Installation

Contents

---

itle: “Crunchy Data Postgres Operator”

ate:

raft: false

Latest Release: 4.0.0

The *postgres-operator* is a controller that runs within a Kubernetes cluster that provides a means to deploy and manage PostgreSQL clusters.

Use the postgres-operator to:

- deploy PostgreSQL containers including streaming replication clusters
- scale up PostgreSQL clusters with extra replicas
- add pgpool, pgbouncer, and metrics sidecars to PostgreSQL clusters
- apply SQL policies to PostgreSQL clusters
- assign metadata tags to PostgreSQL clusters
- maintain PostgreSQL users and passwords
- perform minor upgrades to PostgreSQL clusters
- load simple CSV and JSON files into PostgreSQL clusters
- perform database backups

Deployment Requirements

The Operator deploys on Kubernetes and OpenShift clusters. Some form of storage is required, NFS, HostPath, and Storage Classes are currently supported.

The Operator includes various components that get deployed to your Kubernetes cluster as shown in the following diagram and detailed in the Design.

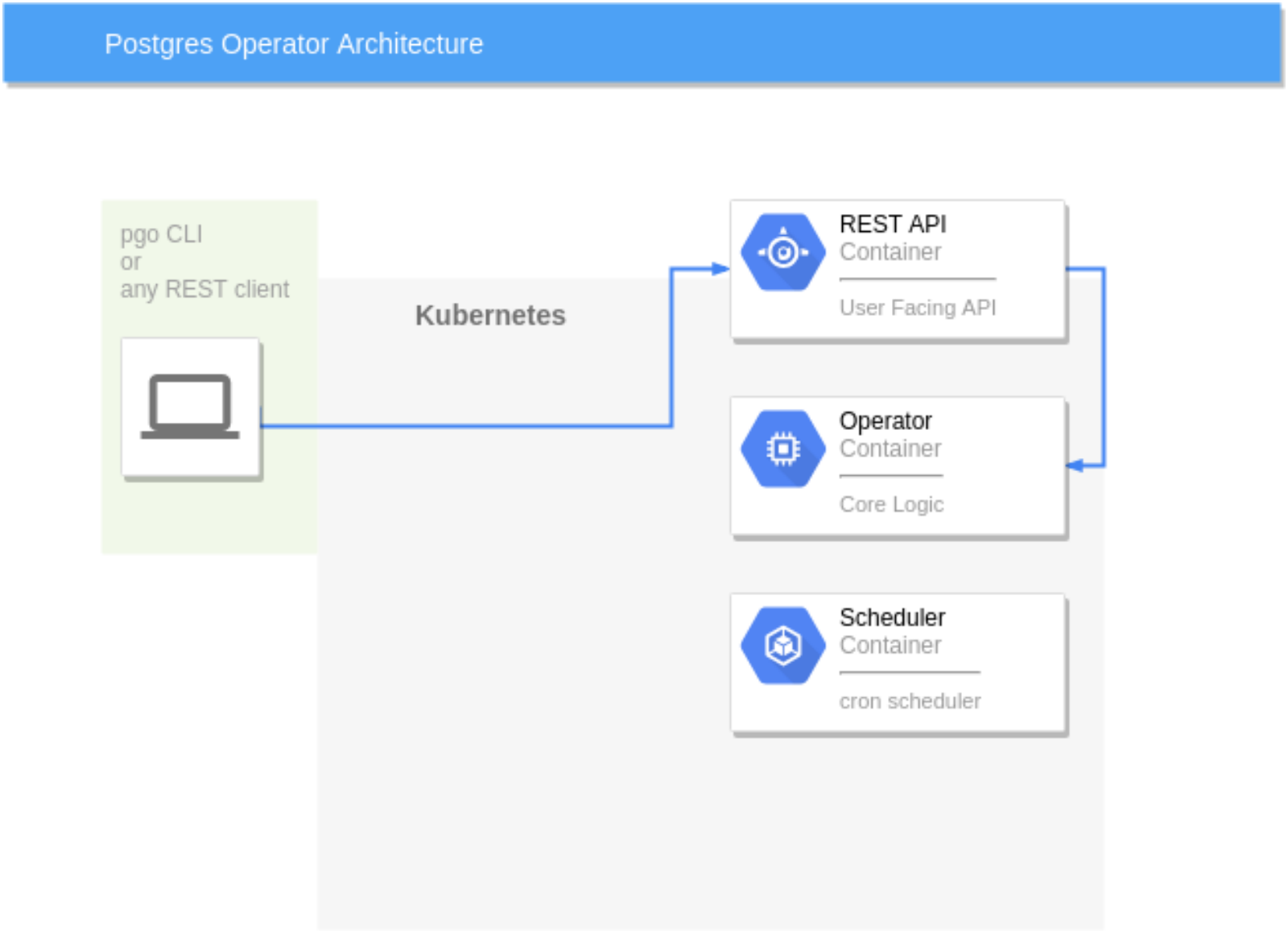

Figure 1: Architecture

The Operator is developed and tested on CentOS and RHEL Linux platforms but is known to run on other Linux variants.

Documentation

The following documentation is provided:

The operator is template-driven; this makes it simple to configure both the client and the operator.

conf Directory

The Operator is configured with a collection of files found in the *conf* directory. These configuration files are deployed to your Kubernetes cluster when the Operator is deployed. Changes made to any of these configuration files currently require a redeployment of the Operator on the Kubernetes cluster.

The server components of the Operator include Role Based Access Control resources which need to be created a single time by a Kubernetes cluster-admin user. See the Installation section for details on installing a Postgres Operator server.

The configuration files used by the Operator are found in 2 places: * the pgo-config ConfigMap in the namespace the Operator is running in * or, a copy of the configuration files are also included by default into the Operator container images themselves to support a very simplistic deployment of the Operator

If the pgo-config ConfigMap is not found by the Operator, it will use the configuration files that are included in the Operator container images.

The container included set of configuration files use the most basic setting values and the image versions of the Operator itself with the latest Crunchy Container image versions. The storage configurations are determined by using the default storage class on the system you are deploying the Operator into, the default storage class is one that is labeled as follows:

```
pgo-default-sc=true
```

If no storage class has that label, then the first storage class found on the system will be used. If no storage class is found on the system, the containers will not run and produce an error in the log.

conf/postgres-operator/pgo.yaml

The *pgo.yaml* file sets many different Operator configuration settings and is described in the [pgo.yaml configuration]({{< ref "pgo-yaml-configuration.md" >}}) documentation section.

The *pgo.yaml* file is deployed along with the other Operator configuration files when you run:

```
make deployoperator
```

conf/postgres-operator Directory

Files within the *conf/postgres-operator* directory contain various templates that are used by the Operator when creating Kubernetes resources. In an advanced Operator deployment, administrators can modify these templates to add their own custom meta-data or make other changes to influence the Resources that get created on your Kubernetes cluster by the Operator.

Files within this directory are used specifically when creating PostgreSQL Cluster resources. Sidecar components such as pgBouncer and pgPool II templates are also located within this directory.

As with the other Operator templates, administrators can make custom changes to this set of templates to add custom features or metadata into the Resources created by the Operator.

Security

Setting up pgo users and general security configuration is described in the Security section of this documentation.

Local pgo CLI Configuration

You can specify the default namespace you want to use by setting the PGO_NAMESPACE environment variable locally on the host the pgo CLI command is running.

```
export PGO_NAMESPACE=pgouser1
```

When that variable is set, each command you issue with *pgo* will use that namespace unless you over-ride it using the *–namespace* command line flag.

```
pgo show cluster foo --namespace=pgouser2
```

Namespace Configuration

The Design Design section of this documentation talks further about the use of namespaces within the Operator and configuring different deployment models of the Operator.

pgo.yaml Configuration

The *pgo.yaml* file contains many different configuration settings as described in this section of the documentation.

The *pgo.yaml* file is broken into major sections as described below: ## Cluster

Setting	Definition
BasicAuth	if set to *true* will enable Basic Authentication
PrimaryNodeLabel	newly created primary deployments will specify this node label if specified, unless you override it using the –
ReplicaNodeLabel	newly created replica deployments will specify this node label if specified, unless you override it using the –n
CCPImagePrefix	newly created containers will be based on this image prefix (e.g. crunchydata), update this if you require a cu
CCPImageTag	newly created containers will be based on this image version (e.g. centos7-11.2-2.4.0-rc5), unless you override
Port	the PostgreSQL port to use for new containers (e.g. 5432)
LogStatement	postgresql.conf log_statement value (required field)
LogMinDurationStatement	postgresql.conf log_min_duration_statement value (required field)
User	the PostgreSQL normal user name
Database	the PostgreSQL normal user database
Replicas	the number of cluster replicas to create for newly created clusters, typically users will scale up replicas on the
PgmonitorPassword	the password to use for pgmonitor metrics collection if you specify –metrics when creating a PG cluster
Metrics	boolean, if set to true will cause each new cluster to include crunchy-collect as a sidecar container for metrics
Badger	boolean, if set to true will cause each new cluster to include crunchy-pgbadger as a sidecar container for stati
Policies	optional, list of policies to apply to a newly created cluster, comma separated, must be valid policies in the c
PasswordAgeDays	optional, if set, will set the VALID UNTIL date on passwords to this many days in the future when creating
PasswordLength	optional, if set, will determine the password length used when creating passwords, defaults to 8
ServiceType	optional, if set, will determine the service type used when creating primary or replica services, defaults to Cl
Backrest	optional, if set, will cause clusters to have the pgbackrest volume PVC provisioned during cluster creation
BackrestPort	currently required to be port 2022
Autofail	optional, if set, will cause clusters to be checked for auto failover in the event of a non-Ready status
AutofailReplaceReplica	optional, default is false, if set, will determine whether a replica is created as part of a failover to replace the

Storage

Setting	Definition
PrimaryStorage	required, the value of the storage configuration to use for the primary PostgreSQL deployment

Setting	Definition
BackupStorage	required, the value of the storage configuration to use for backups, including the storage for pgbackrest rep
ReplicaStorage	required, the value of the storage configuration to use for the replica PostgreSQL deployments
ReplicaStorage	required, the value of the storage configuration to use for the replica PostgreSQL deployments
BackrestStorage	required, the value of the storage configuration to use for the pgbackrest shared repository deployment cre
StorageClass	for a dynamic storage type, you can specify the storage class used for storage provisioning(e.g. standard, g
AccessMode	the access mode for new PVCs (e.g. ReadWriteMany, ReadWriteOnce, ReadOnlyMany). See below for des
Size	the size to use when creating new PVCs (e.g. 100M, 1Gi)
Storage.storage1.StorageType	supported values are either *dynamic*, *create*, if not supplied, *create* is used
Fsgroup	optional, if set, will cause a *SecurityContext* and *fsGroup* attributes to be added to generated Pod and Dep
SupplementalGroups	optional, if set, will cause a SecurityContext to be added to generated Pod and Deployment definitions
MatchLabels	optional, if set, will cause the PVC to add a *matchlabels* selector in order to match a PV, only useful when

Storage Configuration Examples

In *pgo.yaml*, you will need to configure your storage configurations depending on which storage you are wanting to use for Operator provisioning of Persistent Volume Claims. The examples below are provided as a sample. In all the examples you are free to change the *Size* to meet your requirements of Persistent Volume Claim size.

HostPath Example

HostPath is provided for simple testing and use cases where you only intend to run on a single Linux host for your Kubernetes cluster.

```
hostpathstorage:
  AccessMode:  ReadWriteMany
  Size:  1G
  StorageType:  create
```

NFS Example

In the following NFS example, notice that the *SupplementalGroups* setting is set, this can be whatever GID you have your NFS mount set to, typically we set this *nfsnobody* as below. NFS file systems offer a *ReadWriteMany* access mode.

```
nfsstorage:
  AccessMode:  ReadWriteMany
  Size:  1G
  StorageType:  create
  SupplementalGroups:  65534
```

Storage Class Example

In the following example, the important attribute to set for a typical Storage Class is the *Fsgroup* setting. This value is almost always set to *26* which represents the Postgres user ID that the Crunchy Postgres container runs as. Most Storage Class providers offer *ReadWriteOnce* access modes, but refer to your provider documentation for other access modes it might support.

```
storageos:
  AccessMode:  ReadWriteOnce
  Size:  1G
  StorageType:  dynamic
  StorageClass:  fast
  Fsgroup:  26
```

Container Resources

Setting	Definition
DefaultContainerResource	optional, the value of the container resources configuration to use for all database containers, if not set, no re
DefaultLoadResource	optional, the value of the container resources configuration to use for pgo-load containers, if not set, no resou
DefaultLspvcResource	optional, the value of the container resources configuration to use for pgo-lspvc containers, if not set, no reso
DefaultRmdataResource	optional, the value of the container resources configuration to use for pgo-rmdata containers, if not set, no re
DefaultBackupResource	optional, the value of the container resources configuration to use for crunchy-backup containers, if not set, n
DefaultPgbouncerResource	optional, the value of the container resources configuration to use for crunchy-pgbouncer containers, if not se
DefaultPgpoolResource	optional, the value of the container resources configuration to use for crunchy-pgpool containers, if not set, n
RequestsMemory	request size of memory in bytes
RequestsCPU	request size of CPU cores
LimitsMemory	request size of memory in bytes
LimitsCPU	request size of CPU cores

Miscellaneous (Pgo)

Setting	Definition
PreferredFailoverNode	optional, a label selector (e.g. hosttype=offsite) that if set, will be used to pick the failover target which is runnin
COImagePrefix	image tag prefix to use for the Operator containers
COImageTag	image tag to use for the Operator containers
Audit	boolean, if set to true will cause each apiserver call to be logged with an *audit* marking

Storage Configuration Details

You can define n-number of Storage configurations within the *pgo.yaml* file. Those Storage configurations follow these conventions -

- they must have lowercase name (e.g. storage1)
- they must be unique names (e.g. mydrstorage, faststorage, slowstorage)

These Storage configurations are referenced in the BackupStorage, ReplicaStorage, and PrimaryStorage configuration values. However, there are command line options in the *pgo* client that will let a user override these default global values to offer you the user a way to specify very targeted storage configurations when needed (e.g. disaster recovery storage for certain backups).

You can set the storage AccessMode values to the following:

- *ReadWriteMany* - mounts the volume as read-write by many nodes
- *ReadWriteOnce* - mounts the PVC as read-write by a single node
- *ReadOnlyMany* - mounts the PVC as read-only by many nodes

These Storage configurations are validated when the *pgo-apiserver* starts, if a non-valid configuration is found, the apiserver will abort. These Storage values are only read at *apiserver* start time.

The following StorageType values are possible -

- *dynamic* - this will allow for dynamic provisioning of storage using a StorageClass.
- *create* - This setting allows for the creation of a new PVC for each PostgreSQL cluster using a naming convention of *clustername*. When set, the *Size*, *AccessMode* settings are used in constructing the new PVC.

The operator will create new PVCs using this naming convention: *dbname* where *dbname* is the database name you have specified. For example, if you run:

```
pgo create cluster example1 -n pgouser1
```

It will result in a PVC being created named *example1* and in the case of a backup job, the pvc is named *example1-backup*

Note, when Storage Type is *create*, you can specify a storage configuration setting of *MatchLabels*, when set, this will cause a *selector* of *key=value* to be added into the PVC, this will let you target specific PV(s) to be matched for this cluster. Note, if a PV does not match the claim request, then the cluster will not start. Users that want to use this feature have to place labels on their PV resources as part of PG cluster creation before creating the PG cluster. For example, users would add a label like this to their PV before they create the PG cluster:

```
kubectl label pv somepv myzone=somezone -n pgouser1
```

If you do not specify *MatchLabels* in the storage configuration, then no match filter is added and any available PV will be used to satisfy the PVC request. This option does not apply to *dynamic* storage types.

Example PV creation scripts are provided that add labels to a set of PVs and can be used for testing: `$COROOT/pv/create-pv-nfs-labels.sh` in that example, a label of **crunchyzone=red** is set on a set of PVs to test with.

The *pgo.yaml* includes a storage config named **nfsstoragered** that when used will demonstrate the label matching. This feature allows you to support n-number of NFS storage configurations and supports spreading a PG cluster across different NFS storage configurations.

Container Resources Details

In the *pgo.yaml* configuration file you have the option to configure a default container resources configuration that when set will add CPU and memory resource limits and requests values into each database container when the container is created.

You can also override the default value using the `--resources-config` command flag when creating a new cluster:

```
pgo create cluster testcluster --resources-config=large -n pgouser1
```

Note, if you try to allocate more resources than your host or Kube cluster has available then you will see your pods wait in a *Pending* status. The output from a `kubectl describe pod` command will show output like this in this event: Events:

```
  Type     Reason            Age               From               Message
  ----     ------            ----              ----               -------
  Warning  FailedScheduling  49s (x8 over 1m)  default-scheduler  No nodes are available that
     match all of the predicates: Insufficient memory (1).
```

Overriding Storage Configuration Defaults

```
pgo create cluster testcluster --storage-config=bigdisk -n pgouser1
```

That example will create a cluster and specify a storage configuration of *bigdisk* to be used for the primary database storage. The replica storage will default to the value of ReplicaStorage as specified in *pgo.yaml*.

```
pgo create cluster testcluster2 --storage-config=fastdisk --replica-storage-config=slowdisk -n
   pgouser1
```

That example will create a cluster and specify a storage configuration of *fastdisk* to be used for the primary database storage, while the replica storage will use the storage configuration *slowdisk*.

```
pgo backup testcluster --storage-config=offsitestorage -n pgouser1
```

That example will create a backup and use the *offsitestorage* storage configuration for persisting the backup.

Using Storage Configurations for Disaster Recovery

A simple mechanism for partial disaster recovery can be obtained by leveraging network storage, Kubernetes storage classes, and the storage configuration options within the Operator.

For example, if you define a Kubernetes storage class that refers to a storage backend that is running within your disaster recovery site, and then use that storage class as a storage configuration for your backups, you essentially have moved your backup files automatically to your disaster recovery site thanks to network storage.

Kubernetes RBAC

Install the requisite Operator RBAC resources, *as a Kubernetes cluster admin user*, by running a Makefile target:

```
make installrbac
```

This script creates the following RBAC resources on your Kubernetes cluster:

Setting	Definition
Custom Resource Definitions (crd.yaml)	pgbackups
	pgclusters
	pgpolicies
	pgreplicas
	pgtasks
	pgupgrades
Cluster Roles (cluster-roles.yaml)	pgopclusterrole
	pgopclusterrolecrd
Cluster Role Bindings (cluster-roles-bindings.yaml)	pgopclusterbinding
	pgopclusterbindingcrd
Service Account (service-accounts.yaml)	postgres-operator
	pgo-backrest
Roles (rbac.yaml)	pgo-role
	pgo-backrest-role
Role Bindings (rbac.yaml)	pgo-backrest-role-binding
	pgo-role-binding

Note that the cluster role bindings have a naming convention of pgopclusterbinding-$PGO_OPERATOR_NAMESPACE$andpgopclusterbindin
The PGO_OPERATOR_NAMESPACE environment variable is added to make each cluster role binding name unique and to support more than a single Operator being deployed on the same Kube cluster.

Operator RBAC

The *conf/postgresql-operator/pgorole* file is read at start up time when the operator is deployed to the Kubernetes cluster. This file defines the Operator roles whereby Operator API users can be authorized.

The *conf/postgresql-operator/pgouser* file is read at start up time also and contains username, password, role, and namespace information as follows:

```
username:password:pgoadmin:
pgouser1:password:pgoadmin:pgouser1
pgouser2:password:pgoadmin:pgouser2
pgouser3:password:pgoadmin:pgouser1,pgouser2
readonlyuser:password:pgoreader:
```

The format of the pgouser server file is:

```
<username>:<password>:<role>:<namespace,namespace>
```

The namespace is a comma separated list of namespaces that user has access to. If you do not specify a namespace, then all namespaces is assumed, meaning this user can access any namespace that the Operator is watching.

A user creates a *.pgouser* file in their $HOME directory to identify themselves to the Operator. An entry in .pgouser will need to match entries in the *conf/postgresql-operator/pgouser* file. A sample *.pgouser* file contains the following:

```
username:password
```

The format of the .pgouser client file is:

```
<username>:<password>
```

The users pgouser file can also be located at: */etc/pgo/pgouser* or it can be found at a path specified by the PGOUSER environment variable.

If the user tries to access a namespace that they are not configured for within the server side *pgouser* file then they will get an error message as follows:

```
Error: user [pgouser1] is not allowed access to namespace [pgouser2]
```

The following list shows the current complete list of possible pgo permissions that you can specify within the *pgorole* file when creating roles:

Permission	Description
ApplyPolicy	allow *pgo apply*
Cat	allow *pgo cat*
CreateBackup	allow *pgo backup*
CreateBenchmark	allow *pgo create benchmark*
CreateCluster	allow *pgo create cluster*
CreateDump	allow *pgo create pgdump*
CreateFailover	allow *pgo failover*
CreatePgbouncer	allow *pgo create pgbouncer*
CreatePgpool	allow *pgo create pgpool*
CreatePolicy	allow *pgo create policy*
CreateSchedule	allow *pgo create schedule*
CreateUpgrade	allow *pgo upgrade*
CreateUser	allow *pgo create user*
DeleteBackup	allow *pgo delete backup*
DeleteBenchmark	allow *pgo delete benchmark*
DeleteCluster	allow *pgo delete cluster*
DeletePgbouncer	allow *pgo delete pgbouncer*
DeletePgpool	allow *pgo delete pgpool*
DeletePolicy	allow *pgo delete policy*
DeleteSchedule	allow *pgo delete schedule*
DeleteUpgrade	allow *pgo delete upgrade*
DeleteUser	allow *pgo delete user*
DfCluster	allow *pgo df*
Label	allow *pgo label*
Load	allow *pgo load*
Ls	allow *pgo ls*
Reload	allow *pgo reload*
Restore	allow *pgo restore*
RestoreDump	allow *pgo restore* for pgdumps
ShowBackup	allow *pgo show backup*
ShowBenchmark	allow *pgo show benchmark*
ShowCluster	allow *pgo show cluster*
ShowConfig	allow *pgo show config*
ShowPolicy	allow *pgo show policy*
ShowPVC	allow *pgo show pvc*
ShowSchedule	allow *pgo show schedule*
ShowNamespace	allow *pgo show namespace*
ShowUpgrade	allow *pgo show upgrade*
ShowWorkflow	allow *pgo show workflow*
Status	allow *pgo status*
TestCluster	allow *pgo test*
UpdateCluster	allow *pgo update cluster*
User	allow *pgo user*
Version	allow *pgo version*

If the user is unauthorized for a pgo command, the user will get back this response:

```
Error:  Authentication Failed: 401
```

Making Security Changes

The Operator today requires you to make Operator user security changes in the pgouser and pgorole files, and for those changes to take effect you are required to re-deploy the Operator:

```
make deployoperator
```

This will recreate the *pgo-config* ConfigMap that stores these files and is mounted by the Operator during its initialization.

API Security

The Operator REST API is encrypted with keys stored in the *pgo.tls* Secret.

The pgo.tls Secret can be generated prior to starting the Operator or you can let the Operator generate the Secret for you if the Secret does not exist.

Adjust the default keys to meet your security requirements using your own keys. The *pgo.tls* Secret is created when you run:

```
make deployoperator
```

The keys are generated when the RBAC script is executed by the cluster admin:

```
make installrbac
```

In some scenarios like an OLM deployment, it is preferable for the Operator to generate the Secret keys at runtime, if the pgo.tls Secret does not exit when the Operator starts, a new TLS Secret will be generated. In this scenario, you can extract the generated Secret TLS keys using:

u kubectl cp pgo/postgres-operator-585584f57d-ntwr5:/tmp/server.key /tmp/server.key -c apiserver u kubectl cp pgo/postgres-operator-585584f57d-ntwr5:/tmp/server.crt /tmp/server.crt -c apiserver

This server.key and server.crt can then be used to access the *pgo-apiserver* from the pgo CLI by setting the following variables in your client environment:

```
export PGO_CA_CERT=/tmp/server.crt
export PGO_CLIENT_CERT=/tmp/server.crt
export PGO_CLIENT_KEY=/tmp/server.key
```

You can view the TLS secret using:

```
kubectl get secret pgo.tls -n pgo
```

or oc get secret pgo.tls -n pgo

If you create the Secret outside of the Operator, for example using the default installation script, the key and cert that are generated by the default installation are found here:

```
$PGOROOT/conf/postgres-operator/server.crt
$PGOROOT/conf/postgres-operator/server.key
```

The key and cert are generated using the *deploy/gen-api-keys.sh* script. That script gets executed when running:

```
make installrbac
```

You can extract the server.key and server.crt from the Secret using the following:

```
oc get secret pgo.tls -n $PGO_OPERATOR_NAMESPACE -o jsonpath='{.data.tls\.key}' | base64 --decode
   > /tmp/server.key
oc get secret pgo.tls -n $PGO_OPERATOR_NAMESPACE -o jsonpath='{.data.tls\.crt}' | base64 --decode
   > /tmp/server.crt
```

This server.key and server.crt can then be used to access the *pgo-apiserver* REST API from the pgo CLI on your client host.

Upgrading the Operator

Various Operator releases will require action by the Operator administrator of your organization in order to upgrade to the next release of the Operator. Some upgrade steps are automated within the Operator but not all are possible at this time.

This section of the documentation shows specific steps required to the latest version from the previous version.

Upgrading to Version 3.5.0 From Previous Versions

- For clusters created in prior versions that used pgbackrest, you will be required to first create a pgbasebackup for those clusters, and after upgrading to Operator 3.5, you will need to restore those clusters from the pgbasebackup into a new cluster with *–pgbackrest* enabled, this is due to the new pgbackrest shared repository being implemented in 3.5. This is a breaking change for anyone that used pgbackrest in Operator versions prior to 3.5.
- The pgingest CRD is removed. You will need to manually remove it from any deployments of the operator after upgrading to this version. This includes removing ingest related permissions from the pgorole file. Additionally, the API server now removes the ingest related API endpoints.
- Primary and replica labels are only applicable at cluster creation and are not updated after a cluster has executed a failover. A new *service-name* label is applied to PG cluster components to indicate whether a deployment/pod is a primary or replica. *service-name* is also the label now used by the cluster services to route with. This scheme allows for an almost immediate failover promotion and avoids the pod having to be bounced as part of a failover. Any existing PostgreSQL clusters will need to be updated to specify them as a primary or replica using the new *service-name* labeling scheme.

- The autofail label was moved from deployments and pods to just the pgcluster CRD to support autofail toggling.
- The storage configurations in *pgo.yaml* support the MatchLabels attribute for NFS storage. This will allow users to have more than a single NFS backend,. When set, this label (key=value) will be used to match the labels on PVs when a PVC is created.
- The UpdateCluster permission was added to the sample pgorole file to support the new pgo update CLI command. It was also added to the pgoperm file.
- The pgo.yaml adds the PreferredFailoverNode setting. This is a Kubernetes selector string (e.g. key=value). This value if set, will cause fail-over targets to be preferred based on the node they run on if that node is in the set of *preferred*.
- The ability to select nodes based on a selector string was added. For this to feature to be used, multiple replicas have to be in a ready state, and also at the same replication status. If those conditions are not met, the default fail-over target selection is used.
- The pgo.yaml file now includes a new storage configuration, XlogStorage, which when set will cause the xlog volume to be allocated using this storage configuration. If not set, the PrimaryStorage configuration will be used.
- The pgo.yaml file now includes a new storage configuration, BackrestStorage, will cause the pgbackrest shared repository volume to be allocated using this storage configuration.
- The pgo.yaml file now includes a setting, AutofailReplaceReplica, which will enable or disable whether a new replica is created as part of a fail-over. This is turned off by default.

See the GitHub Release notes for the features and other notes about a specific release.