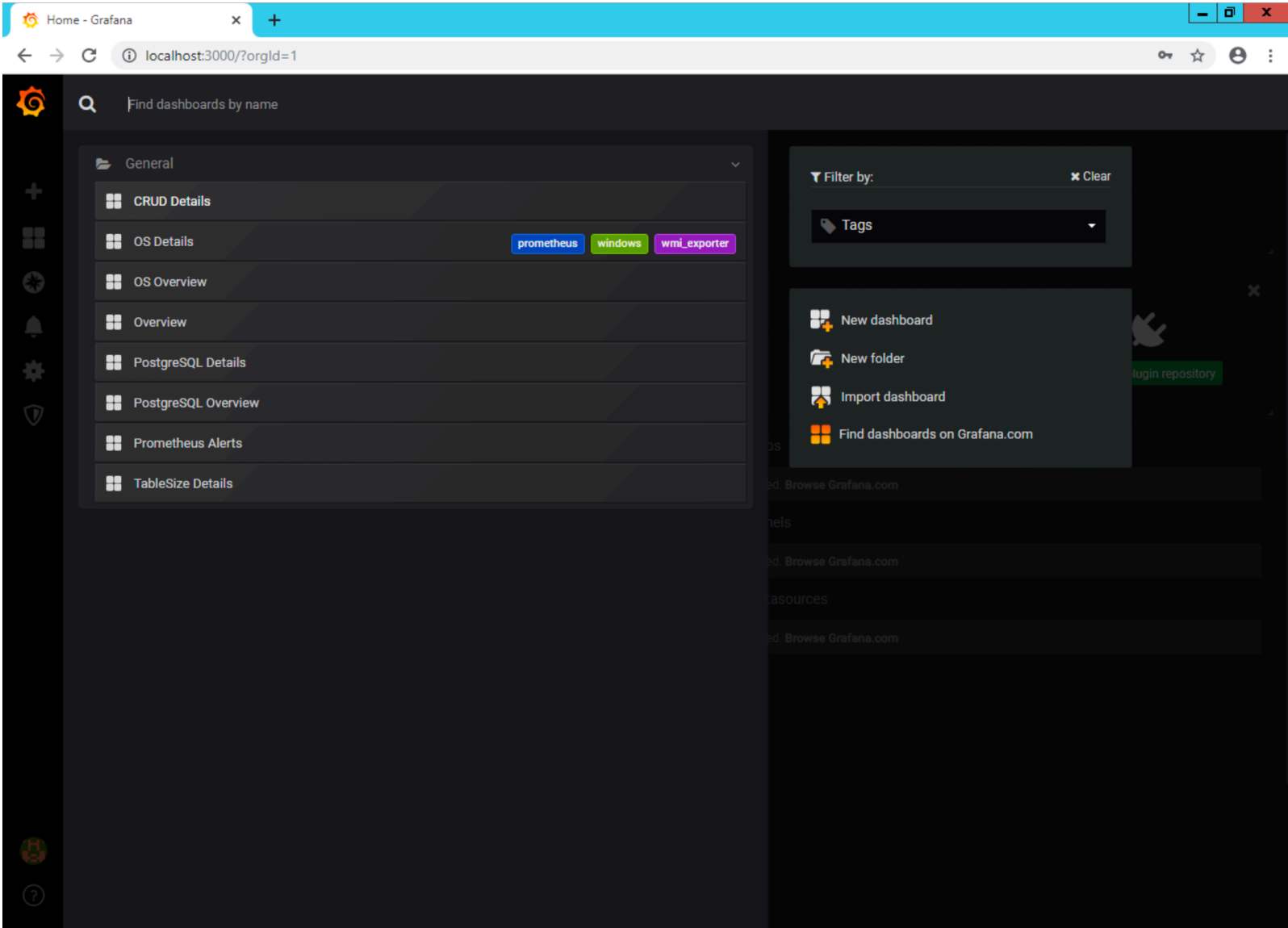
Figure 28: server_installer_15
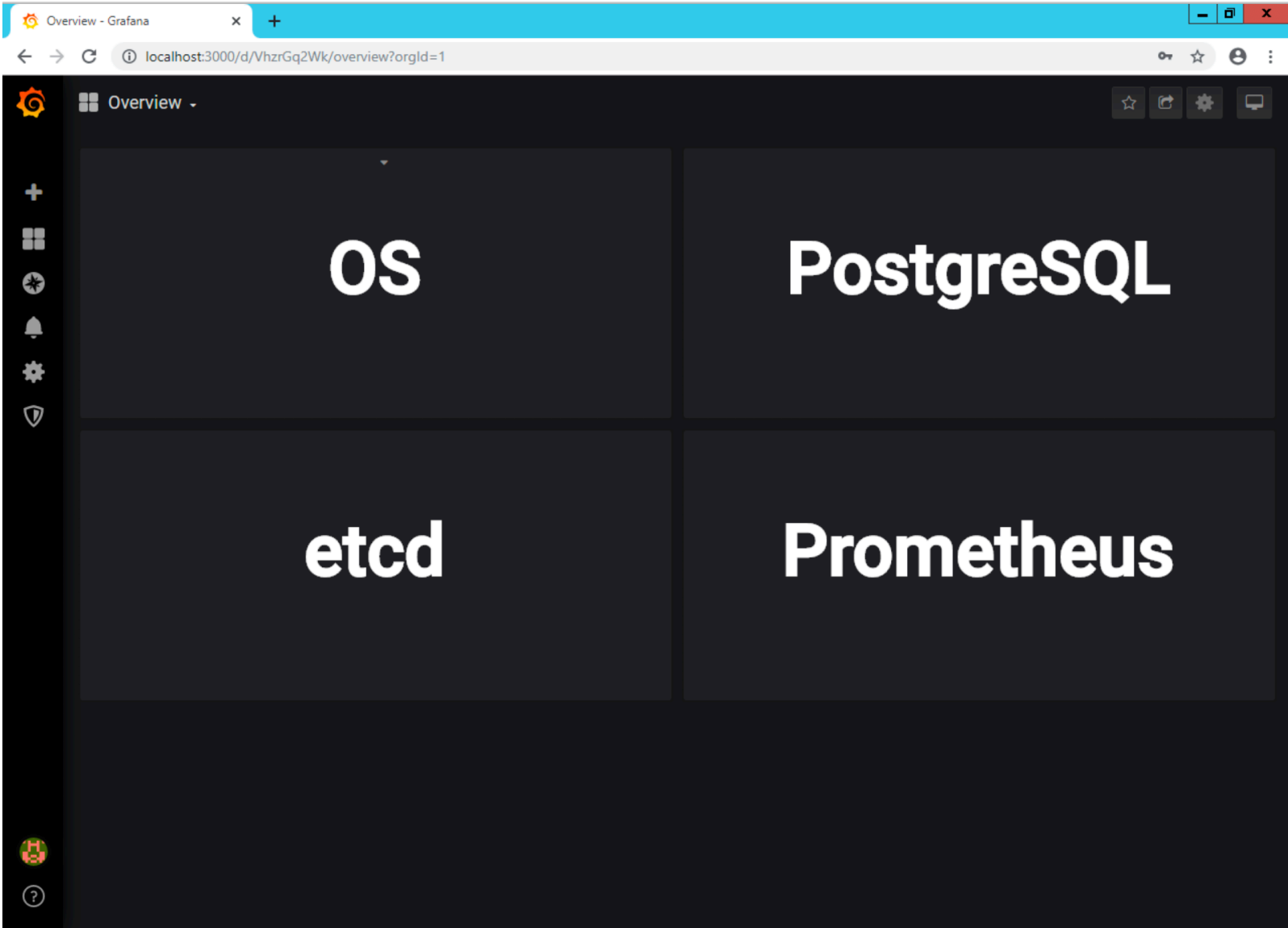
Figure 29: server_installer_16
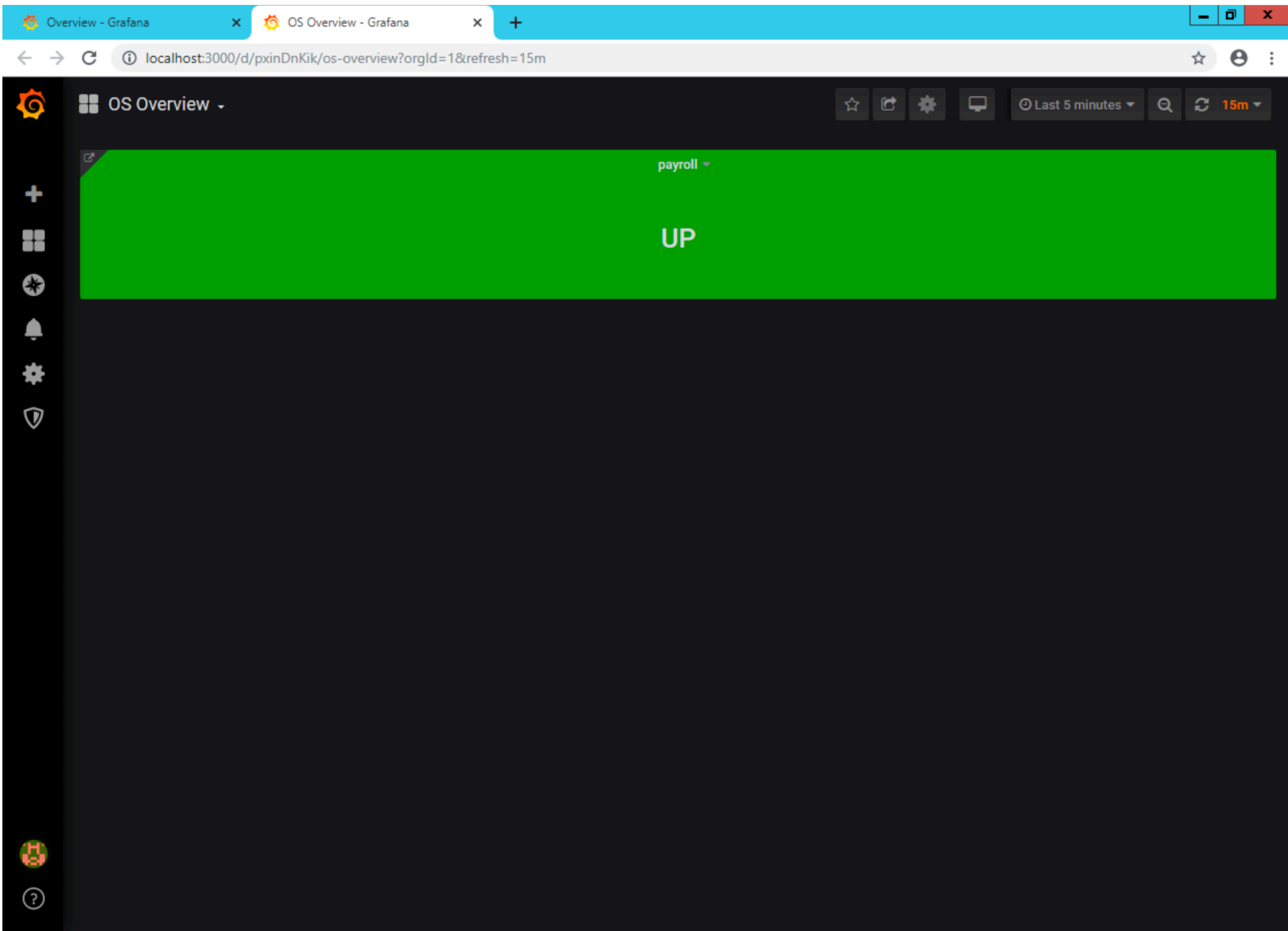
Figure 30: server_installer_17
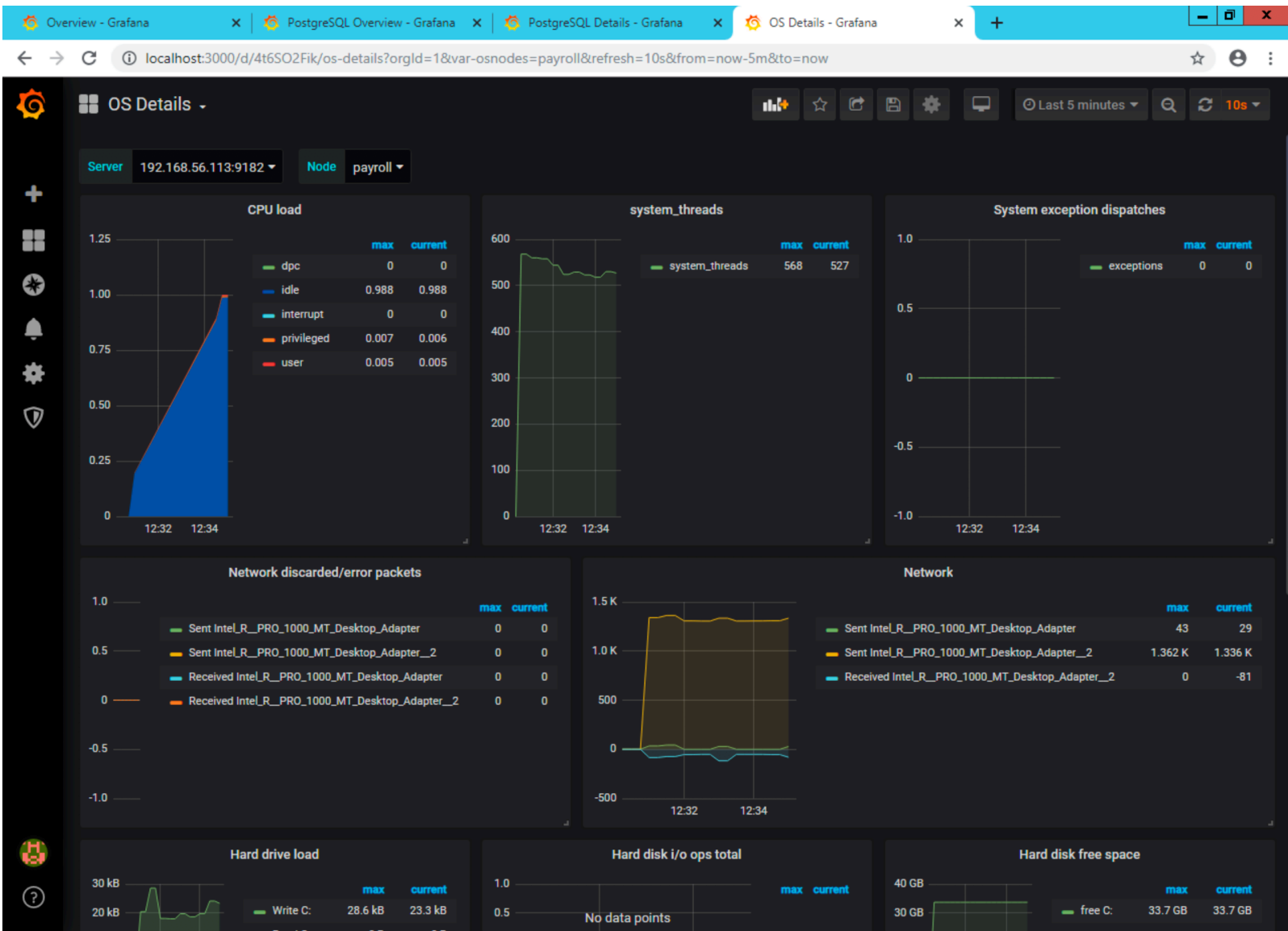
Figure 31: server_installer_18
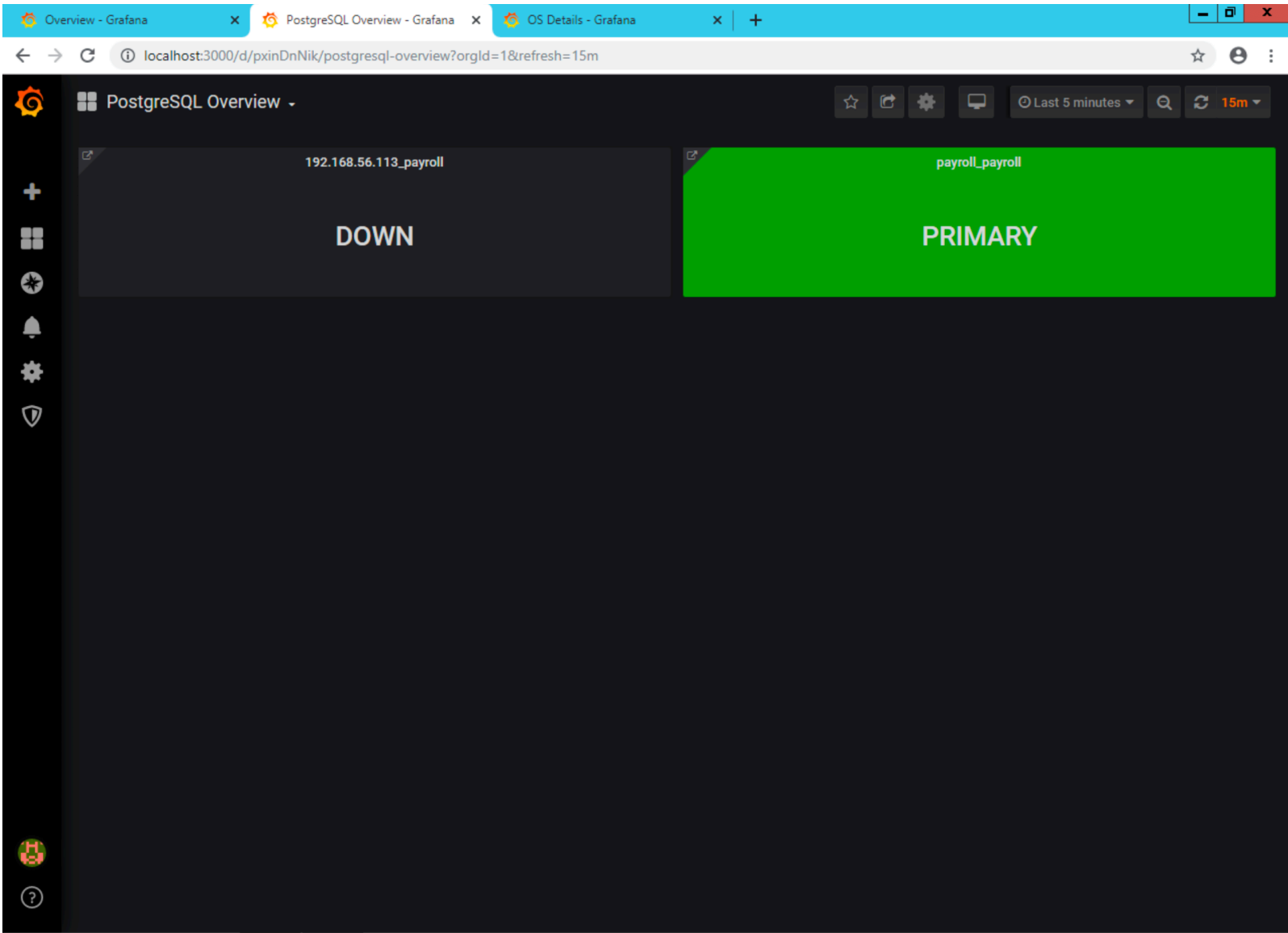
Figure 32: server_installer_19
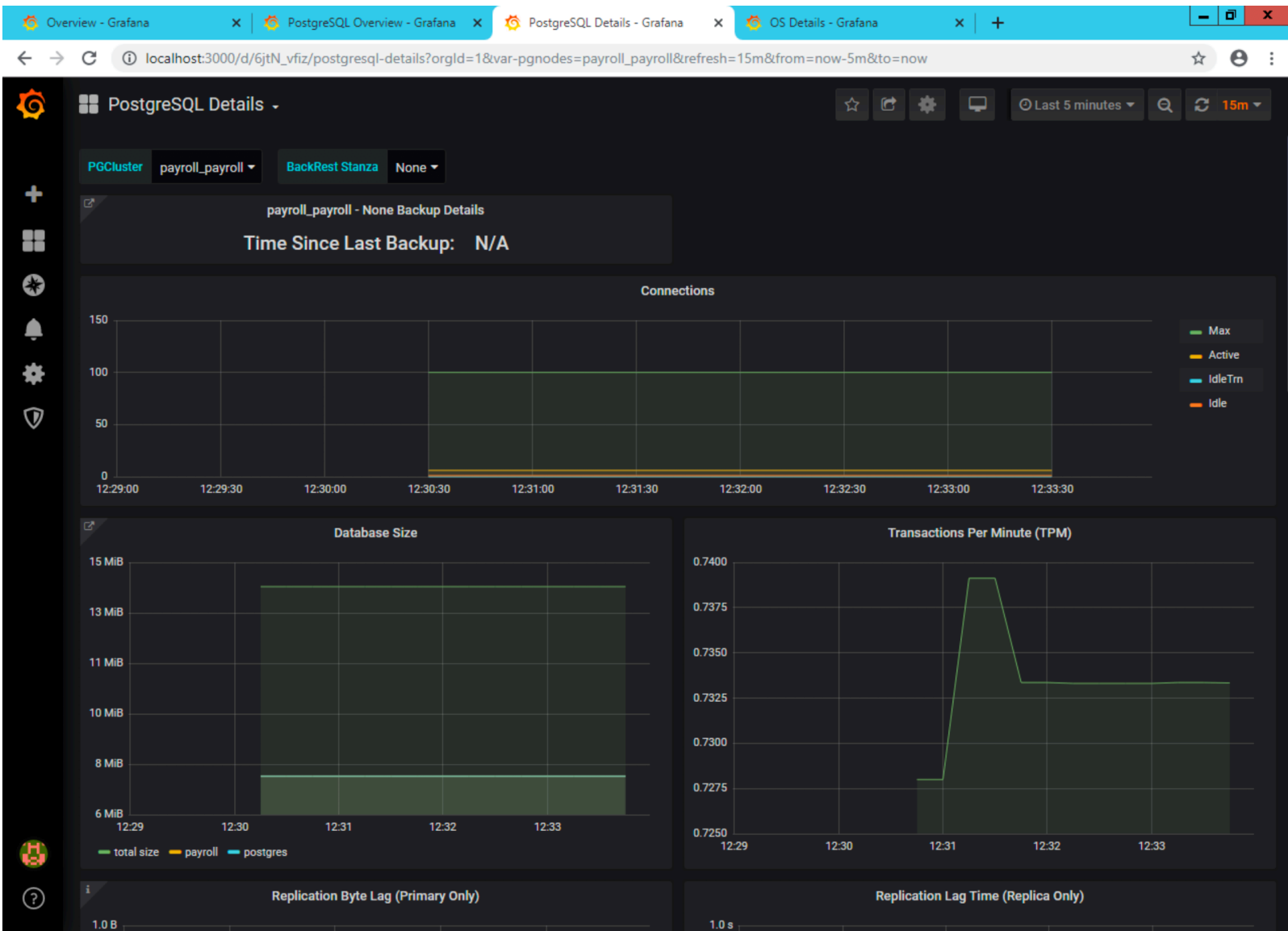
Figure 33: server_installer_20

4.3

New Features

Bug Fixes

- Fixed syntax error in example prometheus alert rules file for postgresql for the pending restart rule.

Non-backward Compatible Changes

Manual Intervention Changes

- Renamed metric `ccp_postmaster_runtime_start_time_seconds` to `ccp_postmaster_uptime_seconds`. Both metrics report the same value, so they are currently duplicates. Note the old metric name has not yet been dropped and will still work, but it will be dropped in an upcoming version of pgMonitor.

- For PostgreSQL 9.5 & 9.6, renamed metric `ccp_wal_activity_count` to `ccp_wal_activity_total_size_bytes`. The actual value being returned has always been the total size in bytes, so the previous name was misleading. PostgreSQL 10+ already had the metric with the proper bytes size name. Note the old metric name has not yet been dropped and will still work, but it will be dropped in an upcoming version of pgMonitor.

4.2

New Features

- Add support for PostgreSQL 12
- Added new metrics (all PG versions):

    - ccp_postmaster_uptime - time in seconds since last restart of PG instance. Useful for monitoring for unexpected restarts.
    - ccp_pg_settings_checksum - monitors for changes in pg_settings

- Added new metrics (PG 9.5+ only)

    - ccp_settings_pending_restart - monitors for any settings in pg_settings in a pending_restart state

- Added new metrics (PG 10+ only)

    - ccp_pg_hba_checksum - monitors for changes in pg_hba.conf

- Added new metrics (PG 12+ only)

    - ccp_data_checksum_failure - monitors for any errors encountered for databases that have data file checksums enabled

Bug Fixes

- Use proper comparison operators in all Grafana dashboards that are using Multi-value variables.
- Change to using label_values() function on Grafana dashboard template variables. Ensures proper values in all dropdown menus are shown
- Remove changing background color of the pgBackRest panel in the PG_Details Grafana dashboard

Non-backward Compatible Changes

- New minimum required version of Grafana is now 6.5. All Grafana dashboards have been re-exported to ensure their settings are consistent and compatable with that version.

Manual Intervention Changes

- In order to use the new metrics that are available, the setup_##.sql script must be run again for your relevant version of PostgreSQL. Then all postgres_exporters services must be restarted.
- The only new rule that has been enabled by default in the Crunchy provided Prometheus rules file is `ccp_settings_pending_restart`. All other new metrics have example rules in the same file but they are commented out. Please adjust them as needed before uncommenting and using them.

4.1

- Fixed bug in PGBouncer Grafana dashboard for the Server Connection Counts Per Pool showing zero data
- Fixed Windows prometheus config file to use proper wildcard to pick up .yml files.
- Renamed Prometheus target example file to include yml extention to better ensure it is not missed. ReplicaOS.example to ReplicaOS.yml.example
- Fixed documentation to display pictures properly.

4.0

New Features

- Add pgbouncer monitoring support
  - Requires new `pgbouncer_fdw` extension provided by Crunchy Data: https://github.com/CrunchyData/pgbouncer_fdw
  - New query file can be included in QUERY_FILE_LIST: queries_pgbouncer.yml
  - New Grafana dashboard: PGBouncer.json
- Minimum version of postgres_exporter required is now 0.5.1
  - Allows connecting to multiple databases from a single exporter, however only one query file can be set per exporter service
  - If statistics are needed for per-database metrics on more than one database, recommend running a second exporter (example included as `sysconfig.postgres_exporter_pg##_per_db`) that connects to all dbs where such stats are required using separate custom query file. Leave the main exporter service to only collect global metrics from one database (preferably `postgres`).
  - DO NOT yet recommend using new `--auto-database-discovery` feature. Currently tries to connect to template databases which is never recommended.
- Added backup sizes to pgBackRest metrics that are collected by default
  - Updated pgBackRest grafana dashboard to include size graphs. Also added per-stanza dropdown filter to the top of dashboard for better readability when there are many backups.
- Added new metric to check what version of PostgreSQL the exporter is currently running on (`ccp_postgresql_version_current`).

Non-backward Compatible Changes

- Version 0.5x of postgres_exporter adds a new "server" label to all custom query output metrics. This breaks several single panel graphs that pgmonitor uses in Grafana (PG Overview, PGBackrest).
  - If upgrading, the update for the prometheus extras package must be done before upgrading to the new version of postgres_exporter. Otherwise the "server" label can cause duplication of some metrics.
  - Added a metric_relabel_configs line to the crunchy-prometheus.yml file to filter out this new label. If you are upgrading, you may have to manually add this to your own prometheus config. The package update will only automatically add this if you haven't changed the default file. Otherwise the new settings will be contained in a crunchy-prometheus.yml.rpmnew file in the package install location.

Manual Intervention Changes

- See Non-backward Compatible Changes section for update that may need to be done to prometheus config.
- Changed default DATA_SOURCE_NAME value for postgres_exporter to use the local socket for the ccp_monitoring role. This should allow the exporter to work using peer authentication, which is the default authentication method allowed by most rpm/deb provided postgres packages. This should not change any existing installations, but may affect new deployments due to new default behavior.
- Split Prometheus crunchy-alert-rules.yml file into separate node & postgres alert files to allow for more flexible rule management.
  - By default alert rules files are now looked for in `/etc/prometheus/alert-rules.d/`. Any alert files located in this folder upon restart/reload will then be picked up automatically.
  - Renamed alert files in repository to have additional .example file extension.
  - IMPORTANT UPGRADE NOTE: If upgrading with packages, prometheus may change and point to the new rules location causing your active alerts to change. Your custom alert rules have not been lost, just ensure your desired rules file(s) are moved to the new location for future compatability.
  - Changed metric name `ccp_backrest_last_runtime` to `ccp_backrest_last_info` to reflect that it is no longer only collecting runtime stats. Note that due to metric name change, you will appear to have lost runtime history in the new grafana dashboard. The data is still there under the old metric name and can be added back as an additional data point if needed.
  - Fixed prometheus disk sizing rules to properly include ext filesystems (ext[234]). The correct syntax for the sizing-based rules is contained in the example rule files that the package provides. You will need to copy them to your current rule files if applicable.

Bug Fixes

- Disable pg_settings values that are exported by default with postgres_exporter. Fixes issue with multi-dsn support in 0.5.1 of postgres_exporter. If settings are desired as output from exporter, it is recommended to add a custom query.

- Fixed postgres_exporter service file to better parse out the destination query file name (exporter/postgres/crunchy-postgres-exporter@.service or exporter/postgres/crunchy-postgres-exporter-pg##-el6.service). Previously if any additional options were added to the OPT variable in the sysconfig, the service could throw errors on start. If you've customized your service file, please make note of changes for future compatability.

- Update Grafana Overview dashboard to be compatible with Grafana 6.4+

3.2

- Fixed postgres_exporter service in EL6 (Redhat/CentOS) to properly use the backrest throttle environment variable in sysconfig (Github Issue #107).

3.1

- Fix broken links in Grafana OS & PG Overview Dashboards
- Updated UPGRADE steps in 3.0 release notes for new exporter service name setup. Need to re-enable service with new name and manually remove old symlink files.
- Update documentation for exporter setup to use new service names

3.0

- New minimum version requirements for software that is part of pgmonitor are as follows, including links to release notes:
    - Prometheus: 2.9.2 - https://github.com/prometheus/prometheus/releases
    - Alertmanager: 0.17.0 - https://github.com/prometheus/alertmanager/releases
    - Grafana: 6.1.6 (major version change from 5.x) - https://community.grafana.com/t/release-notes-v6-1-x/15772
    - node_exporter: 0.18.0 - https://github.com/prometheus/node_exporter (Note breaking changes for some metrics. None of those broken are used by default in pgmonitor).
- The service file for postgres_exporter provided by pgmonitor has been renamed to make it more consistent with typical systemd service names.
    - IMPORTANT: See upgrade notes below about changes to sysconfig file before restarting service!
    - Only applies to systemd file for RHEL/CentOS 7
    - Changed crunchy_postgres_exporter@.service to crunchy-postgres-exporter@.service (underscores to dashes).
    - Note that you will need to use the new service name to interact with it from now on. This requires enabling the new service name and restarting it:
        * `systemctl enable crunchy-postgres-exporter@postgres_exporter_pg11`
        * `systemctl restart crunchy-postgres-exporter@postgres_exporter_pg11`
    - Due to the removal of the old service file, you cannot use systemctl to disable the old service. Instead just remove the symlinks manually:
        * 'rm /etc/systemd/system/multi-user.target.wants/crunchy_postgres_exporter@*
- The single query.yml file used by postgres_exporter to use Crunchy's custom queries is now dynamically generated automatically upon service start/restart.
    - A new variable, QUERY_FILE_LIST, is now set in the sysconfig file for the service. It is a space delimited list of the full paths to all query files that will be concatenated together. See sysconfig file for several examples and a recommended default to set.
    - This now ensures that any updates to desired query files will be automatically applied when the package is updated and the service is restarted without having to manually rebuild the query.yml file.
    - This new variable is not required and you can continue to manually manage your queries.yml file. Ensure that the QUERY_FILE_LIST variable is not set if this is desired.
    - UPGRADE NOTES:
        * Backup your current queries.yml file.
        * If you have not modified the default sysconfig file for your postgres_exporter service (/etc/sysconfig/postgres_exporter_pg## updating to 3.0 will overwrite your current sysconfig file and put the default QUERY_FILE_LIST value in place, possibly overwriting your current queries.yml file. Again, please ensure you backup your current queries.yml file and then set the QUERY_FILE_LIST variable appropriately to dynamically generate your queries file for you in the future. Or unset the variable and continue managing it manually.

* If you have modified your sysconfig file from what the package provides, it will not be overwritten and a new sysconfig file with an `.rpmnew` extension will be created. You can reference this .rpmnew file for how to update your sysconfig file to take advantage of the new QUERY_FILE_LIST option.
  * Ensure all postgres_exporters you have running set the QUERY_FILE_LIST properly if using it. Especially if multiple exporters are using the same query file.
- Prometheus targets for pgmonitor provided exporters (postgres_exporter & node_exporter) have had labels added to them for use in pgmonitor provided Grafana Dashboards.
  - Added new label `exp_type` (export type) in prometheus targets to better distinguish OS and Postgres metrics in Prometheus. Possible current values are `pg` or `node`.
  - UPGRADE NOTES: This new label must be applied to your Prometheus target files if you are using the Grafana dashboards provided by pgmonitor. Note that if you previously defined node and postgres_exporter targets under a single target, you will now need to separate them, keeping the same job name for both. See example target files provided in package/repo for how to apply new label (Ex. ProductionDB.yml.example & ProductionOS.yml.example).
  - If you are not using the pgmonitor provided Grafana dashboards, these new labels are optional.
- Grafana Dashboards Updates
  - New dashboards require at least Grafana 6.x.
  - UPGRADE NOTES: Once new Prometheus label (mentioned above) is applied, dashboard provisioning should take care of updating all dashboards once the new ones are in place. Note that all dashboards provided by pgmonitor 3.0+ now assume this new label and will not work until the Prometheus exp_type label is added.
  - Renamed dashboard files for better naming consistency. Dashboard titles also updated accordingly.
    * UPGRADE NOTES: If installing from package, it will take care of care of renaming dashboard files. Otherwise, dashboards have been renamed as follows below. Ensure old files are renamed/removed to avoid duplicating/breaking current dashboards. Easiest manual update method is to remove all dashboards provided by pgmonitor and copy all new ones back. Provisioning will then take care of updating things for you.
    * renamed: BloatDetails.json -> Bloat_Details.json
    * renamed: FilesystemDetails.json -> Filesystem_Details.json
    * renamed: PostgreSQLDetails.json -> PG_Details.json
    * renamed: PostgreSQL.json -> PG_Overview.json
    * renamed: TableSize_Detail.json -> TableSize_Details.json
  - Dashboard names have been updated to match with new naming consistency. If you had direct links to dashboards, these may need to be updated.
  - Split OS Metrics into their own dashboard separate from PG Metrics.
  - Added link to PGbackrest dashboard to top of Postgres Details Dashboard. Link shows time since last successful backup (any type) for that target system.
  - Added new OS Details dashboard
  - Added new etcd dashboard
  - Add new Top Level Overview dashboard that links to all other Overview dashboards
  - Set default refresh rate for most dashboards to 15 minutes.
  - Obsolete "jobname" grafana variable in all dashboards. Add new grafana variables pgnodes, osnodes that use the new labels added in prometheus targets notded above.
- New configuration option for postgres_exporter sysconfig file to control PGBackrest refresh rate
  - PGBACKREST_INFO_THROTTLE_MINUTES
  - This is the value, in minutes, passed along to the monitor.pgbackrest_info() function in all backrest checks
  - Default is 10 minutes

2.4

- UPGRADE NOTE: All exporter issues below can be fixed by re-running the setup_pg##.sql file for your major version of postgres. For the pgbackrest fix, you will also need to update the queries.yml file for the exporter to include the new queries found in the queries_backrest.yml file.
- Fixed several issues with pgbackrest monitor in postgres_exporter that was included in pgmonitor v2.3
- Fixed incorrect data being returned by monitor query on PostgreSQL 9.6 and earlier. The same, latest backup time was being returned for all stanzas instead of returning the time per stanza.
- Fixed backrest query causing the postgres_exporter to hang and cause all metric output to stop.
- Fixed backrest monitor to work with larger amount of data being returned by the "pgbackrest info" command. Previously, once returned data size reached a certain point, would cause a "missing chunk" error.
- Added a parameter to the function that is called to control how often the underlying info command is actually run. On systems with high backup counts, info can be a slightly more expensive call. This helps to control that, no matter what the scrape interval of prometheus is set to. Default is to get new data every 10 minutes, otherwise just queries from an internal table that stores the last info run.

- Backrest monitoring can now be run on replicas as well, but cannot update the current backup status since that requires writing to the database. This is mostly to enable monitoring setups to be consistent between primary/replica in case of failover.
- Fixed issue with ccp_sequence_exhaustion metric that would cause postgres_exporter output to hang if any table that contained a sequence was dropped during a long running transaction.
- Added new metric (ccp_replication_slots) and alert (PGReplicationSlotsInactive) for monitoring replication slot status. New metric and alert can be found in queries_pg##.yml and crunchy-alert-rules.yml respectively.
- Added lock_timeout of 2 minutes to the ccp_monitoring role. Avoids monitoring causing any extensive lock interference with normal database operations.
- Added Grafana Dashboard for PGBackrest status information.
- Fixed lines being hidden in the "Total Bloat %" graph in BloatDetails Grafana dashboard.
- Removed unnecessary drilldown link in Total Bloat % graph in BloatDetails Grafana dashboard.

2.3

- Fixed bug in Prometheus alerts that was causing some of them to be stuck in PENDING mode indefinitely and never firing. This unfortunately removes the current alert value from the Grafana Prometheus Alerts dashboard.

- If you can't simply overwrite your current alerts configuration file with the one provided, remove the following option from every alert: `alert_value: '{{ $value }}'`

- Added feature to monitor pgbackrest backups (https://pgbackrest.org)

- Separate metrics exist to monitor for the latest full, incremental and/or differential backups. Note that a full will always count as both an incremnetal and diff and a diff will always count as an incremental.

- Another metric can monitor the runtime of the latest backup of each type per stanza.

- Run the setup_pg##.sql file again in the database that your exporter(s) connect to to install the new, required function: "monitor.pgbackrest_info()". It has security definer so execution privileges can be granted as needed, but it must be owned by a superuser.

- New metrics are located in the exporter/postgres/queries_backrest.yml file. Add the one(s) you want to the main queries file being used by your currently running exporter(s) and restart.

- Example alert rules for different backup scenarios have been added to the prometheus/crunchy-alert-rules.yml file. They are commented out to avoid false alarms until valid backup settings for your environment are in place.

- Added new feature to monitor for failing archive_command calls.

    - New metric "ccp_archive_command_status" is located in exporter/postgres/queries_common.yml. Add this to the main queries file being used by your currently running exporter(s) and restart.
    - A new alert rule "PGArchiveCommandStatus" has been added to the prometheus/crunchy-alert-rules.yml file.

- Added new feature to monitor for sequence exhaustion

    - Requires installation of a new function located in the setup_pg##.yml file for your relevant major version of PostgreSQL. Must be installed by a superuser.
    - New metric "ccp_sequence_exhaustion" located in exporter/postgres/queries_common.yml. Add this to the main queries file being used by your currently running exporter(s) and restart. A new alert rule "PGSequenceExhaustion" has been added to the prometheus/crunchy-alert-rules.yml file.

- The setup_pg##.sql file now has logic to avoid throwing errors when the ccp_monitoring role already exists. Also always attempts to drop the functions it manages first to account for when the function signature changes in ways that OR REPLACE doesn't handle. All this allows easier re-running of the script when new features are added or used in automation systems. Thanks to Jason O'Donnell for role logic.

2.2

- Fixed broken ccp_wal_activity check for PostgreSQL 9.4 & 9.5. Updated check is located in the relevant exporter/postgres/-queries_pg##.yml file
- Fixed broken service files for postgres_exporter on RHEL6 systems.
- Removed explicit "public" schema in ccp_bloat_check query so that it will properly use the search_path in case bloat tables were installed in another schema
- Removed query files for PostgreSQL versions no longer supported by pgmonitor (9.2 & 9.3)

2.1

- **IMPORTANT UPGRADE NOTE FOR CRUNCHY PACKAGE USERS**: In version 2.0, the Crunchy provided extras for node_exporter were split out from the pgmonitor-pg##-extras package. A dependency was kept between these packages to make upgrading easier. For 2.1, the dependency between these packages has been removed. When upgrading from 1.7 or earlier, if you have node_exporter and postgres_exporter running on the same systems, ensure that you install the separate pgmonitor-node_exporters_extras package after the update. See the README for the full package name(s).

- Minimum required versions of software used in pgmonitor have been updated to:

- Prometheus 2.5.0

- Prometheus Alertmanager 0.15.3

- postgres_exporter 0.4.7 (enables full PG11 support)

- Grafana 5.3.4.

- Fixed Grafana data source to use the "proxy" mode instead of "direct" with default install. Should fix connection issues encountered during default setup between Grafana & Prometheus.

- Renamed functions_pg##.sql file to setup_pg##.sql to better clarify what it's for (and because it's not just functions).

- Added ccp_wal_activity metric to help monitor WAL generation rate.

- For all PG versions, provides total current size of WAL directory. For PG10+, it also provides the size of WAL generated in the last 5 minutes

- Note that for PG96 and lower, a new security definer function must be added (can just run setup_pg##.sql again).

- New metric definition is located in the queries_pg##.yml file.

- No default rules have been added since this is very use-case dependent.

- Improved accuracy of "Idle In Transaction" monitoring times in PostgreSQL. Base the time measured on the state change of the session vs the total transaction runtime.

- Split setup_pg92-96.sql and queries_pg92-96.sql into individual files per major version.

- Added commented out example prometheus alert rule for checking if a postgres database has changed from replica to primary or vice versa. Must be set on a per system basis since you have to tell it if a system is supposed to be a primary or replica.

- Removed pg_stat_statements prometheus metric and security definer function from setup script. We highly recommend having pg_stat_statements installed on a database, and we still include its installation in the documentation, but we currently don't have any useful metric recommendations from it to collect in prometheus.

- Added some default filters for the bloat check cronjob to avoid unnecessary waste in the prometheus storage of bloat metrics.

- Update documentation.

2.0

- Recommended version of Prometheus is now 2.3.2. Recommended version of Alertmanager is 0.15.1. Recommended version of postgres_exporter is 0.4.6.
- Upgrade required version of node_exporter to minimum of 0.16.0. Note that many of the metrics that are used in Grafana and Prometheus alerting have had their names changed.
- This version adds these new metrics into Grafana graphs without removing the old metric names on most, but not all, graphs. This allows trending history to be kept. Note that line colors will change in graphs and legend names will be duplicated until the old metric data is expired out.
- Prometheus alerts have been set to use the new metric names since the alerts are based only on recent values.
- IMPORTANT: A future pgmonitor update will remove these old metric names from Grafana graphs, so please ensure these changes are accounted for in your architecture.
- See full release notes for 0.16.0 - https://github.com/prometheus/node_exporter/releases/tag/v0.16.0
- The postgres_exporter service no longer uses a symlink in /etc/sysconfig to point to a default "postgres_exporter" file. This was causing issues with several upgrade scenarios. New installation instructions now have the service pointing directly to the relevant sysconfig file for the major PostgreSQL version.
- **IMPORTANT**: If you are using the default postgres_exporter service, you will need to update your service name so it uses the proper sysconfig file. See the README file for the new default service name in the "Enable Services" section and run the "enable" command found there. You should then also disable/remove the old service so it doesn't try to start again in the future.

- The additional Crunchy provided configurations for node_exporter have been split out from the pgmonitor-pg##-extras package to the pgmonitor-node_exporter-extras package. This was done to allow multiple versions of the pg##-extras package to be installed with different major versions of Postgres. There is still currently a dependency that the node extras packages must be installed with the pg##-extras so that upgrading doesn't break existing systems. This dependency will be revisited in the future.
- Removed the requirement for a shell script to monitor if the database is up and its status as either a primary or replica. Up status is now using the native "pg_up" metric from postgres_exporter and a new metric query was written for checking the recovery status of a system (ccp_is_in_recovery).
- The PostgreSQL.json overview dashboard that used this metric has been redesigned. Unfortunately it can no longer be colored RED for down systems, only go colorless and say "DOWN". This is a known limitation of handling null metric values in Grafana and part of a larger fix coming in future versions - https://github.com/grafana/grafana/issues/11418
- Upgrade required version of Grafana to minimum of 5.2.1.
- All provided dashboards require this minimum version to work.
- If you notice that links between the dashboards are broken after the upgrade, clear your browser's cache. The 301 redirects used between dashboards can get cached and they have changed in the new major version.
- See extensive release notes for major version changes in Grafana - https://community.grafana.com/t/release-notes-v-5-1-x
- Change Grafana datasource and dashboard installation to use provisioning vs manual setup via the web interface. Note this means that future updates to the provided datasources and dashboards must be done through config files as well. Or they can be saved as a new dashboard for more extensive customization.
- Change recommended configuration for Grafana to use PostgreSQL as database backend. Updated installation documentation.
- Added Prometheus Alerts Dashboard. Shows both active alerts and 1 week history in table format.
- Removed Gauges from PostgreSQLDetails Dashboard. "Current" value was not being shown properly and gauges were misleading in their values depending on the time range chosen. For a quick glance to see if there are any problems, be sure to set your alert thresholds properly and use the new Prometheus Alerts Dashboard.
- Added max_query_time metric to track long running queries in general. Also added an alert for that metric to crunchy prometheus alerts.
- Added "IO Time Per Device in Seconds" graph to Filesystems dashboard.
- Fixed Memory and Swap Graphs on PostgreSQLDetails dashboard to more accurately show used resources. History for these graphs before this upgrade is not being shown since it is no longer graphing the same data.
- Crontabs are no longer PostgreSQL major version dependent at this time. Consolidated down to a single crontab file for all versions.
- Removed unnecessary functions from functions_pg10.sql. All queries in queries_pg10.yml currently only require the pg_monitor system role to be granted and have been updated with this assumption.
- Changed default cron runtime of pg_bloat_check to once a week on early morning weekend.
- Change PostgreSQL overview dashboard to use background colors instead of gauges for better visibility.
- Fixed permission issues with /etc/postgres_exporter folder to allow ccp_monitoring system user better control.

1.7

- Fixed duplicate and incorrect replication byte lag queries. The one contained in queries_common.yml should not have been there. It should be in queries_pg92-96.yml, but there was also one already there. However, the one already in pg92-96 was incorrect since prior to PG10, it requires superuser/security definer to fully access replication statistics. Corrected the version specific file to have the correct query. Made the query in the pg10 file consistent. Ensure you update your generated queries.yml file with he new queries.
- Fixed the PostgreSQLDetails.json dashboard to use the correct replication byte lag metric (referencing above fix). The easiest way to fix this is to delete this dashboard and re-import it. Otherwise, if you've made customizations you don't want to lose, you can grab the correct metric query from the updated dashboard gauge and edit your existing dashboard to use it.
- The combination of the above two fixes corrects the pgmonitor setup being able to properly handle there being multiple replicas from a single primary. Previously this would cause postgres_exporter to throw duplicate metric errors.
- Fixed the query in queries_bloat.yml to be able to properly handle if there was a bloat amount larger than max int4 bytes. Ensure you update your generated queries.yml file with the new query.

1.6

- Fixed formatting bug in crunchy-prometheus.yml. Thanks to Doug Hunley for reporting the issue.

1.5

- Add support for disabling built in queries in postgres_exporter 0.4.5. Also explicitly ignore these metrics via a prometheus filter so they're not ingested even if new option isn't used. This means that v1.5 of pgmonitor now requires 0.4.5 of postgres_exporter by default.
- Improved exporter down alert to avoid unnecessary alerts for brief outages that resolve themselves quickly.
- Added new FilesystemDetails dashboard for grafana that is linked to from the Filesystem graph on PostgreSQLDetails.

- Top level PostgreSQL grafana dashboard now identifies whether a system is read/write or readonly to better distinguish primary/replica systems.
- Added instructions for non-packaged installation using pgmonitor configuration files.
- Revised and better formatted README documentation

1.4

- Fixed filesystem graphs in PostgreSQLDetails dashboard
- Cosmetic changes to PostgreSQLDetails dashboard
- Added instructions for importing dashboards via Grafana API

1.3

- Fixed error in PG10 queries file.
- Fixed disk usage alert for prometheus to work better when there are many jobs with similar mountpoints. Also fixed syntax error in warning alert.
- Moved connection stats query from common to version specific queries due to PG10 differences. Clarified naming of files for which versions they work for.
- Added dropdown for the Job to the lower level drill down dashboards in Grafana. Allows selecting of a specific system from the dashboard itself without having to click through on a higher level.
- Removed pg_stat_statements graph from PostgreSQLDetails dashboard. Needs refinement to make it more useful.

1.2

- Change service and sysconfig files to use single OPT environment variable instead of one variable per cmd option
- Fix error in PG10 monitoring functions file
- Initial version of Prometheus 2.0 job deletion script. Requires API call not available yet in 2.0.0 for full functionality

1.1

- Implement rpmnew/rpmsave feature instead of using .example files to prevent package overwriting user changes to configs

1.0

- Initial stable release