



EDB Postgres for Kubernetes

Release 1.15.0

EDB

May 19, 2022

CONTENTS

1	EDB Postgres for Kubernetes Release notes	3
1.1	EDB Postgres for Kubernetes 1.15.0 release notes	4
1.2	EDB Postgres for Kubernetes 1.14.0 release notes	4
1.3	EDB Postgres for Kubernetes 1.13.0 release notes	5
1.4	EDB Postgres for Kubernetes 1.12.0 release notes	6
1.5	EDB Postgres for Kubernetes 1.11.0 release notes	7
1.6	EDB Postgres for Kubernetes 1.10.0 release notes	8
1.7	EDB Postgres for Kubernetes 1.9.2 release notes	9
1.8	EDB Postgres for Kubernetes 1.9.1 release notes	10
1.9	EDB Postgres for Kubernetes 1.9.0 release notes	10
1.10	EDB Postgres for Kubernetes 1.8.0 release notes	11
1.11	EDB Postgres for Kubernetes 1.7.1 release notes	11
1.12	EDB Postgres for Kubernetes 1.7.0 release notes	12
1.13	EDB Postgres for Kubernetes 1.6.0 release notes	13
1.14	EDB Postgres for Kubernetes 1.5.1 release notes	13
1.15	EDB Postgres for Kubernetes 1.5.0 release notes	13
1.16	EDB Postgres for Kubernetes 1.4.0 release notes	14
1.17	EDB Postgres for Kubernetes 1.3.0 release notes	15
1.18	EDB Postgres for Kubernetes 1.2.1 release notes	15
1.19	EDB Postgres for Kubernetes 1.2.0 release notes	15
1.20	EDB Postgres for Kubernetes 1.1 release notes	16
1.21	EDB Postgres for Kubernetes 1.0 release notes	16
2	‘Differences with CloudNativePG v1’	17
3	‘Differences with CloudNativePG’	19
4	‘Installation and upgrades’	21
4.1	Installation on Kubernetes	21
4.1.1	Directly using the operator manifest	21
4.2	Details about the deployment	21
5	‘Quickstart’	23
5.1	Part 1 - Setup the local Kubernetes/Openshift playground	23
5.1.1	CodeReady Containers (CRC)	24
5.2	Part 2 - Install EDB Postgres for Kubernetes	24
5.3	Part 3 - Deploy a PostgreSQL cluster	24
6	Installation, Configuration and Deployment Demo	25
6.1	Install EDB Postgres for Kubernetes	26
6.2	Deploy a PostgreSQL cluster	26

6.3	Install the kubectl-cnp plugin	30
6.4	Testing failover	31
6.4.1	Further reading	33
7	‘Operator configuration’	35
7.1	Available options	35
7.2	Defining an operator config map	36
7.3	Defining an operator secret	36
7.4	Restarting the operator to reload configs	37
7.5	PPROF HTTP SERVER	37
8	EDB Audit logs	39
9	‘EDB Postgres for Kubernetes Plugin’	41
9.1	Features	41
9.2	Install	41
9.2.1	Supported Architectures	41
9.3	Use	41
9.3.1	Status	42
9.3.2	Promote	45
9.3.3	Certificates	45
9.3.4	Restart	45
9.3.5	Reload	46
9.3.6	Maintenance	46
9.3.7	Report	46
9.3.8	report Operator	47
9.3.9	report Cluster	48
10	‘Red Hat OpenShift’	51
10.1	Important OpenShift concepts	52
10.1.1	Cluster Service Version (CSV)	53
10.1.2	Limitations for multi-tenant management	53
10.2	Installation via web console	53
10.2.1	Cluster-wide installation	56
10.2.2	Single project installation	57
10.3	Installation via the oc CLI	59
10.3.1	Cluster-wide installation with oc	61
10.3.2	Installing the OpenShift CLI (oc)	61
10.4	Predefined RBAC objects	61
10.4.1	Custom Resource Definitions (CRD)	63
10.4.2	Service accounts	63
10.4.3	Cluster roles	64
10.4.4	Cluster roles on EDB Postgres for Kubernetes CRDs	64
10.4.5	Cluster roles on Kubernetes CRDs	65
10.5	Users and Permissions	67
10.6	Customization of the Pooler image	69
11	‘Free evaluation’	71
11.1	Evaluating PostgreSQL	71
11.2	Evaluating EDB Postgres Advanced Server	71
12	‘License and license keys’	73
12.1	Company level license keys	73
12.1.1	Kubernetes installations via YAML manifest	73
12.1.2	OpenShift and Kubernetes installations with OLM	74

12.2	Cluster level license keys	74
12.3	License key secret at cluster level	75
12.4	What happens when a license expires	75
13	'API Reference'	77
13.1	AffinityConfiguration	79
13.2	AzureCredentials	80
13.3	Backup	80
13.4	BackupConfiguration	80
13.5	BackupList	81
13.6	BackupSource	81
13.7	BackupSpec	81
13.8	BackupStatus	81
13.9	BarmanObjectStoreConfiguration	82
13.10	BootstrapConfiguration	83
13.11	BootstrapInitDB	83
13.12	BootstrapPgBaseBackup	84
13.13	BootstrapRecovery	84
13.14	CertificatesConfiguration	85
13.15	CertificatesStatus	86
13.16	Cluster	86
13.17	ClusterCondition	86
13.18	ClusterList	87
13.19	ClusterSpec	87
13.20	ClusterStatus	89
13.21	ConfigMapKeySelector	90
13.22	ConfigMapResourceVersion	90
13.23	DataBackupConfiguration	90
13.24	EPASConfiguration	91
13.25	EmbeddedObjectMetadata	91
13.26	ExternalCluster	91
13.27	GoogleCredentials	92
13.28	InstanceID	92
13.29	LDAPBindAsAuth	92
13.30	LDAPBindSearchAuth	92
13.31	LDAPConfig	93
13.32	LocalObjectReference	93
13.33	MonitoringConfiguration	93
13.34	NodeMaintenanceWindow	94
13.35	PgBouncerIntegrationStatus	94
13.36	PgBouncerSecrets	94
13.37	PgBouncerSpec	94
13.38	PodMeta	95
13.39	PodTemplateSpec	95
13.40	Pooler	96
13.41	PoolerIntegrations	96
13.42	PoolerList	96
13.43	PoolerSecrets	96
13.44	PoolerSpec	97
13.45	PoolerStatus	97
13.46	PostgresConfiguration	97
13.47	RecoveryTarget	98
13.48	ReplicaClusterConfiguration	98
13.49	RollingUpdateStatus	98

13.50 S3Credentials	99
13.51 ScheduledBackup	99
13.52 ScheduledBackupList	99
13.53 ScheduledBackupSpec	100
13.54 ScheduledBackupStatus	100
13.55 SecretKeySelector	100
13.56 SecretVersion	100
13.57 SecretsResourceVersion	101
13.58 StorageConfiguration	101
13.59 WalBackupConfiguration	102

EDB Postgres for Kubernetes is an operator is designed, developed, and supported by EDB that covers the full lifecycle of a highly available Postgres database clusters with a primary/standby architecture, using native streaming replication. It is based on the open source CloudNativePG operator, and provides additional value such as compatibility with Oracle using EDB Postgres Advanced Server and additional supported platforms such as IBM Power and OpenShift.

Key features available include:- Kubernetes API integration for high availability- Self-healing through failover and automated recreation of replicas- Capacity management with scale up/down capabilities- Planned switchovers for scheduled maintenance- Read-only and read-write Kubernetes services definitions- Rolling updates for Postgres minor versions and operator upgrades- Continuous backup and point-in-time recovery- Connection Pooling with PgBouncer- Integrated metrics exporter out of the box- PostgreSQL replication across multiple Kubernetes clusters- Red Hat certified operator for OpenShift

The operator has been renamed from Cloud Native PostgreSQL. Existing users of Cloud Native PostgreSQL will not experience any change, as the underlying components and resources have not changed.

EDB POSTGRES FOR KUBERNETES RELEASE NOTES

The EDB Postgres for Kubernetes documentation describes the major version of EDB Postgres for Kubernetes, including minor releases and patches. The release notes provide information on what is new in each release. For new functionality introduced in a minor or patch release, the content also indicates the release that introduced the feature.

Version	Release date	Upstream merges
<i>EDB Postgres for Kubernetes 1.15.0 release notes</i>	2022 Apr 21	Upstream <i>EDB Postgres for Kubernetes 1.15.0 release notes</i>
<i>EDB Postgres for Kubernetes 1.14.0 release notes</i>	2022 Mar 25	NA
<i>EDB Postgres for Kubernetes 1.13.0 release notes</i>	2022 Feb 17	NA
<i>EDB Postgres for Kubernetes 1.12.0 release notes</i>	2022 Jan 11	NA
<i>EDB Postgres for Kubernetes 1.11.0 release notes</i>	2021 Dec 15	NA
<i>EDB Postgres for Kubernetes 1.10.0 release notes</i>	2021 Nov 11	NA
<i>EDB Postgres for Kubernetes 1.9.2 release notes</i>	2021 Oct 15	NA
<i>EDB Postgres for Kubernetes 1.9.1 release notes</i>	2021 Sep 30	NA
<i>EDB Postgres for Kubernetes 1.9.0 release notes</i>	2021 Sep 28	NA
<i>EDB Postgres for Kubernetes 1.8.0 release notes</i>	2021 Sep 13	NA
<i>EDB Postgres for Kubernetes 1.7.1 release notes</i>	2021 Aug 11	NA

continues on next page

Table 1 – continued from previous page

Version	Release date	Upstream merges
EDB Postgres for Kubernetes 1.7.0 release notes	2021 Jul 28	NA
EDB Postgres for Kubernetes 1.6.0 release notes	2021 Jul 12	NA
EDB Postgres for Kubernetes 1.5.1 release notes	2021 Jun 11	NA
EDB Postgres for Kubernetes 1.5.0 release notes	2021 Jun 17	NA
EDB Postgres for Kubernetes 1.4.0 release notes	2021 May 18	NA
EDB Postgres for Kubernetes 1.3.0 release notes	2021 Apr 23	NA
EDB Postgres for Kubernetes 1.2.1 release notes	2021 Apr 6	NA
EDB Postgres for Kubernetes 1.2.0 release notes	2021 Mar 31	NA
1.1.0	2021 Mar 3	NA
1.0.0	2021 Feb 4	NA

1.1 EDB Postgres for Kubernetes 1.15.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Upstream merge	Merged with community CloudNativePG 1.15. See the community EDB Postgres for Kubernetes Release notes .

1.2 EDB Postgres for Kubernetes 1.14.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Natively support Google Cloud Storage for backup and recovery, by taking advantage of the features introduced in Barman Cloud 2.19.
Feature	Improved observability of backups through the introduction of the LastBackupSucceeded condition for the Cluster object.

continues on next page

Table 3 – continued from previous page

Type	Description
Feature	Support update of Hot Standby sensitive parameters: max_connections, max_prepared_transactions, max_locks_per_transaction, max_wal_senders, max_worker_processes.
Feature	Add the Online upgrade in progress phase in the Cluster object to show when an online upgrade of the operator is in progress.
Feature	Ability to inherit an AWS IAM Role as an alternative way to provide credentials for the S3 object storage.
Feature	Support for Opaque secrets for Pooler's authQuerySecret and certificates.
Feature	Updated default PostgreSQL version to 14.2.
Feature	Add a new command to kubectl cnp plugin named maintenance to set maintenance window to cluster(s) in one or all namespaces across the Kubernetes cluster.
Container images	Latest PostgreSQL and EDB Postgres Advanced Server containers include Barman Cloud 2.19.
Security fix	Stronger RBAC enforcement for namespaced operator installations with Operator Lifecycle Manager, including OpenShift. OpenShift users are recommended to update to this version.
Bug fix	Allow the instance manager to retry an interrupted pg_rewind by preserving a copy of the original pg_control file.
Bug fix	Clean up stale PID files before running pg_rewind.
Bug fix	Force sorting by key in primary_conninfo to avoid random restarts with PostgreSQL versions prior to 13.
Bug fix	Preserve ServiceAccount changes (e.g., labels, annotations) upon reconciliation.
Bug fix	Disable enforcement of the imagePullPolicy default value.
Bug fix	Improve initDB validation for WAL segment size.
Bug fix	Properly handle the targetLSN option when recovering a cluster with the LSN specified.
Bug fix	Fix custom TLS certificates validation by allowing a certificates chain both in the server and CA certificates.

1.3 EDB Postgres for Kubernetes 1.13.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Support for Snappy compression. Snappy is a fast compression option for backups that increase the speed of uploads to the object store using a lower compression ratio.
Feature	Support for tagging files uploaded to the Barman object store. This feature requires Barman 2.18 in the operand image. of backups after Cluster deletion.
Feature	Extension of the status of a Cluster with status.conditions. The condition ContinuousArchiving indicates that the Cluster has started to archive WAL files.
Feature	Improve the status command of the cnp plugin for kubectl with additional information: add a Cluster Summary section showing the status of the Cluster and a Certificates Status section including the status of the certificates used in the Cluster along with the time left to expire.
Feature	Support the new barman-cloud-check-wal-archive command to detect a non-empty backup destination when creating a new cluster.
Feature	Add support for using a Secret to add default monitoring queries through MONITORING_QUERIES_SECRET configuration variable.

continues on next page

Table 4 – continued from previous page

Type	Description
Feature	Allow the user to restrict container's permissions using AppArmor (on Kubernetes clusters deployed with AppArmor support).
Feature	Add Windows platform support to cnp plugin for kubectl, now the plugin is available on Windows x86 and ARM.
Feature	Drop support for Kubernetes 1.18 and deprecated API versions
Container images	PostgreSQL containers include Barman 2.18.
Security fix	Add coherence check of username field inside owner and superuser secrets; previously, a malicious user could have used the secrets to change the password of any PostgreSQL user.
Bug fix	Fix a memory leak in code fetching status from Postgres pods.
Bug fix	Disable PostgreSQL self-restart after a crash. The instance controller handles the life-cycle of the PostgreSQL instance.
Bug fix	Prevent modification of spec.postgresUID and spec.postgresGID fields in validation webhook. Changing these fields after Cluster creation makes PostgreSQL unable to start.
Bug fix	Reduce the log verbosity from the backup and WAL archiving handling code.
Bug fix	Correct a bug resulting in a Cluster being marked as Healthy when not initialized yet.
Bug fix	Allows standby servers in clusters with a very high WAL production rate to switch to streaming once they are aligned.
Bug fix	Fix a race condition during the startup of a PostgreSQL pod that could seldom lead to a crash.
Bug fix	Fix a race condition that could lead to a failure initializing the first PVC in a Cluster.
Bug fix	Remove an extra restart of a just demoted primary Pod before joining the Cluster as a replica.
Bug fix	Correctly handle replication-sensitive PostgreSQL configuration parameters when recovering from a backup.
Bug fix	Fix missing validation of PostgreSQL configurations during Cluster creation.

1.4 EDB Postgres for Kubernetes 1.12.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Add Kubernetes 1.23 to the list of supported Kubernetes distributions and remove end-to-end tests for 1.17, which ended support by the Kubernetes project in Dec 2020.
Feature	Improve the responsiveness of pod status checks in case of network issues by adding a connection timeout of 2 seconds and a communication timeout of 30 seconds. This change sets a limit on the time the operator waits for a pod to report its status before declaring it as failed, enhancing the robustness and predictability of a failover operation.
Feature	Introduce the .spec.inheritedMetadata field to the Cluster allowing the user to specify labels and annotations that will apply to all objects generated by the Cluster.
Feature	Reduce the number of queries executed when calculating the status of an instance.
Feature	Add a readiness probe for PgBouncer.
Feature	Add support for custom Certification Authority of the endpoint of Barman's backup object store when using Azure protocol.

continues on next page

Table 5 – continued from previous page

Type	Description
Bug fix	During a failover, wait to select a new primary until all the WAL streaming connections are closed. The operator now sets by default <code>wal_sender_timeout</code> and <code>wal_receiver_timeout</code> to 5 seconds to make sure standby nodes will quickly notice if the primary has network issues.
Bug fix	Change WAL archiving strategy in replica clusters to fix rolling updates by setting “archive_mode” to “always” for any PostgreSQL instance in a replica cluster. We then restrict the upload of the WAL only from the current and target designated primary. A WAL may be uploaded twice during switchovers, which is not an issue.
Bug fix	Fix support for custom Certification Authority of the endpoint of Barman’s backup object store in replica clusters source.
Bug fix	Use a fixed name for default monitoring config map in the cluster namespace.
Bug fix	If the defaulting webhook is not working for any reason, the operator now updates the Cluster with the defaults also during the reconciliation cycle.
Bug fix	Fix the comparison of resource requests and limits to fix a rare issue leading to an update of all the pods on every reconciliation cycle.
Bug fix	Improve log messages from webhooks to also include the object namespace.
Bug fix	Stop logging a “default” message at the start of every reconciliation loop.
Bug fix	Stop logging a PodMonitor deletion on every reconciliation cycle if <code>enablePodMonitor</code> is false.
Bug fix	Do not complain about possible architecture mismatch if a pod is not reachable.

1.5 EDB Postgres for Kubernetes 1.11.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Parallel WAL archiving and restore: allow the database to keep up with WAL generation on high write systems by introducing the <code>backupObjectStore.maxParallel</code> option to set the maximum number of parallel jobs to be executed during both WAL archiving (by PostgreSQL’s <code>archive_command</code>) and WAL restore (by <code>restore_command</code>). Using parallel restore option can allow newly promoted Standbys to get to a ready state faster by fetching needed WAL files to replay in parallel rather than sequentially.
Feature	Default set of metrics for monitoring: a new ConfigMap called <code>default-monitoring</code> is automatically deployed in the same namespace of the operator and, by default, added to any existing Postgres cluster. Such behavior can be changed globally by setting the <code>MONITORING_QUERIES_CONFIGMAP</code> parameter in the operator’s configuration, or at cluster level through the <code>.spec.monitoring</code> .
Feature	<code>disableDefaultQueries</code> option (by default set to false).
Feature	Introduce the <code>enablePodMonitor</code> option in the monitoring section of a cluster to automatically manage a PodMonitor resource and seamlessly integrate with Prometheus.
Feature	Improve the PostgreSQL shutdown procedure by trying to execute a smart shutdown for the first half of the desired <code>stopDelay</code> time, and a fast shutdown for the remaining half, before the pod is killed by Kubernetes.
Feature	Add the <code>switchoverDelay</code> option to control the time given to the former primary to shut down gracefully and archive all the WAL files before promoting the new primary (by default, Cloud Native PostgreSQL waits indefinitely to privilege data durability).
Feature	Handle changes to resource requests and limits for a PostgreSQL Cluster by issuing a rolling update.

continues on next page

Table 6 – continued from previous page

Type	Description
Feature	Improve the status command of the cnpg plugin for kubectl with additional information: streaming replication status, total size of the database, role of an instance in the cluster.
Feature	Enhance support of workloads with many parallel workers by enabling configuration of the <code>dynamic_shared_memory_type</code> and <code>shared_memory_type</code> parameters for PostgreSQL's management of shared memory.
Feature	Propagate labels and annotations defined at cluster level to the associated resources, including pods (deletions are not supported).
Feature	Automatically remove pods that have been evicted by the Kubelet.
Feature	Manage automated resizing of persistent volumes in Azure through the <code>ENABLE_AZURE_PVC_UPDATES</code> operator configuration option, by issuing a rolling update of the cluster if needed (disabled by default).
Feature	Introduce the <code>the8s.enterprisedb.io/reconciliationLoop</code> annotation that, when set to disabled on a given Postgres cluster, prevents the reconciliation loop from running.
Feature	Introduce the <code>postInitApplicationSQL</code> option as part of the <code>initdb</code> bootstrap method to specify a list of SQL queries to be executed on the main application database as a superuser immediately after the cluster has been created.
Feature	Support for EDB Postgres Advanced 14.2.
Bug fix	Liveness probe now correctly handles the startup process of a PostgreSQL server. This fixes an issue reported by a few customers and affects a restarted standby server that needs to recover WAL files to reach a consistent state, but it was not able to do it before the timeout of liveness probe would kick in, leaving the pods in <code>CrashLoopBackOff</code> status.
Bug fix	Liveness probe now correctly handles the case of a former primary that needs to use <code>pg_rewind</code> to re-align with the current primary after a timeline diversion. This fixes the pod of the new standby from repeatedly being killed by Kubernetes.
Bug fix	Reduce client-side throttling from Postgres pods (e.g. Waited for 1.182388649s due to client-side throttling, not priority and fairness, request: GET).
Bug fix	Disable Public Key Infrastructure (PKI) initialization on OpenShift and OLM installations, by using the provided one.
Bug fix	When changing configuration parameters that require a restart, always leave the primary as last.
Bug fix	Mark a PVC to be ready only after a job has been completed successfully, preventing a race condition in PVC initialization.
Bug fix	Use the correct public key when renewing the expired webhook TLS secret.
Bug fix	Fix an overflow when parsing an LSN.
Bug fix	Remove stale PID files at startup.
Bug fix	Let the Pooler resource inherit the <code>imagePullSecret</code> defined in the operator, if exists.

1.6 EDB Postgres for Kubernetes 1.10.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Connection Pooling with PgBouncer: introduce the Pooler resource and controller to automatically manage a PgBouncer deployment to be used as a connection pooler for a local PostgreSQL Cluster. The feature includes TLS client/server connections, password authentication, High Availability, pod templates support, configuration of key PgBouncer parameters, PAUSE/RESUME, logging in JSON format, Prometheus exporter for stats, pools, and lists.
Feature	Backup Retention Policies: support definition of recovery window retention policies for backups (e.g. '30d' to ensure a recovery window of 30 days).
Feature	In-Place updates of the operator: introduce an in-place online update of the instance manager, which removes the need to perform a rolling update of the entire cluster following an update of the operator. By default this option is disabled (please refer to the documentation for more detailed information).
Feature	Limit the list of options that can be customized in the initdb bootstrap method to dataChecksums, encoding, localeCollate, localeCTYPE, walSegmentSize. This makes the options array obsolete and planned to be removed in the v2 API.
Feature	Introduce the postInitTemplateSQL option as part of the initdb bootstrap method to specify a list of SQL queries to be executed on the template1 database as a superuser immediately after the cluster has been created. This feature allows you to include default objects in all application databases created in the cluster.
Feature	New default metrics added to the instance Prometheus exporter: Postgres version, cluster name, and first point of recoverability according to the backup catalog.
Feature	Retry taking a backup after a failure.
Feature	Build awareness about Barman Cloud capabilities in order to prevent the operator from invoking recently introduced features (such as retention policies, or Azure Blob Container storage) that are not present in operand images that are not frequently updated.
Feature	Integrate the output of the status command of the cnp plugin with information about the backup.
Feature	Introduce a new annotation that reports the status of a PVC (being initialized or ready).
Feature	Set the cluster name in the k8s.enterprisedb.io/cluster label for every object generated in a Cluster, including Backup objects.
Feature	Drop support for deprecated API version postgresql.k8s.enterprisedb.io/v1alpha1 on the Cluster, Backup, and ScheduledBackup kinds.
Feature	Set default operand image to PostgreSQL 14.2.
Security fix	Set allowPrivilegeEscalation to false for the operator containers securityContext.
Bug fix	Disable primary PodDisruptionBudget during maintenance in single-instance clusters.
Bug fix	Use the correct certificate certification authority (CA) during recovery operations.
Bug fix	Prevent Postgres connection leaking when checking WAL archiving status before taking a backup.
Bug fix	Let WAL archive/restore sleep for 100ms following transient errors that would flood logs otherwise.

1.7 EDB Postgres for Kubernetes 1.9.2 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Enhance JSON log with two new loggers: wal-archive for PostgreSQL's archive_command, and wal-restore for restore_command in a standby.

continues on next page

Table 8 – continued from previous page

Type	Description
Bug fix	Enable WAL archiving during the standby promotion (prevented .history files from being archived).
Bug fix	Pass the <code>--cloud-provider</code> option to Barman Cloud tools only when using Barman 2.13 or higher to avoid errors with older operands.
Bug fix	Wait for the pod of the primary to be ready before triggering a backup.

1.8 EDB Postgres for Kubernetes 1.9.1 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	This release is to celebrate the launch of PostgreSQL 14 by making it the default major version when a new Cluster is created without defining a specific image name.
Bug fix	Fix issue causing Error while getting barman endpoint CA secret message to appear in the logs of the primary pod, which prevented the backup to work correctly.
Bug fix	Properly retry requesting a new backup in case of temporary communication issues with the instance manager.

1.9 EDB Postgres for Kubernetes 1.9.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Add Kubernetes 1.22 to the list of supported Kubernetes distributions, and remove 1.16.
Feature	Introduce support for the <code>--restore-target-wal</code> option in <code>pg_rewind</code> , in order to fetch WAL files from the backup archive, if necessary (available only with PostgreSQL/EPAS 13+).
Feature	Version 1.9.0 is not available on OpenShift due to delays with the release process and the subsequent release of version 1.9.1.
Feature	Expose a default metric for the Prometheus exporter that estimates the number of pages in the <code>pg_catalog.pg_largeobject</code> table in each database.
Feature	Enhance the performance of WAL archiving and fetching, through local in-memory cache.
Bug fix	Explicitly set the postgres user when invoking <code>pg_isready</code> - required by restricted SCC in OpenShift.
Bug fix	Properly update the <code>FirstRecoverabilityPoint</code> in the status.
Bug fix	Set <code>archive_mode = always</code> on the designated primary if backup is requested.

1.10 EDB Postgres for Kubernetes 1.8.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Bootstrap a new cluster via full or Point-In-Time Recovery directly from an object store defined in the external cluster section, eliminating the previous requirement to have a Backup CR defined.
Feature	Introduce the immediate option in scheduled backups to request a backup immediately after the first Postgres instance running, adding the capability to rewind to the very beginning of a cluster when Point-In-Time Recovery is configured.
Feature	Add the firstRecoverabilityPoint in the cluster status to report the oldest consistent point in time to request a recovery based on the backup object store's content.
Feature	Enhance the default Prometheus exporter for a PostgreSQL instance by exposing the following new metrics: number of WAL files and computed total size on disk, number of WAL files and computed total size on disk, number of .ready and .done files in the archive status folder, flag for replica mode, number of requested minimum/maximum synchronous replicas, as well as the expected and actually observed ones.
Feature	Add support for the runonserver option when defining custom metrics in the Prometheus exporter to limit the collection of a metric to a range of PostgreSQL versions.
Feature	Natively support Azure Blob Storage for backup and recovery, by taking advantage of the feature introduced in Barman 2.13 for Barman Cloud.
Feature	Rely on pg_isready for the liveness probe.
Feature	Support RFC3339 format for timestamp specification in recovery target times.
Feature	Introduce .spec.imagePullPolicy to control the pull policy of image containers for all pods and jobs created for a cluster.
Feature	Add support for OpenShift 4.8, which replaces OpenShift 4.5.
Feature	Support PostgreSQL 14 (beta).
Feature	Enhance the replica cluster feature with cross-cluster replication from an object store defined in an external cluster section, without requiring a streaming connection (experimental).
Feature	Introduce logLevel option to the cluster's spec to specify one of the following levels: error, info, debug or trace.
Security fix	Introduce .spec.enableSuperuserAccess to enable/disable network access with the postgres user through password authentication.
Security fix	Enable specification of a license key in a secret with spec.licenseKeySecret.
Bug fix	Properly inform users when a cluster enters an unrecoverable state and requires human intervention.

1.11 EDB Postgres for Kubernetes 1.7.1 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Prefer self-healing over configuration with regards to synchronous replication, empowering the operator to temporarily override minSyncReplicas and maxSyncReplicas settings in case the cluster is not able to meet the requirements during self-healing operations.

continues on next page

Table 12 – continued from previous page

Type	Description
Feature	Introduce the postInitSQL option as part of the initdb bootstrap method to specify a list of SQL queries to be executed as a superuser immediately after the cluster has been created.
Bug fix	Allow the operator to failover when the primary is not ready (bug introduced in 1.7.0).
Bug fix	Execute administrative queries using the LOCAL synchronous commit level.
Bug fix	Correctly parse multi-line log entries in PGAudit.

1.12 EDB Postgres for Kubernetes 1.7.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Add native support to PGAudit with a new type of logger called pgaudit directly available in the JSON output.
Feature	Native support for the pg_stat_statements and auto_explain extensions.
Feature	The target_databases option in the Prometheus exporter to run a user-defined metric query on one or more databases (including auto-discovery of databases through shell-like pattern matching).
Feature	Exposure of the manual_switchover_required metric to promptly report whether a cluster with primaryUpdateStrategy set to supervised requires a manual switchover.
Feature	Transparently handle shared_preload_libraries for pg_audit, auto_explain and pg_stat_statements.
Feature	Automatic configuration of shared_preload_libraries for PostgreSQL when pg_stat_statements, pgaudit or auto_explain options are added to the postgresql parameters section.
Feature	Support the k8s.enterprisedb.io/reload label to finely control the automated reload of config maps and secrets, including those used for custom monitoring/alerting metrics in the Prometheus exporter or to store certificates.
Feature	Add the reload command to the cnf plugin for kubectl to trigger a reconciliation loop on the instances.
Feature	Improve control of pod affinity and anti-affinity configurations through additionalPodAffinity and additionalPodAntiAffinity.
Feature	Introduce a separate PodDisruptionBudget for primary instances, by requiring at least a primary instance to run at any time.
Security fix	Add the .spec.certificates.clientCASecret and spec.certificates.replicationTLSSecret options to define custom client Certification.
Security fix	Authority and certificate for the PostgreSQL server, to be used to authenticate client certificates and secure communication between PostgreSQL nodes.
Security fix	Add the .spec.backup.barmanObjectStore.endpointCA option to define the custom Certification Authority bundle of the endpoint of Barman's backup object store.
Bug fix	Correctly parse histograms in the Prometheus exporter.
Bug fix	Reconcile services created by the operator for a cluster.

1.13 EDB Postgres for Kubernetes 1.6.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Replica mode (EXPERIMENTAL): allow a cluster to be created as a replica of a source cluster. A replica cluster has a designated primary and any number of standbys.
Feature	EDB Audit support on EDB Postgres Advanced Server images.
Feature	Add the <code>.spec.postgresql.promotionTimeout</code> parameter to specify the maximum amount of seconds to wait when promoting an instance to primary, defaulting to 40000000 seconds.
Feature	Add the <code>.spec.affinity.podAntiAffinityType</code> parameter. It can be set to preferred (default), resulting in preferredDuringSchedulingIgnoredDuringExecution being used, or to required, resulting in requiredDuringSchedulingIgnoredDuringExecution.
Security fix	Prevent license keys from appearing in the logs.
Change	Fixed a race condition when deleting a PVC and a pod which prevented the operator from creating a new pod.
Change	Fixed a race condition preventing the manager from detecting the need for a PostgreSQL restart on a configuration change.
Change	Fixed a panic in kubectl-cnp on clusters without annotations.
Change	Lowered the level of some log messages to debug.
Change	E2E tests for server CA and TLS injection.

1.14 EDB Postgres for Kubernetes 1.5.1 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Change	Fix a bug with CRD validation preventing auto-update with Operator Deployments on Red Hat OpenShift.
Change	Allow passing operator's configuration using a Secret.

1.15 EDB Postgres for Kubernetes 1.5.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Introduce the <code>pg_basebackup</code> bootstrap method to create a new PostgreSQL cluster as a copy of an existing PostgreSQL instance of the same major version, even outside Kubernetes.
Feature	Add support for Kubernetes' tolerations in the Affinity section of the Cluster resource, allowing users to distribute PostgreSQL instances on Kubernetes nodes with the required taint.
Feature	Enable specification of a digest to an image name, through the <code><image>:<tag>@sha256:<digestValue></code> format, for more deterministic and repeatable deployments.

continues on next page

Table 16 – continued from previous page

Type	Description
Security fix	Customize TLS certificates to authenticate the PostgreSQL server by defining secrets for the server certificate and the related Certification Authority that signed it.
Security fix	Raise the sslmode for the WAL receiver process of internal and automatically managed streaming replicas from require to verify-ca.
Change	Enhance the promote subcommand of the cnp plugin for kubectl to accept just the node number rather than the whole name of the pod.
Change	Adopt DNS-1035 validation scheme for cluster names (from which service names are inherited).
Change	Enforce streaming replication connection when cloning a standby instance or when bootstrapping using the pg_basebackup method.
Change	Integrate the Backup resource with beginWal, endWal, beginLSN, endLSN, startedAt and stoppedAt regarding the physical base backup.
Documentation fix	Provide a list of ports exposed by the operator and the operand container.
Documentation fix	Introduce the cnp-bench helm charts and guidelines for benchmarking the storage and PostgreSQL for database workloads.
E2E test fix	Test Kubernetes 1.21.
E2E test fix	Add test for High Availability of the operator.
E2E test fix	Add test for node draining.
Bug fix	Timeout to pg_ctl start during recovery operations too short.
Bug fix	Operator not watching over direct events on PVCs.
Bug fix	Fix handling of immediateCheckpoint and jobs parameter in barmanObjectStore backups.
Bug fix	Empty logs when recovering from a backup.

1.16 EDB Postgres for Kubernetes 1.4.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Standard output logging of PostgreSQL error messages in JSON format.
Feature	Provide a basic set of PostgreSQL metrics for the Prometheus exporter.
Feature	Add the restart command to the cnp plugin for kubectl to restart the pods of a given PostgreSQL cluster in a rollout fashion.
Security fix	Set readOnlyRootFilesystem security context for pods.
Change	IMPORTANT: If you have previously deployed the Cloud Native PostgreSQL operator using the YAML manifest, you must delete the existing operator deployment before installing the new version. This is required to avoid conflicts with other Kubernetes API's due to a change in labels and label selectors being directly managed by the operator. Please refer to the Cloud Native PostgreSQL documentation for additional detail on upgrading to 1.4.0.
Change	Fix the labels that are automatically defined by the operator, renaming them from control-plane: controller-manager to app.kubernetes.io/name: cloud-native-postgresql.
Change	Assign the metrics name to the TCP port for the Prometheus exporter.
Change	Set cnp_metrics_exporter as the application_name to the metrics exporter connection in PostgreSQL.
Change	When available, use the application database for monitoring queries of the Prometheus exporter instead of the postgres database.

continues on next page

Table 17 – continued from previous page

Type	Description
Documentation fixes	Customization of monitoring queries.
Documentation fixes	Operator upgrade instructions.
Bug fix	Avoid using -R when calling pg_basebackup.
Bug fix	Remove stack trace from error log when getting the status.

1.17 EDB Postgres for Kubernetes 1.3.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Inheritance of labels and annotations.
Feature	Set resource limits for every container.
Security fix	Support for restricted security context constraint on Red Hat OpenShift to limit pod execution to a namespace allocated UID and SELinux context.
Security fix	Pod security contexts explicitly defined by the operator to run as non-root, non-privileged and without privilege escalation.
Change	Prometheus exporter endpoint listening on port 9187 (port 8000 is now reserved to instance coordination with API server).

1.18 EDB Postgres for Kubernetes 1.2.1 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	ScheduledBackup are no longer owners of the Backups, meaning that backups are not removed when ScheduledBackup objects are deleted.
Security fix	Update on ubi8-minimal image to solve RHSA-2021:1024 (Security Advisory: Important).

1.19 EDB Postgres for Kubernetes 1.2.0 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Introduce experimental support for custom monitoring queries as ConfigMap and Secret objects using a compatible syntax with postgres_exporter for Prometheus.
Feature	Support Operator Lifecycle Manager (OLM) deployments, with the subsequent presence on OperatorHub.io.
Feature	Expand license key support for company-wide usage (previous restrictions limited only to a single cluster namespace).
Feature	Enhance container security by applying guidelines from the US Department of Defense (DoD)'s Defense Information Systems Agency (DISA) and the Center for Internet Security (CIS) and verifying them directly in the pipeline with Dockle.

1.20 EDB Postgres for Kubernetes 1.1 release notes

This release of EDB Postgres for Kubernetes includes the following:

Type	Description
Feature	Add kubectlnp status to pretty-print the status of a cluster, including JSON and YAML output.
Feature	Add kubectlnp certificate to enable TLS authentication for client applications.
Feature	Add the -ro service to route connections to the available hot standby replicas only, enabling offload of read-only queries from the cluster's primary instance.
Feature	Rollback scaling down a cluster to a value lower than maxSyncReplicas.
Feature	Request a checkpoint before demoting a former primary.
Feature	Send SIGINT signal (fast shutdown) to PostgreSQL process on SIGTERM.

1.21 EDB Postgres for Kubernetes 1.0 release notes

The first major release implements Cluster, Backup and ScheduledBackup in the API group postgres.k8s.enterprisedb.io/v1. It uses these resources to create and manage PostgreSQL clusters inside Kubernetes with the following main capabilities:

Type	Description
Feature	Direct integration with Kubernetes API server for High Availability, without requiring an external tool.
Feature	Failover of the primary instance by promoting the most aligned replica.
Feature	Automated recreation of a replica.
Feature	Planned switchover of the primary instance by promoting a selected replica.
Feature	Scale up/down capabilities.
Feature	Definition of an arbitrary number of instances (minimum 1 - one primary server).
Feature	Definition of the read-write service to connect your applications to the only primary server of the cluster.
Feature	Definition of the read service to connect your applications to any of the instances for reading workloads.
Feature	Support for Local Persistent Volumes with PVC templates.
Feature	Reuse of Persistent Volumes storage in Pods.
Feature	Rolling updates for PostgreSQL minor versions and operator upgrades.
Feature	TLS connections and client certificate authentication.
Feature	Continuous backup to an S3 compatible object store.
Feature	Full recovery and point-in-time recovery from an S3 compatible object store backup.
Feature	Support for synchronous replicas.
Feature	Support for node affinity via nodeSelector property.
Feature	Standard output logging of PostgreSQL error messages.

‘DIFFERENCES WITH CLOUDNATIVEPG V1’

The EDB Postgres for Kubernetes operator is a fork based on CloudNativePG.

Despite both defining a customer resource called Cluster, they have two different API groups. As a result, they can co-exist in the same Kubernetes cluster. Specifically:

- CloudNativePG uses the postgresql.cnpg.io/v1 API version
- EDB Postgres for Kubernetes uses the postgresql.k8s.enterprisedb.io/v1 API version

Important: Existing EDB customers with deployments of Cloud Native PostgreSQL might have noticed that the API group for version 1 has not changed with the introduction of EDB Postgres for Kubernetes.

As a general rule, all features available in the open source CloudNativePG are also available in EDB Postgres for Kubernetes - provided the correct API version is used.

Refer to the CloudNativePG documentation for detailed information on the available features.

‘DIFFERENCES WITH CLOUDNATIVEPG’

The EDB Postgres for Kubernetes operator is a fork based on CloudNativePG.

Despite both defining a customer resource called `Cluster`, they have two different API groups. As a result, they can co-exist in the same Kubernetes cluster. Specifically:

- CloudNativePG uses the `postgresql.cnpg.io/v1` API version
- EDB Postgres for Kubernetes uses the `postgresql.k8s.enterprisedb.io/v1` API version

Important: Existing EDB customers with deployments of Cloud Native PostgreSQL might have noticed that the API group for version 1 has not changed with the introduction of EDB Postgres for Kubernetes.

As a general rule, all features available in the open source CloudNativePG are also available in EDB Postgres for Kubernetes - provided the correct API version is used.

Refer to the [CloudNativePG documentation](#) for detailed information on the available features.

‘INSTALLATION AND UPGRADES’

The following supplements the ‘Installation and upgrades’ topic in the upstream CloudNativePG project.

4.1 Installation on Kubernetes

4.1.1 Directly using the operator manifest

The operator can be installed like any other resource in Kubernetes, through a YAML manifest applied via `kubectl`.

You can install the [latest operator manifest](#) as follows:

```
kubectl apply -f \
  https://get.enterprisedb.io/cnp/postgresql-operator-1.15.0.yaml
```

Once you have run the `kubectl` command, EDB Postgres for Kubernetes will be installed in your Kubernetes cluster.

You can verify that with:

```
kubectl get deploy -n postgresql-operator-system postgresql-operator-controller-manager
```

<!-- THIS NEEDS TO BE DONE### Using the Operator Lifecycle Manager (OLM)

OperatorHub is a community-sourced index of operators available via the [Operator Lifecycle Manager](#), which is a package managing system for operators.

You can install EDB Postgres for Kubernetes using the metadata available in the [EDB Postgres for Kubernetes page](#) from the [OperatorHub.io website](#), following the installation steps listed on that page.

- ->

4.2 Details about the deployment

Differently from CloudNativePG, the operator is installed in the `postgresql-operator-system` namespace as a Kubernetes Deployment called `postgresql-operator-controller-manager` by default. You can get more information by running:

```
kubectl describe deploy \
  -n postgresql-operator-system \
  postgresql-operator-controller-manager
```


‘QUICKSTART’

Refer to the ‘Installation and upgrades’ topic in the CloudNativePG documentation and use this topic for specifics regarding EDB Postgres for Kubernetes usage.

Red Hat OpenShift Container Platform users can test the certified operator for EDB Postgres for Kubernetes on the [Red Hat CodeReady Containers \(CRC\)](#) for OpenShift.

Warning: The instructions contained in this section are for demonstration, testing, and practice purposes only and must not be used in production.

Like any other Kubernetes application, EDB Postgres for Kubernetes is deployed using regular manifests written in YAML.

By following these instructions you should be able to start a PostgreSQL cluster on your local Kubernetes/OpenShift installation and experiment with it.

Important: Make sure that you have *kubectl* installed on your machine in order to connect to the Kubernetes cluster, or *oc* if using CRC for OpenShift. Please follow the Kubernetes documentation on *how to install kubectl* <<https://kubernetes.io/docs/tasks/tools/install-kubectl/>>`\_\_ or the OpenShift documentation on *how to install oc* <[https://docs.openshift.com/container-platform/4.6/cli\\_reference/openshift\\_cli/getting-started-cli.html](https://docs.openshift.com/container-platform/4.6/cli_reference/openshift_cli/getting-started-cli.html)>`__ .

Note: If you are running OpenShift, use *oc* every time *kubectl* is mentioned in this documentation. *kubectl* commands are compatible with *oc* ones.

5.1 Part 1 - Setup the local Kubernetes/OpenShift playground

This section covers the installation of CRC. For Kind or Minikube, refer to the upstream documentation for CloudNativePG.

5.1.1 CodeReady Containers (CRC)

1. Download [Red Hat CRC](#) and move the binary inside a directory in your PATH.
2. Run the following commands:

```
crc setup
crc start
```

The `crc start` output explains how to proceed. 1. Execute the output of the `crc oc-env` command. 1. Log in as `kubeadmin` with the printed `oc login` command. You can also open the web console running `crc console`. User and password are the same as for the `oc login` command.

1. CRC doesn't come with a StorageClass, so one has to be configured. Follow the [Dynamic volume provisioning wiki page](#) and install `rancher/local-path-provisioner`.

5.2 Part 2 - Install EDB Postgres for Kubernetes

Now that you have a Kubernetes or OpenShift installation up and running on your laptop, you can proceed with EDB Postgres for Kubernetes installation.

Refer to the 'Installation and upgrades' topic in the CloudNativePG documentation and then proceed with the deployment of a PostgreSQL cluster.

5.3 Part 3 - Deploy a PostgreSQL cluster

As with any other deployment in Kubernetes, to deploy a PostgreSQL cluster you need to apply a configuration file that defines your desired Cluster.

The `cluster-example.yaml` sample file defines a simple Cluster using the default storage class to allocate disk space:

```
# Example of PostgreSQL cluster
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3

  # Require 1Gi of space
  storage:
    size: 1Gi
```

INSTALLATION, CONFIGURATION AND DEPLOYMENT DEMO

Want to see what it takes to get the EDB Postgres for Kubernetes Operator up and running? This section will demonstrate the following:

1. Installing the EDB Postgres for Kubernetes Operator
2. Deploying a three-node PostgreSQL cluster
3. Installing and using the kubectlnp plugin
4. Testing failover to verify the resilience of the cluster

It will take roughly 5-10 minutes to work through.

Once `k3d` is ready, we need to start a cluster:

```
k3d cluster create
__OUTPUT__
INFO[0000] Prep: Network
INFO[0000] Created network k3d-k3s-default
INFO[0000] Created image volume k3d-k3s-default-images
INFO[0000] Starting new tools node...
INFO[0001] Pulling image ghcr.io/k3d-io/k3d-tools:5.4.1
INFO[0001] Creating node k3d-k3s-default-server-0
INFO[0002] Pulling image docker.io/rancher/k3s:v1.22.7-k3s1
INFO[0003] Starting Node k3d-k3s-default-tools
INFO[0007] Creating LoadBalancer k3d-k3s-default-serverlb
INFO[0008] Pulling image ghcr.io/k3d-io/k3d-proxy:5.4.1
INFO[0011] Using the k3d-tools node to gather environment information
INFO[0011] HostIP: using network gateway 172.17.0.1 address
INFO[0011] Starting cluster k3s-default
INFO[0011] Starting servers...
INFO[0011] Starting Node k3d-k3s-default-server-0
INFO[0016] All agents already running.
INFO[0016] Starting helpers...
INFO[0017] Starting Node k3d-k3s-default-serverlb
INFO[0024] Injecting records for hostAliases (incl. host.k3d.internal) and for 2 network
members into CoreDNS configmap...
INFO[0026] Cluster k3s-default created successfully!
INFO[0026] You can now use it like this:
kubectlnp cluster-info
```

This will create the Kubernetes cluster, and you will be ready to use it. Verify that it works with the following command:

```
kubectlnp get nodes
__OUTPUT__
NAME                                STATUS    ROLES                                AGE     VERSION
k3d-k3s-default-server-0            Ready    control-plane,master                35s     v1.22.7+k3s1
```

You will see one node called `k3d-k3s-default-server-0`. If the status isn't yet "Ready", wait for a few seconds and run the command above again.

6.1 Install EDB Postgres for Kubernetes

Now that the Kubernetes cluster is running, you can proceed with EDB Postgres for Kubernetes installation as described in the 'Installation and upgrades' section:

```
kubectl apply -f https://get.enterprisedb.io/cnp/postgresql-operator-1.15.0.yaml
__OUTPUT__
namespace/postgresql-operator-system created
customresourcedefinition.apiextensions.k8s.io/backups.postgresql.k8s.enterprisedb.io
↳ created
customresourcedefinition.apiextensions.k8s.io/clusters.postgresql.k8s.enterprisedb.io
↳ created
customresourcedefinition.apiextensions.k8s.io/poolers.postgresql.k8s.enterprisedb.io
↳ created
customresourcedefinition.apiextensions.k8s.io/scheduledbackups.postgresql.k8s.
↳ enterprisedb.io created
serviceaccount/postgresql-operator-manager created
clusterrole.rbac.authorization.k8s.io/postgresql-operator-manager created
clusterrolebinding.rbac.authorization.k8s.io/postgresql-operator-manager-rolebinding
↳ created
configmap/postgresql-operator-default-monitoring created
service/postgresql-operator-webhook-service created
deployment.apps/postgresql-operator-controller-manager created
mutatingwebhookconfiguration.admissionregistration.k8s.io/postgresql-operator-mutating-
↳ webhook-configuration created
validatingwebhookconfiguration.admissionregistration.k8s.io/postgresql-operator-
↳ validating-webhook-configuration created
```

And then verify that it was successfully installed:

```
kubectl get deploy -n postgresql-operator-system postgresql-operator-controller-manager
__OUTPUT__
NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
postgresql-operator-controller-manager 1/1      1              1            52s
```

6.2 Deploy a PostgreSQL cluster

As with any other deployment in Kubernetes, to deploy a PostgreSQL cluster you need to apply a configuration file that defines your desired Cluster.

The `cluster-example.yaml` sample file defines a simple Cluster using the default storage class to allocate disk space:

```
cat <<EOF > cluster-example.yaml
# Example of PostgreSQL cluster
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
```

(continues on next page)

(continued from previous page)

```

name: cluster-example
spec:
  instances: 3

  # Example of rolling update strategy:
  # - unsupervised: automated update of the primary once all
  #                   replicas have been upgraded (default)
  # - supervised: requires manual supervision to perform
  #                   the switchover of the primary
  primaryUpdateStrategy: unsupervised

  # Require 1Gi of space
  storage:
    size: 1Gi
EOF

```

In order to create the 3-node PostgreSQL cluster, you need to run the following command:

```

kubectl apply -f cluster-example.yaml
__OUTPUT__
cluster.postgresql.k8s.enterprisedb.io/cluster-example created

```

You can check that the pods are being created with the `get pods` command. It'll take a bit to initialize, so if you run that immediately after applying the cluster configuration you'll see the status as `Init:` or `PodInitializing`:

```

kubectl get pods
__OUTPUT__

```

| NAME | READY | STATUS | RESTARTS | AGE |
|-----------------------------------|-------|---------|----------|-----|
| cluster-example-1-initdb--1-2cqfw | 0/1 | Pending | 0 | 3s |

...give it a minute, and then check on it again:

```

kubectl get pods
__OUTPUT__

```

| NAME | READY | STATUS | RESTARTS | AGE |
|-------------------|-------|---------|----------|-----|
| cluster-example-1 | 1/1 | Running | 0 | 56s |
| cluster-example-2 | 1/1 | Running | 0 | 35s |
| cluster-example-3 | 1/1 | Running | 0 | 19s |

Now we can check the status of the cluster:

```

kubectl get cluster cluster-example -o yaml
__OUTPUT__
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"postgresql.k8s.enterprisedb.io/v1","kind":"Cluster","metadata":{"
↪ "annotations":{},"name":"cluster-example","namespace":"default"},"spec":{"instances":3,
↪ "primaryUpdateStrategy":"unsupervised","storage":{"size":"1Gi"}}}
  creationTimestamp: "2022-04-21T18:33:12Z"
  generation: 1

```

(continues on next page)

(continued from previous page)

```

name: cluster-example
namespace: default
resourceVersion: "1868"
uid: bf85fdbf-4404-46a0-a16a-63676a377d90
spec:
  affinity:
    podAntiAffinityType: preferred
    topologyKey: ""
  bootstrap:
    initdb:
      database: app
      encoding: UTF8
      localeCTYPE: C
      localeCollate: C
      owner: app
  enableSuperuserAccess: true
  imageName: quay.io/enterprisedb/postgresql:14.2
  instances: 3
  logLevel: info
  maxSyncReplicas: 0
  minSyncReplicas: 0
  monitoring:
    customQueriesConfigMap:
      - key: queries
        name: postgresql-operator-default-monitoring
    disableDefaultQueries: false
    enablePodMonitor: false
  postgresGID: 26
  postgresUID: 26
  postgresql:
    parameters:
      archive_mode: "on"
      dynamic_shared_memory_type: posix
      log_destination: csvlog
      log_directory: /controller/log
      log_filename: postgres
      log_rotation_age: "0"
      log_rotation_size: "0"
      log_truncate_on_rotation: "false"
      logging_collector: "on"
      max_parallel_workers: "32"
      max_replication_slots: "32"
      max_worker_processes: "32"
      shared_memory_type: mmap
      shared_preload_libraries: ""
      wal_keep_size: 512MB
      wal_receiver_timeout: 5s
      wal_sender_timeout: 5s
  primaryUpdateMethod: switchover
  primaryUpdateStrategy: unsupervised
  resources: {}
  startDelay: 30

```

(continues on next page)

(continued from previous page)

```

stopDelay: 30
storage:
  resizeInUseVolumes: true
  size: 1Gi
switchoverDelay: 40000000
status:
  certificates:
    clientCASecret: cluster-example-ca
    expirations:
      cluster-example-ca: 2022-07-20 18:28:12 +0000 UTC
      cluster-example-replication: 2022-07-20 18:28:12 +0000 UTC
      cluster-example-server: 2022-07-20 18:28:12 +0000 UTC
    replicationTLSSecret: cluster-example-replication
    serverAltDNSNames:
      - cluster-example-rw
      - cluster-example-rw.default
      - cluster-example-rw.default.svc
      - cluster-example-r
      - cluster-example-r.default
      - cluster-example-r.default.svc
      - cluster-example-ro
      - cluster-example-ro.default
      - cluster-example-ro.default.svc
    serverCASecret: cluster-example-ca
    serverTLSSecret: cluster-example-server
  cloudNativePostgresqlCommitHash: 0531b739
  cloudNativePostgresqlOperatorHash: ↵
↵1dd0fcb2f3fa9e7293fa5696f0cd7dd560923cc9f7d714b75c10d282072fcf2d
  configMapResourceVersion:
    metrics:
      postgresql-operator-default-monitoring: "848"
  currentPrimary: cluster-example-1
  currentPrimaryTimestamp: "2022-04-21T18:33:49Z"
  healthyPVC:
    - cluster-example-1
    - cluster-example-2
    - cluster-example-3
  instances: 3
  instancesStatus:
    healthy:
      - cluster-example-1
      - cluster-example-2
      - cluster-example-3
  latestGeneratedNode: 3
  licenseStatus:
    isImplicit: true
    isTrial: true
    licenseExpiration: "2022-05-21T18:33:12Z"
    licenseStatus: Implicit trial license
    repositoryAccess: false
    valid: true
  phase: Cluster in healthy state

```

(continues on next page)

(continued from previous page)

```

poolerIntegrations:
  pgBouncerIntegration: {}
pvcCount: 3
readService: cluster-example-r
readyInstances: 3
secretsResourceVersion:
  applicationSecretVersion: "813"
  clientCaSecretVersion: "809"
  replicationSecretVersion: "811"
  serverCaSecretVersion: "809"
  serverSecretVersion: "810"
  superuserSecretVersion: "812"
targetPrimary: cluster-example-1
targetPrimaryTimestamp: "2022-04-21T18:33:13Z"
writeService: cluster-example-rw

```

Note: By default, the operator will install the latest available minor version of the latest major version of PostgreSQL when the operator was released. You can override this by setting [the `imageName` key in the `spec` section of the `Cluster` definition](`api_reference/#clusterspec`).

Important: The immutable infrastructure paradigm requires that you always point to a specific version of the container image. Never use tags like `latest` or `I3` in a production environment as it might lead to unpredictable scenarios in terms of update policies and version consistency in the cluster.

6.3 Install the kubectl-cnp plugin

EDB Postgres for Kubernetes provides a plugin for kubectl to manage a cluster in Kubernetes, along with a script to install it:

```

curl -sSfL \
  https://github.com/EnterpriseDB/kubectl-cnp/raw/main/install.sh | \
  sudo sh -s -- -b /usr/local/bin
__OUTPUT__
EnterpriseDB/kubectl-cnp info checking GitHub for latest tag
EnterpriseDB/kubectl-cnp info found version: 1.15.0 for v1.15.0/linux/x86_64
EnterpriseDB/kubectl-cnp info installed /usr/local/bin/kubectl-cnp

```

The `cnp` command is now available in kubectl:

```

kubectl cnp status cluster-example
__OUTPUT__
Cluster Summary
Name:          cluster-example
Namespace:     default
System ID:     7089127092443762706
PostgreSQL Image: quay.io/enterprisedb/postgresql:14.2
Primary instance: cluster-example-1

```

(continues on next page)

(continued from previous page)

| | | | | | | | |
|------------------------------|--|-------------|------------------|----------------------------|------------|-----------|-------|
| Status: | Cluster in healthy state | | | | | | |
| Instances: | 3 | | | | | | |
| Ready instances: | 3 | | | | | | |
| Current Write LSN: | 0/4000060 (Timeline: 1 - WAL File: 000000010000000000000004) | | | | | | |
| Certificates Status | | | | | | | |
| Certificate Name | Expiration Date | | | Days Left Until Expiration | | | |
| ----- | ----- | | | ----- | | | |
| cluster-example-replication | 2022-07-20 18:28:12 +0000 UTC | | | 89.99 | | | |
| cluster-example-server | 2022-07-20 18:28:12 +0000 UTC | | | 89.99 | | | |
| cluster-example-ca | 2022-07-20 18:28:12 +0000 UTC | | | 89.99 | | | |
| Continuous Backup status | | | | | | | |
| Not configured | | | | | | | |
| Streaming Replication status | | | | | | | |
| Name | Sent LSN | Write LSN | Flush LSN | Replay LSN | Write Lag | Flush Lag | ↵ |
| ↵Replay Lag | State | Sync State | Sync Priority | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| ↵----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| cluster-example-2 | 0/4000060 | 0/4000060 | 0/4000060 | 0/4000060 | 00:00:00 | 00:00:00 | ↵ |
| ↵00:00:00 | streaming | async | 0 | | | | |
| cluster-example-3 | 0/4000060 | 0/4000060 | 0/4000060 | 0/4000060 | 00:00:00 | 00:00:00 | ↵ |
| ↵00:00:00 | streaming | async | 0 | | | | |
| Instances status | | | | | | | |
| Name | Database Size | Current LSN | Replication role | Status | QoS | | ↵ |
| ↵Manager Version | | | | | | | |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| ↵----- | | | | | | | |
| cluster-example-1 | 33 MB | 0/4000060 | Primary | OK | BestEffort | 1. | |
| ↵15.0 | | | | | | | |
| cluster-example-2 | 33 MB | 0/4000060 | Standby (async) | OK | BestEffort | 1. | |
| ↵15.0 | | | | | | | |
| cluster-example-3 | 33 MB | 0/4000060 | Standby (async) | OK | BestEffort | 1. | |
| ↵15.0 | | | | | | | |

6.4 Testing failover

As our status checks show, we're running two replicas - if something happens to the primary instance of PostgreSQL, the cluster will fail over to one of them. Let's demonstrate this by killing the primary pod:

```
kubect1 delete pod --wait=false cluster-example-1
__OUTPUT__
pod "cluster-example-1" deleted
```

This simulates a hard shutdown of the server - a scenario where something has gone wrong.

Now if we check the status... ``shell kubect1 cnp status cluster-example **OUTPUT** Cluster Summary Name: cluster-example Namespace: default System ID: 7089127092443762706 PostgreSQL Image: quay.io/enterprisedb/postgresql:14.2 Primary instance: cluster-example-2 Status: Failing over Failing over from cluster-

example-1 to cluster-example-2 Instances: 3 Ready instances: 2 Current Write LSN: 0/5000F58 (Timeline: 2 - WAL File: 00000002000000000000000005)

```
Certificates Status Certificate Name Expiration Date Days Left Until Expiration -
cluster-example-replication 2022-07-20 18:28:12 +0000 UTC 89.99 cluster-example-server
2022-07-20 18:28:12 +0000 UTC 89.99 cluster-example-ca 2022-07-20 18:28:12 +0000 UTC 89.99
```

Continuous Backup status Not configured

Streaming Replication status Not available yet

```
Instances status Name Database Size Current LSN Replication role Status QoS Manager Version -
cluster-example-3 33 MB 0/50000A0 Standby (file based) OK BestEffort
1.15.0 cluster-example-2 33 MB 0/5000F58 Primary OK BestEffort 1.15.0 cluster-example-1 - - - pod not available
BestEffort -
```

...the failover process has begun, with the second pod promoted to primary. Once the failed pod has restarted, it will become a replica of the new primary:

```
` ` ` shell
kubectl cnp status cluster-example
__OUTPUT__
Cluster Summary
Name:          cluster-example
Namespace:     default
System ID:     7089127092443762706
PostgreSQL Image: quay.io/enterprisedb/postgresql:14.2
Primary instance: cluster-example-2
Status:        Cluster in healthy state
Instances:     3
Ready instances: 3
Current Write LSN: 0/5004CD8 (Timeline: 2 - WAL File: 00000002000000000000000005)
```

Certificates Status

| Certificate Name | Expiration Date | Days Left Until Expiration |
|-----------------------------|-------------------------------|----------------------------|
| cluster-example-ca | 2022-07-20 18:28:12 +0000 UTC | 89.99 |
| cluster-example-replication | 2022-07-20 18:28:12 +0000 UTC | 89.99 |
| cluster-example-server | 2022-07-20 18:28:12 +0000 UTC | 89.99 |

Continuous Backup status

Not configured

Streaming Replication status

| Name | Sent LSN | Write LSN | Flush LSN | Replay LSN | Write Lag | Flush Lag |
|-------------------|-----------|-----------|-----------|------------|-----------|-----------|
| cluster-example-1 | 0/5004CD8 | 0/5004CD8 | 0/5004CD8 | 0/5004CD8 | 00:00:00 | 00:00:00 |
| cluster-example-3 | 0/5004CD8 | 0/5004CD8 | 0/5004CD8 | 0/5004CD8 | 00:00:00 | 00:00:00 |

Instances status

| Name | Database Size | Current LSN | Replication role | Status | QoS |
|-------------------|---------------|-------------|------------------|--------|------------|
| cluster-example-1 | 33 MB | 0/5000F58 | Primary | OK | BestEffort |

(continues on next page)

(continued from previous page)

| | | | | | | |
|-------------------|-------|-----------|-----------------|-------|------------|-----|
| --- | ----- | ----- | ----- | ----- | --- | --- |
| ↪----- | | | | | | |
| ↪15.0 | | | | | | |
| cluster-example-3 | 33 MB | 0/5004CD8 | Standby (async) | OK | BestEffort | 1. |
| ↪15.0 | | | | | | |
| cluster-example-2 | 33 MB | 0/5004CD8 | Primary | OK | BestEffort | 1. |
| ↪15.0 | | | | | | |
| cluster-example-1 | 33 MB | 0/5004CD8 | Standby (async) | OK | BestEffort | 1. |
| ↪15.0 | | | | | | |

6.4.1 Further reading

This is all it takes to get a PostgreSQL cluster up and running, but of course there’s a lot more possible - and certainly much more that is prudent before you should ever deploy in a production environment!

- Design goals and possibilities offered by the CloudNativePG Operator: check out the *Supported Architectures* and *Use cases* sections.
- **Configuring a secure and reliable system:** read through the *Security* , *Failure Modes* and *Backup and Recovery* sections.

‘OPERATOR CONFIGURATION’

The operator for EDB Postgres for Kubernetes is installed from a standard deployment manifest and follows the convention over configuration paradigm. While this is fine in most cases, there are some scenarios where you want to change the default behavior, such as:

By default, the operator is installed in the `postgresql-operator-system` namespace as a Kubernetes Deployment called `postgresql-operator-controller-manager`.

Note: In the examples below we assume the default name and namespace for the operator deployment.

The behavior of the operator can be customized through a `ConfigMap` / `Secret` that is located in the same namespace of the operator deployment and with `postgresql-operator-controller-manager-config` as the name.

Important: Any change to the config’s *ConfigMap* / *Secret* will not be automatically detected by the operator, - and as such, it needs to be reloaded (see below). Moreover, changes only apply to the resources created after the configuration is reloaded.

Important: The operator first processes the `ConfigMap` values and then the `Secret`’s, in this order. As a result, if a parameter is defined in both places, the one in the `Secret` will be used.

7.1 Available options

The operator looks for the following environment variables to be defined in the `ConfigMap` / `Secret` :

| Name | Description |
|----------------------------------|--|
| <i>EDB_LICENSE_KEY</i> | default license key (to be used only if the cluster does not define one, and preferably in the <i>Secret</i>) |
| <i>ENABLE_REDWOOD_BY_DEFAULT</i> | Enable the Redwood compatibility by default when using EPAS. |
| <i>INHERITED_ANNOTATIONS</i> | list of annotation names that, when defined in a <i>Cluster</i> metadata, will be inherited by all the generated resources, including pods |
| <i>INHERITED_LABELS</i> | list of label names that, when defined in a <i>Cluster</i> metadata, will be inherited by all the generated resources, including pods |
| <i>PULL_SECRET_NAME</i> | name of an additional pull secret to be defined in the operator’s namespace and to be used to download images |

continues on next page

Table 1 – continued from previous page

| Name | Description |
|--|--|
| <code>ENABLE_AZURE_PVC_UPDATES</code> | Enables to delete Postgres pod if its PVC is stuck in Resizing condition. This feature is mainly for the Azure environment (default <i>false</i>) |
| <code>ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES</code> | when set to <i>true</i> , enables in-place updates of the instance manager after an update of the operator (default <i>false</i>) |
| <code>MONITORING_QUERIES_CONFIGMAP</code> | The name of a ConfigMap in the operator's namespace with a set of default queries to be specified under the key <i>queries</i> to be applied to all created Clusters |
| <code>MONITORING_QUERIES_SECRET</code> | The name of a Secret in the operator's namespace with a set of default queries (to be specified under the key <i>queries</i>) to be applied to all created Clusters |

Values in `INHERITED_ANNOTATIONS` and `INHERITED_LABELS` support path-like wildcards. For example, the value `example.com/*` will match both the value `example.com/one` and `example.com/two`.

When you specify an additional pull secret name using the `PULL_SECRET_NAME` parameter, the operator will use that secret to create a pull secret for every created PostgreSQL cluster. That secret will be named `<cluster-name>-pull`.

The namespace where the operator looks for the `PULL_SECRET_NAME` secret is where you installed the operator. If the operator is not able to find that secret, it will ignore the configuration parameter.

Warning: Previous versions of the operator copied the `PULL_SECRET_NAME` secret inside the namespaces where you deploy the PostgreSQL clusters. From version “1.11.0” the behavior changed to match the previous description. The pull secrets created by the previous versions of the operator are unused.

7.2 Defining an operator config map

The example below customizes the behavior of the operator, by defining a default license key (namely a company key), the label/annotation names to be inherited by the resources created by any `Cluster` object that is deployed at a later time, and by enabling in-place updates for the instance manager.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: postgresql-operator-controller-manager-config
  namespace: postgresql-operator-system
data:
  INHERITED_ANNOTATIONS: categories
  INHERITED_LABELS: environment, workload, app
  ENABLE_INSTANCE_MANAGER_INPLACE_UPDATES: true
```

7.3 Defining an operator secret

The example below customizes the behavior of the operator, by defining a default license key.

```
apiVersion: v1
kind: Secret
metadata:
  name: postgresql-operator-controller-manager-config
  namespace: postgresql-operator-system
```

(continues on next page)

(continued from previous page)

```

type: Opaque
data:
  EDB_LICENSE_KEY: <YOUR_BASE64_ENCODED_EDB_LICENSE_KEY_HERE>

```

7.4 Restarting the operator to reload configs

For the change to be effective, you need to recreate the operator pods to reload the config map. If you have installed the operator on Kubernetes using the manifest you can do that by issuing:

```

kubectl rollout restart deployment \
  -n postgresql-operator-system \
  postgresql-operator-controller-manager

```

Otherwise, If you have installed the operator using OLM, or you are running on Openshift, run the following command specifying the namespace the operator is installed in:

```

kubectl delete pods -n [NAMESPACE_NAME_HERE] \
  -l app.kubernetes.io/name=cloud-native-postgresql

```

Warning: Customizations will be applied only to *Cluster* resources created after the reload of the operator deployment.

Following the above example, if the Cluster definition contains a `categories` annotation and any of the `environment`, `workload`, or `app` labels, these will be inherited by all the resources generated by the deployment.

7.5 PPROF HTTP SERVER

The operator can expose a PPROF HTTP server with the following endpoints on localhost:6060:

- `/debug/pprof/``. Responds to a request for `"/debug/pprof/"` with an HTML page listing ↵
↵ the available profiles
- `/debug/pprof/cmdline``. Responds with the running programs command line, with ↵
↵ arguments separated by NUL bytes.
- `/debug/pprof/profile``. Responds with the pprof-formatted cpu profile. Profiling lasts ↵
↵ for duration specified in seconds GET parameter, or for 30 seconds if not specified.
- `/debug/pprof/symbol``. Looks up the program counters listed in the request, responding ↵
↵ with a table mapping program counters to function names.
- `/debug/pprof/trace``. Responds with the execution trace in binary form. Tracing lasts ↵
↵ for duration specified in seconds GET parameter, or for 1 second if not specified.

To enable the operator you need to edit the operator deployment add the flag `--pprof-server=true`.

You can do this by executing these commands:

```

kubectl edit deployment -n postgresql-operator-system postgresql-operator-controller-
↵ manager

```

Then on the edit page scroll down the container args and add `--pprof-server=true`, example:

```
containers:
- args:
  - controller
  - --enable-leader-election
  - --config-map-name=postgresql-operator-controller-manager-config
  - --secret-name=postgresql-operator-controller-manager-config
  - --log-level=info
  - --pprof-server=true # relevant line
  command:
  - /manager
```

Save the changes, the deployment now will execute a rollout and the new pod will have the PPROF server enabled.

Once the pod is running you can exec inside the container by doing:

```
kubectl exec -ti -n postgresql-operator-system <pod name> -- bash
```

Once inside execute:

```
curl localhost:6060/debug/pprof/
```

EDB AUDIT LOGS

Clusters that are running on EDB Postgres Advanced Server can enable *EDB Audit logs* as follows:

```
apiVersion: postgresql.k8s.enterprisedb.io/v1
kind: Cluster
metadata:
  name: cluster-example
spec:
  instances: 3
  imageName: quay.io/enterprisedb/edb-postgres-advanced:13
  licenseKey: <LICENSE>

  postgresql:
    epas:
      audit: true

  storage:
    size: 1Gi
```

Setting `.spec.postgresql.epas.audit: true` enforces the following parameters:

```
edb_audit = csv
edb_audit_destination = file
edb_audit_directory = /controller/log
edb_audit_filename = edb_audit
edb_audit_rotation_day = none
edb_audit_rotation_seconds = 0
edb_audit_rotation_size = 0
edb_audit_tag =
edb_log_every_bulk_value = false
```

Other parameters can be passed via `.spec.postgresql.parameters` as usual.

The audit CSV logs are parsed and routed to stdout in JSON format, similarly to all the remaining logs:

- ``.logger`` set to ``edb_audit``
- ``.msg`` set to ``record``
- ``.record`` containing the whole parsed record as a JSON object

See the example below:

```
{
  "level": "info",
  "ts": 1624629110.7641866,
  "logger": "edb_audit",
  "msg": "record",
  "record": {
    "log_time": "2021-06-25 13:51:50.763 UTC",
    "user_name": "postgres",
    "database_name": "postgres",
    "process_id": "68",
    "connection_from": "[local]",
    "session_id": "60d5df76.44",
    "session_line_num": "5",
    "process_status": "idle in transaction",
    "session_start_time": "2021-06-25 13:51:50 UTC",
    "virtual_transaction_id": "3/93",
    "transaction_id": "1183",
    "error_severity": "AUDIT",
    "sql_state_code": "00000",
    "message": "statement: GRANT EXECUTE ON function pg_catalog.pg_read_binary_
↪file(text) TO \"streaming_replica\"",
    "detail": "",
    "hint": "",
    "internal_query": "",
    "internal_query_pos": "",
    "context": "",
    "query": "",
    "query_pos": "",
    "location": "",
    "application_name": "",
    "backend_type": "client backend",
    "command_tag": "GRANT",
    "audit_tag": "",
    "type": "grant"
  },
  "logging_pod": "cluster-example-1",
}
```

See EDB Audit file for more details about the records' fields.

See the [CloudNativePG documentation](#) for more information on logging.

‘EDB POSTGRES FOR KUBERNETES PLUGIN’

EDB Postgres for Kubernetes provides a plugin for `kubectl` to manage a cluster in Kubernetes. This plugin is based on [the CloudNativePG Plugin](#), and can work with `oc` in an OpenShift environment.

9.1 Features

The EDB Postgres for Kubernetes Plugin provides the same features as the CloudNativePG plugin, plus:

- It works with `oc` in an OpenShift environment.
- It can manage

9.2 Install

You can install the plugin in your system with:

```
curl -sSfL \
  https://github.com/EnterpriseDB/kubectl-cnp/raw/main/install.sh | \
  sudo sh -s -- -b /usr/local/bin
```

9.2.1 Supported Architectures

EDB Postgres for Kubernetes Plugin is currently build for the following operating system and architectures:

9.3 Use

Once the plugin was installed and deployed, you can start using it like this:

```
kubectl cnp <command> <args...>
```

9.3.1 Status

The `status` command provides an overview of the current status of your cluster, including:

Important: The status information above is taken at different times and at different locations, resulting in slightly inconsistent returned values. For example, the *Current Write LSN* location in the main header, might be different from the *Current LSN* field in the instances status as it is taken at two different time intervals.

```
kubect1 cnp status sandbox
```

```
Cluster in healthy state
Name:                sandbox
Namespace:           default
System ID:            7039966298120953877
PostgreSQL Image:    quay.io/enterprisedb/postgresql:14.2
Primary instance:    sandbox-2
Instances:            3
Ready instances:      3
Current Write LSN:    3AF/EAF6168 (Timeline: 8 - WAL File: 00000008000003AF00000075)

Continuous Backup status
First Point of Recoverability: Not Available
Working WAL archiving:        OK
Last Archived WAL:            00000008000003AE00000079    @    2021-12-14T10:16:29.
    ↳ 340047Z
Last Failed WAL: -

Certificates Status
Certificate Name                Expiration Date                Days Left Until Expiration
-----
cluster-example-ca              2022-05-05 15:02:42 +0000 UTC    87.23
cluster-example-replication     2022-05-05 15:02:42 +0000 UTC    87.23
cluster-example-server          2022-05-05 15:02:42 +0000 UTC    87.23

Streaming Replication status
Name      Sent LSN      Write LSN      Flush LSN      Replay LSN      Write Lag      ↳
    ↳ Flush Lag      Replay Lag      State      Sync State      Sync Priority
-----
    ↳
sandbox-1 3AF/EB0524F0 3AF/EB011760 3AF/EAFED50 3AF/EAFED50 00:00:00.004461 ↳
    ↳ 00:00:00.007901 00:00:00.007901 streaming quorum 1
sandbox-3 3AF/EB0524F0 3AF/EB030B00 3AF/EB030B00 3AF/EB011760 00:00:00.000977 ↳
    ↳ 00:00:00.004194 00:00:00.008252 streaming quorum 1

Instances status
Name      Database Size  Current LSN      Replication role  Status  QoS      Manager ↳
    ↳ Version
-----
    ↳
sandbox-1 302 GB        3AF/E9FFFFE0  Standby (sync)    OK      Guaranteed 1.11.0
sandbox-2 302 GB        3AF/EAF6168  Primary           OK      Guaranteed 1.11.0
sandbox-3 302 GB        3AF/EBAD5D18  Standby (sync)    OK      Guaranteed 1.11.0
```

You can also get a more verbose version of the status by adding `--verbose` or just `-v`

```
kubectl cnp status sandbox --verbose
```

```
Cluster in healthy state
Name:          sandbox
Namespace:     default
System ID:     7039966298120953877
PostgreSQL Image: quay.io/enterprisedb/postgresql:14.2
Primary instance: sandbox-2
Instances:     3
Ready instances: 3
Current Write LSN: 3B1/61DE3158 (Timeline: 8 - WAL File: 000000080000003B1000000030)

PostgreSQL Configuration
archive_command = /controller/manager wal-archive --log-destination /controller/log/
↳ postgres.json %p
archive_mode = on
archive_timeout = 5min
checkpoint_completion_target = 0.9
checkpoint_timeout = 900s
cluster_name = sandbox
dynamic_shared_memory_type = sysv
full_page_writes = on
hot_standby = true
jit = on
listen_addresses = *
log_autovacuum_min_duration = 1s
log_checkpoints = on
log_destination = csvlog
log_directory = /controller/log
log_filename = postgres
log_lock_waits = on
log_min_duration_statement = 1000
log_rotation_age = 0
log_rotation_size = 0
log_statement = ddl
log_temp_files = 1024
log_truncate_on_rotation = false
logging_collector = on
maintenance_work_mem = 2GB
max_connections = 1000
max_parallel_workers = 32
max_replication_slots = 32
max_wal_size = 15GB
max_worker_processes = 32
pg_stat_statements.max = 10000
pg_stat_statements.track = all
port = 5432
shared_buffers = 16GB
shared_memory_type = sysv
shared_preload_libraries = pg_stat_statements
ssl = on
```

(continues on next page)

(continued from previous page)

```

ssl_ca_file = /controller/certificates/client-ca.crt
ssl_cert_file = /controller/certificates/server.crt
ssl_key_file = /controller/certificates/server.key
synchronous_standby_names = ANY 1 ("sandbox-1","sandbox-3")
unix_socket_directories = /controller/run
wal_keep_size = 512MB
wal_level = logical
wal_log_hints = on
cnpg.config_sha256 = 3cfa683e23fe513afaae7c97b50ce0628e0cc634bca8b096517538a9a4428efc

```

PostgreSQL HBA Rules

```

# Grant local access
local all all peer map=local

# Require client certificate authentication for the streaming_replica user
hostssl postgres streaming_replica all cert
hostssl replication streaming_replica all cert
hostssl all cnpg_pooler_pgouncer all cert

# Otherwise use the default authentication method
host all all all scram-sha-256

```

Continuous Backup status

```

First Point of Recoverability: Not Available
Working WAL archiving: OK
Last Archived WAL: 0000000080000003B000000001D @ 2021-12-14T10:20:42.
↳ 272815Z
Last Failed WAL: -

```

Streaming Replication status

| Name | Sent LSN | Write LSN | Flush LSN | Replay LSN | Write Lag | |
|-------------------|-----------------|--------------|--------------|---------------|-----------------|---|
| ↳ Flush Lag | Replay Lag | State | Sync State | Sync Priority | | |
| sandbox-1 | 3B1/61E26448 | 3B1/61DF82F0 | 3B1/61DF82F0 | 3B1/61DF82F0 | 00:00:00.000333 | ↳ |
| ↳ 00:00:00.000333 | 00:00:00.005484 | streaming | quorum | 1 | | |
| sandbox-3 | 3B1/61E26448 | 3B1/61E26448 | 3B1/61DF82F0 | 3B1/61DF82F0 | 00:00:00.000756 | ↳ |
| ↳ 00:00:00.000756 | 00:00:00.000756 | streaming | quorum | 1 | | |

Instances status

| Name | Database Size | Current LSN | Replication role | Status | QoS | Manager |
|-----------|---------------|--------------|------------------|--------|------------|---------|
| ↳ Version | | | | | | |
| sandbox-1 | | 3B1/610204B8 | Standby (sync) | OK | Guaranteed | 1.11.0 |
| sandbox-2 | | 3B1/61DE3158 | Primary | OK | Guaranteed | 1.11.0 |
| sandbox-3 | | 3B1/62618470 | Standby (sync) | OK | Guaranteed | 1.11.0 |

The command also supports output in yaml and json format.

9.3.2 Promote

The meaning of this command is to promote a pod in the cluster to primary, so you can start with maintenance work or test a switch-over situation in your cluster

```
kubectl cnp promote cluster-example cluster-example-2
```

Or you can use the instance node number to promote

```
kubectl cnp promote cluster-example 2
```

9.3.3 Certificates

Clusters created using the EDB Postgres for Kubernetes operator work with a CA to sign a TLS authentication certificate.

To get a certificate, you need to provide a name for the secret to store the credentials, the cluster name, and a user for this certificate

```
kubectl cnp certificate cluster-cert --cnp-cluster cluster-example --cnp-user appuser
```

After the secret is created, you can get it using `kubectl`

```
kubectl get secret cluster-cert
```

And the content of the same in plain text using the following commands:

```
kubectl get secret cluster-cert -o json | jq -r .data | map(@base64d) | .[]
```

9.3.4 Restart

The `kubectl cnp restart` command can be used in two cases:

```
# this command will restart a whole cluster in a rollout fashion
kubectl cnp restart [clusterName]

# this command will restart a single instance, according to the policy above
kubectl cnp restart [clusterName] [pod]
```

If the in-place restart is requested but the change cannot be applied without a switchover, the switchover will take precedence over the in-place restart. A common case for this will be a minor upgrade of PostgreSQL image.

Note: If you want ConfigMaps and Secrets to be **automatically** reloaded by instances, you can add a label with key `k8s.enterprisedb.io/reload` to it.

9.3.5 Reload

The `kubectl cnp reload` command requests the operator to trigger a reconciliation loop for a certain cluster. This is useful to apply configuration changes to cluster dependent objects, such as ConfigMaps containing custom monitoring queries.

The following command will reload all configurations for a given cluster:

```
kubectl cnp reload [cluster_name]
```

9.3.6 Maintenance

The `kubectl cnp maintenance` command helps to modify one or more clusters across namespaces and set the maintenance window values, it will change the following fields:

- `.spec.nodeMaintenanceWindow.inProgress`
- `.spec.nodeMaintenanceWindow.reusePVC`

Accepts as argument `set` and `unset` using this to set the `inProgress` to `true` in case `set` and to `false` in case of `unset`.

By default, `reusePVC` is always set to `false` unless the `--reusePVC` flag is passed.

The plugin will ask for a confirmation with a list of the cluster to modify and their new values, if this is accepted this action will be applied to all the cluster in the list.

If you want to set in maintenance all the PostgreSQL in your Kubernetes cluster, just need to write the following command:

```
kubectl cnp maintenance set --all-namespaces
```

And you'll have the list of all the cluster to update

```
The following are the new values for the clusters
Namespace Cluster Name      Maintenance reusePVC
- - - - -
default    cluster-example true         false
default    pg-backup      true         false
test       cluster-example true         false
Do you want to proceed? [y/n]: y
```

9.3.7 Report

The `kubectl cnp report` command bundles various pieces of information into a ZIP file. It aims to provide the needed context to debug problems with clusters in production.

It has two sub-commands: `operator` and `cluster`.

9.3.8 report Operator

The `operator` sub-command requests the operator to provide information regarding the operator deployment, configuration and events.

Important: All confidential information in Secrets and ConfigMaps is REDACTED. The Data map will show the keys but the values will be empty. The flag `-S / --stopRedaction` will defeat the redaction and show the values. Use only at your own risk, this will share private data.

Note: By default, operator logs are not collected, but you can enable operator log collection with the `--logs` flag

- `**deploymentinformation**` : the operator Deployment and operator Pod
- `**configuration**` : the Secrets and ConfigMaps in the operator namespace
- `**events**` : the Events in the operator namespace
- `**webhookconfiguration**` : the mutating and validating webhook configurations
- `**webhookservice**` : the webhook service
- `**logs**` : logs for the operator Pod (optional, off by default) in JSON-lines format

The command will generate a ZIP file containing various manifest in YAML format (by default, but settable to JSON with the `-o` flag). Use the `-f` flag to name a result file explicitly. If the `-f` flag is not used, a default time-stamped filename is created for the zip file.

```
kubectl cnp report operator
```

results in

```
Successfully written report to "report_operator_<TIMESTAMP>.zip" (format: "yaml")
```

With the `-f` flag set:

```
kubectl cnp report operator -f reportRedacted.zip
```

Unzipping the file will produce a time-stamped top-level folder to keep the directory tidy:

```
unzip reportRedacted.zip
```

will result in:

```
Archive:  reportRedacted.zip
  creating: report_operator_<TIMESTAMP>/
  creating: report_operator_<TIMESTAMP>/manifests/
 inflating: report_operator_<TIMESTAMP>/manifests/deployment.yaml
 inflating: report_operator_<TIMESTAMP>/manifests/operator-pod.yaml
 inflating: report_operator_<TIMESTAMP>/manifests/events.yaml
 inflating: report_operator_<TIMESTAMP>/manifests/validating-webhook-configuration.yaml
```

(continues on next page)

(continued from previous page)

```

inflating: report_operator_<TIMESTAMP>/manifests/mutating-webhook-configuration.yaml
inflating: report_operator_<TIMESTAMP>/manifests/webhook-service.yaml
inflating: report_operator_<TIMESTAMP>/manifests/postgresql-operator-ca-secret.yaml
inflating: report_operator_<TIMESTAMP>/manifests/postgresql-operator-webhook-cert.yaml

```

You can verify that the confidential information is REDACTED:

```

cd report_operator_<TIMESTAMP>/manifests/
head postgresql-operator-ca-secret.yaml

```

```

data:
  ca.crt: ""
  ca.key: ""
metadata:
  creationTimestamp: "2022-03-22T10:42:28Z"
  managedFields:
    - apiVersion: v1
      fieldType: FieldsV1
      fieldsV1:

```

With the `-S ( --stopRedaction )` option activated, secrets are shown:

```
kubectcl cnp report operator -f reportNonRedacted.zip -S
```

You'll get a reminder that you're about to view confidential information:

```

WARNING: secret Redaction is OFF. Use it with caution
Successfully written report to "reportNonRedacted.zip" (format: "yaml")

```

```

unzip reportNonRedacted.zip
head postgresql-operator-ca-secret.yaml

```

```

data:
  ca.crt: LS0tLS1CRUdJTiBD...
  ca.key: LS0tLS1CRUdJTiBF...
metadata:
  creationTimestamp: "2022-03-22T10:42:28Z"
  managedFields:
    - apiVersion: v1
      fieldType: FieldsV1

```

9.3.9 report Cluster

The cluster sub-command gathers the following:

- ****clusterresources**** : the cluster information, same as ``kubectcl get cluster -o_↵↵yaml``
- ****clusterpods**** : pods **in** the cluster namespace matching the cluster name

- ****clusterjobs**** : jobs, **if any, in** the cluster namespace matching the cluster name
- ****events**** : events **in** the cluster namespace
- ****podlogs**** : logs **for** the cluster Pods (optional, off by default) **in** JSON-lines,
↪ **format**
- ****joblogs**** : logs **for** the Pods created by jobs (optional, off by default) **in** JSON-
↪ lines **format**

The `cluster` sub-command accepts the `-f` and `-o` flags, as the `operator` does. If the `-f` flag is not used, a default timestamped report name will be used. Note that the cluster information does not contain configuration Secrets / ConfigMaps, so the `-S` is disabled.

Note: By default, cluster logs are not collected, but you can enable cluster log collection with the `-logs` flag

Usage:

```
kubect1-cnp report cluster [clusterName] -f <filename.zip> [flags]
```

Note that, unlike the `operator` sub-command, for the `cluster` sub-command you need to provide the cluster name, and very likely the namespace, unless the cluster is in the default one.

```
kubect1 cnp report cluster cluster-example-full -f report.zip -n example2
```

and then:

```
unzip report.zip
```

```
Archive:  report.zip
  creating: report_cluster_<TIMESTAMP>/
  creating: report_cluster_<TIMESTAMP>/manifests/
 inflating: report_cluster_<TIMESTAMP>/manifests/cluster.yaml
 inflating: report_cluster_<TIMESTAMP>/manifests/cluster-pods.yaml
 inflating: report_cluster_<TIMESTAMP>/manifests/cluster-jobs.yaml
 inflating: report_cluster_<TIMESTAMP>/manifests/events.yaml
```

Remember that you can use the `--logs` flag to add the pod and job logs to the ZIP.

```
kubect1 cnp report cluster cluster-example-full -n example2 --logs
```

will result in:

```
Successfully written report to "report_cluster_<TIMESTAMP>.zip" (format: "yaml")
```

```
unzip report_cluster_<TIMESTAMP>.zip
```

```
Archive:  report_cluster_<TIMESTAMP>.zip
  creating: report_cluster_<TIMESTAMP>/
  creating: report_cluster_<TIMESTAMP>/manifests/
 inflating: report_cluster_<TIMESTAMP>/manifests/cluster.yaml
```

(continues on next page)

(continued from previous page)

```
inflating: report_cluster_<TIMESTAMP>/manifests/cluster-pods.yaml
inflating: report_cluster_<TIMESTAMP>/manifests/cluster-jobs.yaml
inflating: report_cluster_<TIMESTAMP>/manifests/events.yaml
  creating: report_cluster_<TIMESTAMP>/logs/
inflating: report_cluster_<TIMESTAMP>/logs/cluster-example-full-1.jsonl
  creating: report_cluster_<TIMESTAMP>/job-logs/
inflating: report_cluster_<TIMESTAMP>/job-logs/cluster-example-full-1-initdb-qnnvw.
↪ jsonl
inflating: report_cluster_<TIMESTAMP>/job-logs/cluster-example-full-2-join-tvj8r.jsonl
```

‘RED HAT OPENSIFT’

EDB Postgres for Kubernetes is certified to run on [Red Hat OpenShift Container Platform \(OCP\) version 4.x](#) and is available directly from the [Red Hat Catalog](#) .

The goal of this section is to help you decide the best installation method for EDB Postgres for Kubernetes based on your organizations’ security and access control policies.

EDB Postgres for Kubernetes can be installed and managed via:

- OpenShift web console
- OpenShift command-line interface (CLI) called `oc` , for full control

EDB Postgres for Kubernetes supports all available install modes defined by OpenShift:

- cluster-wide, in all namespaces
- local, in a single namespace
- local, watching multiple namespaces (only available using `oc`)

Note: A project is a Kubernetes namespace with additional annotations, and is the central vehicle by which access to resources for regular users is managed.

In most cases, the default cluster-wide installation of EDB Postgres for Kubernetes is the recommended one, with either central management of PostgreSQL clusters or delegated management (limited to specific users/projects according to RBAC definitions - see *Important OpenShift concepts* and

Users and Permissions below).

Important: Both the installation and upgrade processes require access to an OpenShift Container Platform cluster using an account with *cluster-admin* permissions. From [Default cluster roles](#) , a *cluster-admin* is “a super-user that can perform any action in any project. When bound to a user with a local binding, they have full control over quota and every action on every resource in the project”.

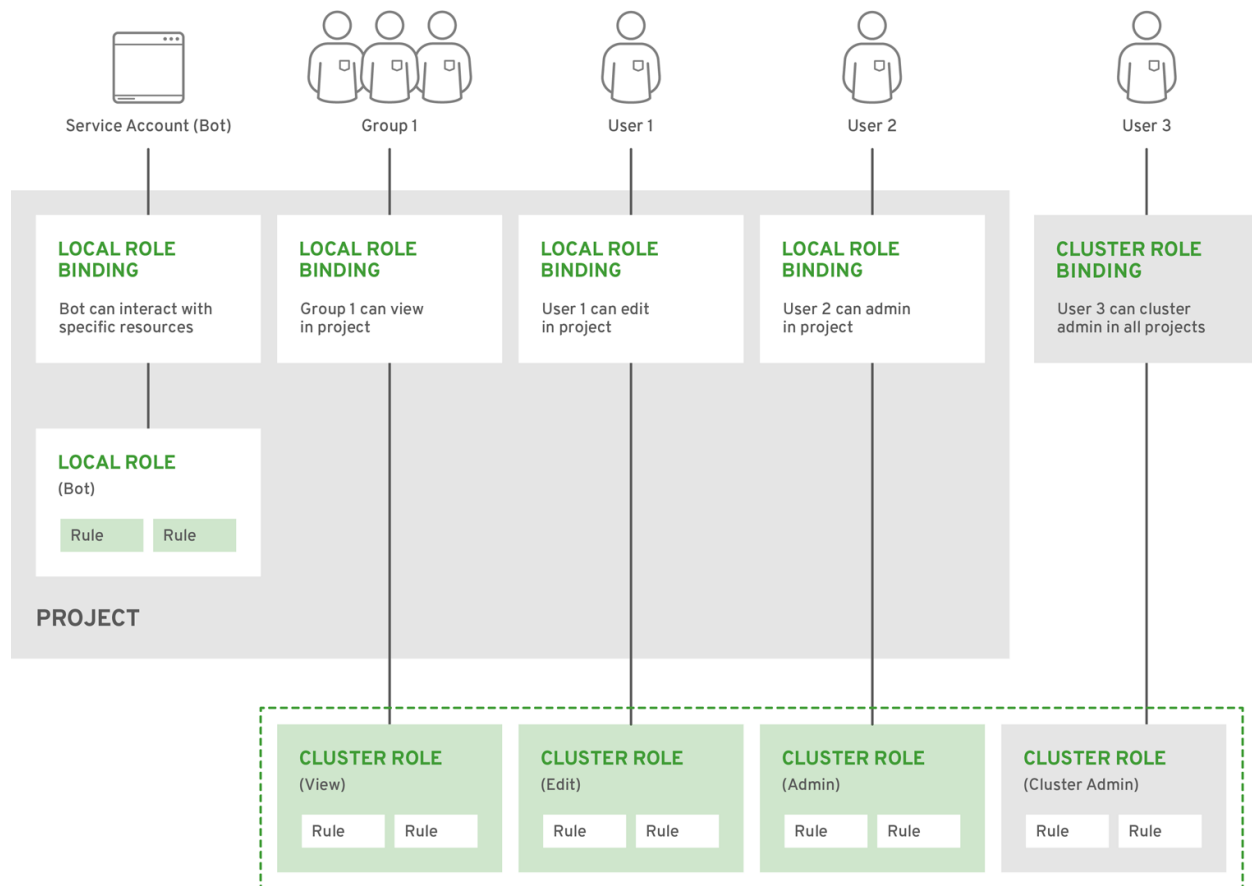
10.1 Important OpenShift concepts

To understand how the EDB Postgres for Kubernetes operator fits in an OpenShift environment, you must familiarize yourself with the following Kubernetes-related topics:

- Operators
- Authentication
- Authorization via Role-based Access Control (RBAC)
- Service Accounts and Users
- Rules, Roles and Bindings
- Cluster RBAC vs local RBAC through projects

This is especially true in case you are not comfortable with the elevated permissions required by the default cluster-wide installation of the operator.

We have also selected the diagram below from the OpenShift documentation, as it clearly illustrates the relationships between cluster roles, local roles, cluster role bindings, local role bindings, users, groups and service accounts.



OPENSIFT_415489_0218

The *Predefined RBAC objects* below contains important information about how EDB Postgres for Kubernetes adheres to Kubernetes and OpenShift RBAC implementation, covering default installed cluster roles, roles, service accounts.

If you are familiar with the above concepts, you can proceed directly to the selected installation method. Otherwise, we recommend that you read the following resources taken from the OpenShift documentation and the Red Hat blog:

10.1.1 Cluster Service Version (CSV)

Technically, the operator is designed to run in OpenShift via the OperatorLifecycle Manager (OLM), according to the Cluster Service Version (CSV) defined by EDB.

The CSV is a YAML manifest that defines not only the user interfaces (available through the web dashboard), but also the RBAC rules required by the operator and the custom resources defined and owned by the operator (such as the Cluster one, for example). The CSV defines also the available `installModes` for the operator, namely: `AllNamespaces` (cluster-wide), `SingleNamespace` (single project), `MultiNamespace` (multi-project), and `OwnNamespace`.

10.1.2 Limitations for multi-tenant management

Red Hat OpenShift Container Platform provides limited support for simultaneously installing different variations of an operator on a single cluster. Like any other operator, EDB Postgres for Kubernetes becomes an extension of the control plane. As the control plane is shared among all tenants (projects) of an OpenShift cluster, operators too become shared resources in a multi-tenant environment.

Operator Lifecycle Manager (OLM) can install operators multiple times in different namespaces, with one important limitation: they all need to share the same API version of the operator.

For more information, refer to [Operator groups](#) in the OpenShift documentation.

10.2 Installation via web console

The EDB Postgres for Kubernetes operator can be found in the Red Hat OperatorHub directly from your OpenShift dashboard.

1. Navigate in the web console to the **Operators** -> **OperatorHub** page:
2. Scroll in the **Database** section or type a keyword into the **Filter by keyword** box (in this case, "PostgreSQL") to find the EDB Postgres for Kubernetes Operator, then select it:
3. Read the information about the Operator and select **Install**.
4. The following **Operator** installation page expects you to choose:
 - the installation mode: *Cluster-wide installation* or single namespace installation - the update channel: currently only stable is available - the approval strategy, following the availability on the market place of a new release of the operator, certified by Red Hat: - **Automatic**: OLM automatically upgrades the running operator with the new version - **Manual**: OpenShift waits for human intervention, by requiring an approval in the **Installed Operators** section

Important: The process of the operator upgrade is described in the 'Installation and upgrades'.

Important: It is possible to install the operator in a single project (technically speaking: *OwnNamespace* install mode) multiple times in the same cluster. There will be an operator installation in every namespace, with different upgrade policies as long as the API is the same (see *Limitations for multi-tenant management*).

Choosing cluster-wide vs local installation of the operator is a critical turning point. Trying to install the operator globally with an existing local installation is blocked, by throwing the error below. If you want to proceed you need to remove every local installation of the operator first.

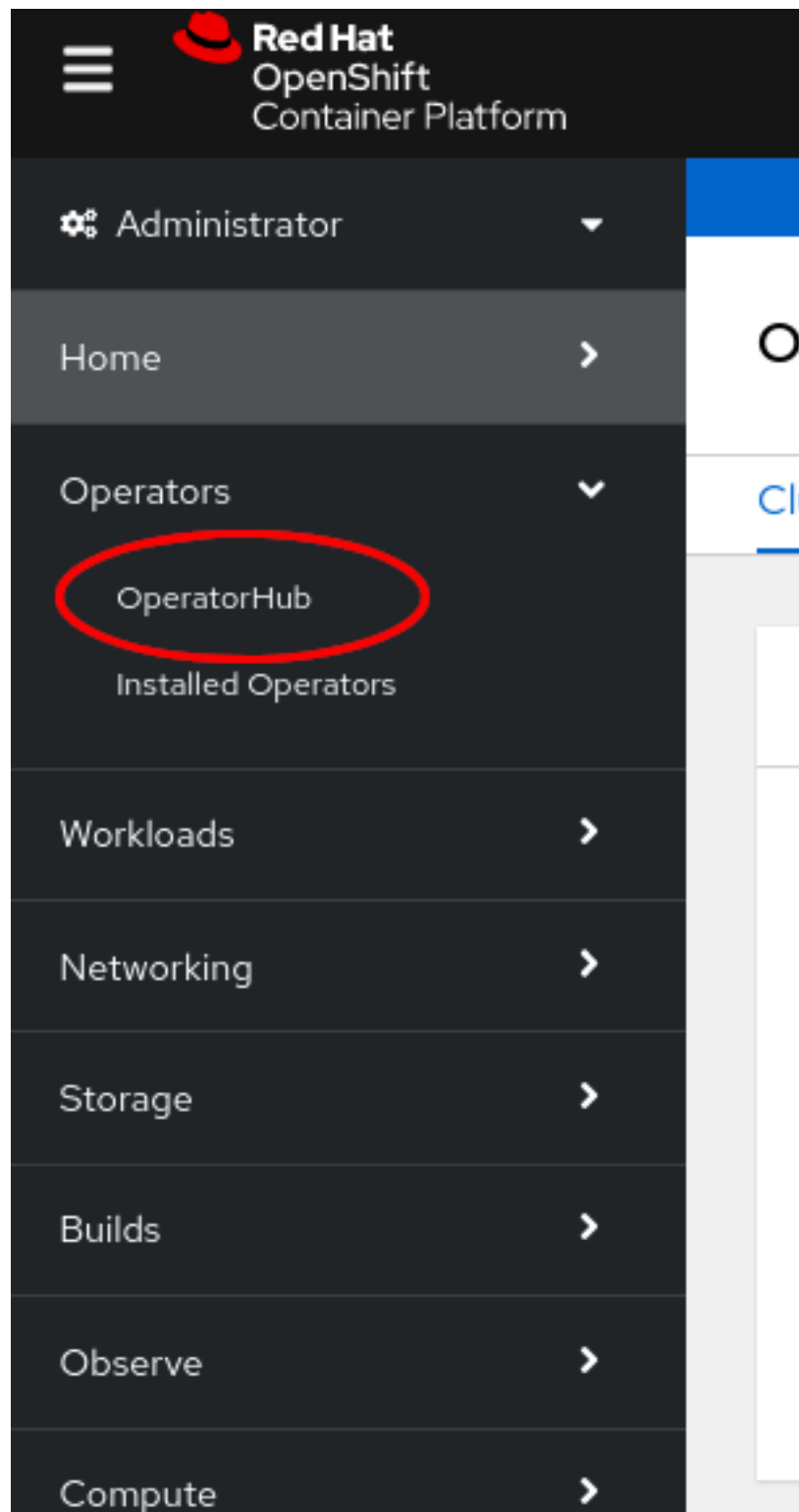


Fig. 1: Menu OperatorHub

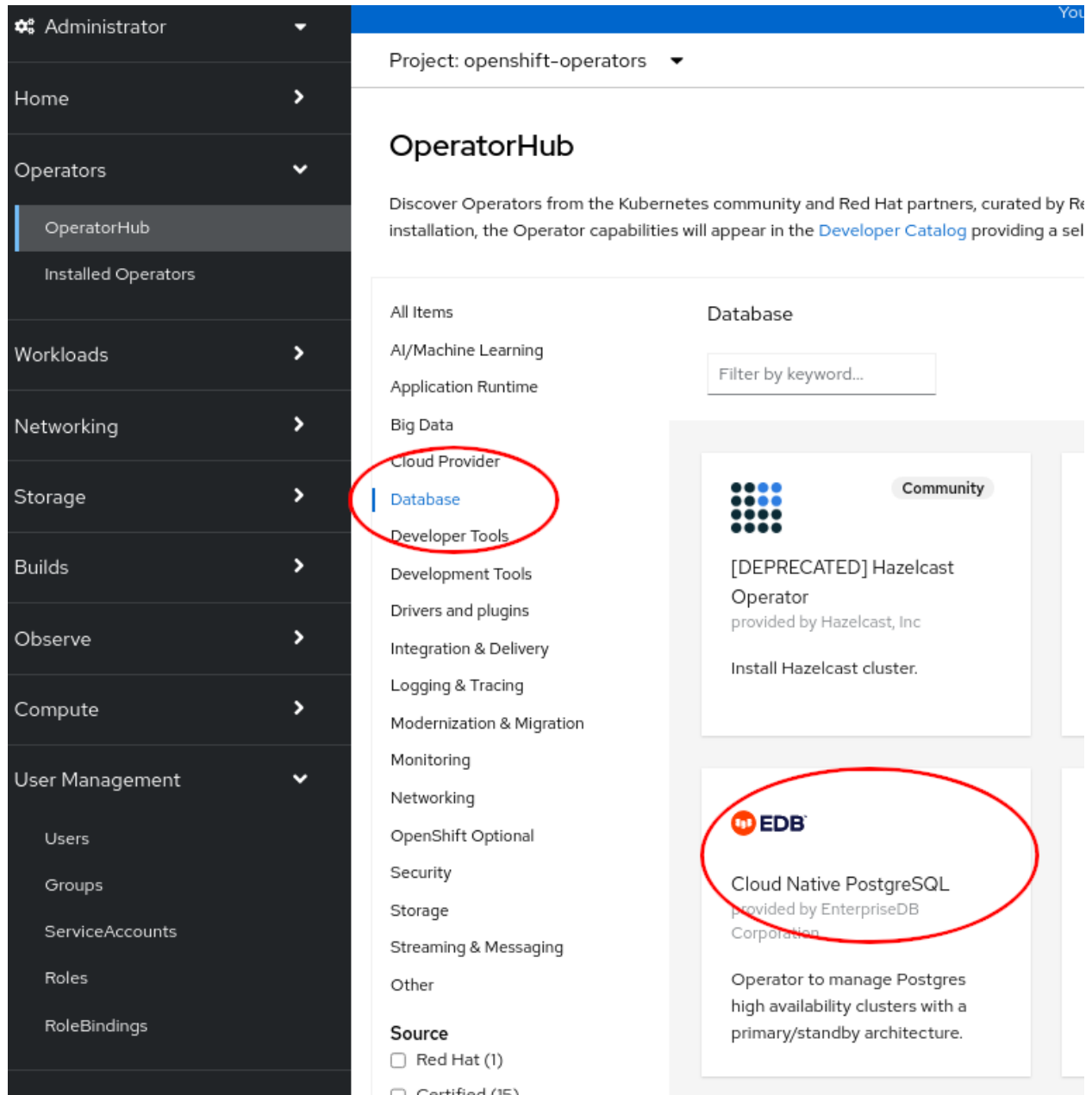


Fig. 2: Install OperatorHub

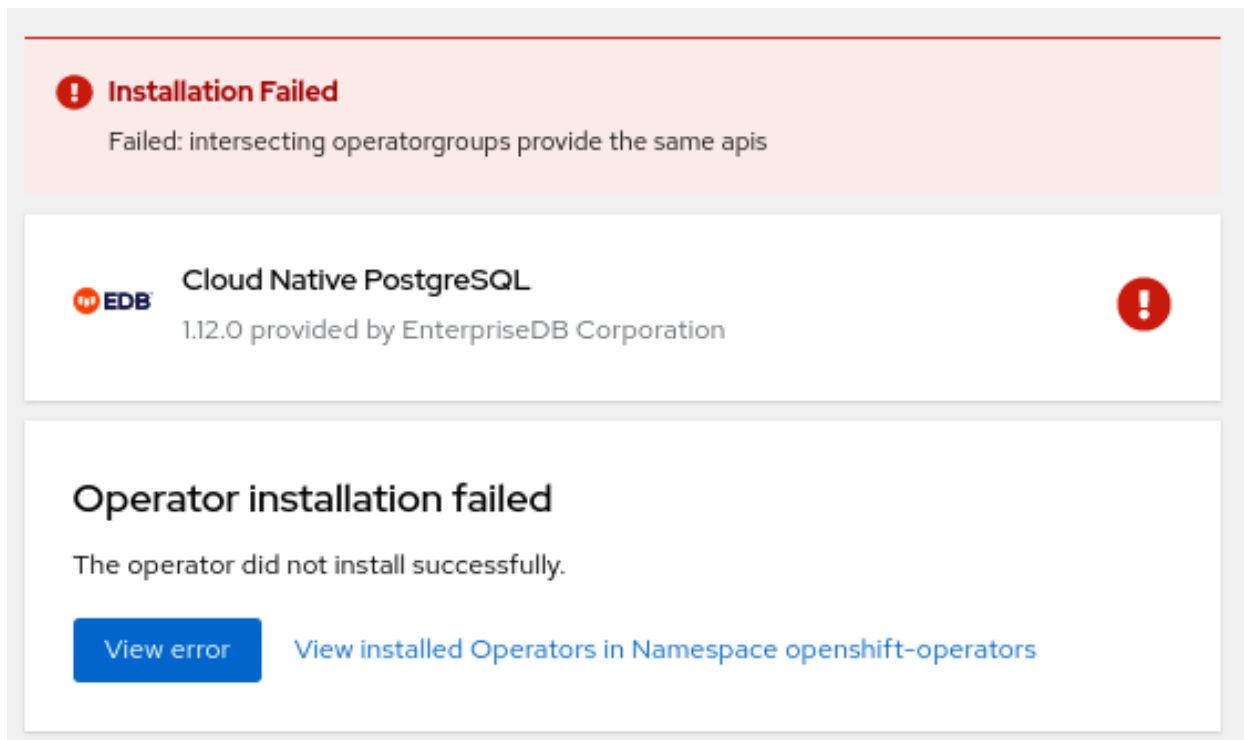


Fig. 3: Install failing

10.2.1 Cluster-wide installation

With cluster-wide installation, you are asking OpenShift to install the Operator in the default `openshift-operators` namespace and to make it available to all the projects in the cluster. This is the default and normally recommended approach to install EDB Postgres for Kubernetes.

Warning: This doesn't mean that every user in the OpenShift cluster can use the Cloud Native PostgreSQL Operator, deploy a *Cluster* object or even see the *Cluster* objects that are running in their own namespaces. There are some special roles that users must have in the namespace in order to interact with EDB Postgres for Kubernetes's managed custom resources - primarily the *Cluster* one. Please refer to the [Users and Permissions](#) for details.

From the web console, select **All namespaces on the cluster (default)** as Installation mode :

As a result, the operator will be visible in every namespaces. Otherwise, as with any other OpenShift operator, check the logs in any pods in the `openshift-operators` project on the **Workloads** → **Pods** page that are reporting issues to troubleshoot further.

OperatorHub > Operator Installation

Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

Update channel *

☒ stable

Installation mode *

☒ All namespaces on the cluster (default)

Operator will be available in all Namespaces.

☐ A specific namespace on the cluster

Operator will be available in a single Namespace only.

Installed Namespace *

Update approval *

☒ Automatic

☐ Manual

Install

Cancel

EDB Cloud Native PostgreSQL
provided by EnterpriseDB Corporation

Provided APIs

Backups

PostgreSQL backup manager

Cluster

PostgreSQL cluster manager

Pooler

PgBouncer Pooler Manager

Scheduled Backups

PostgreSQL backups schedule

Fig. 4: Install all namespaces

10.2.2 Single project installation

With single project installation, you are asking OpenShift to install the Operator in a given namespace, and to make it available to that project only.

Warning: This doesn't mean that every user in the namespace can use the Cloud Native PostgreSQL Operator, deploy a *Cluster* object or even see the *Cluster* objects that are running in the namespace. Similarly to the cluster-wide installation mode, There are some special roles that users must have in the namespace in order to interact with EDB Postgres for Kubernetes's managed custom resources - primarily the *Cluster* one. Please refer to the [Users and Permissions](#) for details.

From the web console, select **A specific namespace on the cluster** as Installation mode, then pick the target namespace (in our example `proj-dev`):

As a result, the operator will be visible in the selected namespace only. You can verify this from the **Installed operators** page:

In case of a problem, from the **Workloads** → **Pods** page check the logs in any pods in the selected installation namespace that are reporting issues to troubleshoot further.

This installation process can be repeated in multiple namespaces in the same OpenShift cluster, enabling independent installations of the operator in different projects. In this case, make sure you read

Limitations for multi-tenant management.

OperatorHub > Operator Installation

Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to date. The strategy determines either manual or automatic updates.

Update channel *

☒ stable

Installation mode *

☐ All namespaces on the cluster (default)

Operator will be available in all namespaces.

☒ A specific namespace on the cluster

Operator will be available in a single Namespace only.

Installed Namespace *

☒ proj-dev

Update approval *

☒ Automatic

☐ Manual

Install

Cancel

EDB Cloud Native PostgreSQL
provided by EnterpriseDB Corporation

Provided APIs

Backups

PostgreSQL backup manager

Cluster

PostgreSQL cluster manager

Pooler

PgBouncer Pooler Manager

Scheduled Backups

PostgreSQL backups schedule

Fig. 5: Install one namespace

| | | | | |
|---|--------------------|---------------------------|---------------|---|
| You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in. | | | | |
| Project: proj-dev | | | | |
| Installed Operators | | | | |
| Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the Understanding Operators documentation . Or create an Operator and ClusterServiceVersion using the Operator SDK . | | | | |
| Name Search by name... | | | | |
| Name | Managed Namespaces | Status | Last updated | Provided APIs |
| EDB Cloud Native PostgreSQL
1.12.0 provided by EnterpriseDB Corporation | proj-dev | ✓ Succeeded
Up to date | 4 minutes ago | Backups
Cluster
Pooler
Scheduled Backups |

Fig. 6: Install one namespace list

10.3 Installation via the oc CLI

Important: Please refer to the [Installing the OpenShift CLI \(`oc` \)](#) for information on how to install the `oc` command-line interface.

Instead of using the OpenShift Container Platform web console, you can install the EDB Postgres for Kubernetes Operator from the OperatorHub and create a subscription using the `oc` command-line interface. Through the `oc` CLI you can install the operator in all namespaces, a single namespace or multiple namespaces.

Warning: Multiple namespace installation is currently supported by OpenShift. However, [definition of multiple target namespaces for an operator may be removed in future versions of OpenShift](#) .

This section primarily covers the installation of the operator in multiple projects with a simple example, by creating an `OperatorGroup` and a `Subscription` objects.

Note: In our example, we will install the operator in the `my-operators` namespace and make it only available in the `web-staging` , `web-prod` , `bi-staging` , and `bi-prod` namespaces. Feel free to change the names of the projects as you like or add/remove some namespaces.

1. Check that the `cloud-native-postgresql` operator is available from the OperatorHub:

```
oc get packagemanifests -n openshift-marketplace cloud-native-postgresql
```

2. Inspect the operator to verify the installation modes (`MultiNamespace` in particular) and the available channels:

```
oc describe packagemanifests -n openshift-marketplace cloud-native-postgresql
```

3. Create an `OperatorGroup` object in the `my-operators` namespace so that it targets the `web-staging` , `web-prod` , `bi-staging` , and `bi-prod` namespaces:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cloud-native-postgresql
  namespace: my-operators
spec:
  targetNamespaces:
    - web-staging
    - web-prod
    - bi-staging
    - bi-prod
```

Important: Alternatively, you can list namespaces using a label selector, as explained in [Target namespace selection](#) .

4. Create a `Subscription` object in the `my-operators` namespace to subscribe to the `stable` channel of the `cloud-native-postgresql` operator that is available in the `certified-operators` source of the `openshift-marketplace` (as previously located in steps 1 and 2):

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cloud-native-postgresql
  namespace: my-operators
spec:
  channel: stable
  name: cloud-native-postgresql
  source: certified-operators
  sourceNamespace: openshift-marketplace

```

5. Use `oc apply -f` with the above YAML file definitions for the OperatorGroup and Subscription objects.

The method described in this section can be very powerful in conjunction with proper RoleBinding objects, as it enables mapping EDB Postgres for Kubernetes' predefined ClusterRoles to specific users in selected namespaces.

Note: The above instructions can also be used for single project binding. The only difference is the number of specified target namespaces (one) and, possibly, the namespace of the operator group (ideally, the same as the target namespace).

The result of the above operation can also be verified from the webconsole, as shown in the image below.

Project: my-operators ▼

Installed Operators

Installed Operators
[Understanding Operators](#)

Name ▼ Search

Managed Namespaces ✕

- NS bi-prod
- NS bi-staging
- NS web-prod
- NS web-staging

4 Namespaces

| Name | Status | Provided APIs |
|--|---------------------------|---|
|  Cloud Native PostgreSQL
provided by EnterpriseDB Corporation | ✓ Succeeded
Up to date | Backups
Cluster
Pooler
Scheduled Backups |

Namespace. For more information, see the ClusterServiceVersion using the [Operator SDK](#).

Fig. 7: Multi namespace installation displayed in the web console

10.3.1 Cluster-wide installation with oc

If you prefer, you can also use `oc` to install the operator globally, by taking advantage of the default `OperatorGroup` called `global-operators` in the `openshift-operators` namespace, and create a new `Subscription` object for the `cloud-native-postgresql` operator in the same namespace:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cloud-native-postgresql
  namespace: openshift-operators
spec:
  channel: stable
  name: cloud-native-postgresql
  source: certified-operators
  sourceNamespace: openshift-marketplace
```

Once you run `oc apply -f` with the above YAML file, the operator will be available in all namespaces.

10.3.2 Installing the OpenShift CLI (oc)

The `oc` command represents the OpenShift command-line interface (CLI). It is highly recommended to install it on your system. Below you find a basic set of instructions to install `oc` from your OpenShift dashboard.

First, select the question mark at the top right corner of the dashboard:

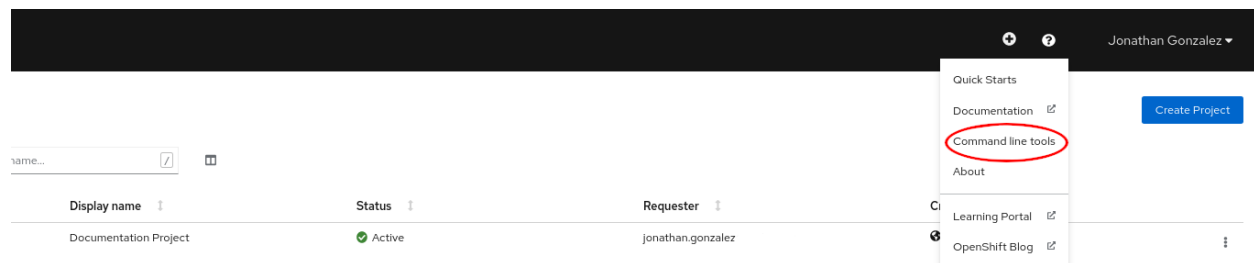


Fig. 8: Command Line Tools menu

Then follow the instructions you are given, by downloading the binary that suits your needs in terms of operating system and architecture:

10.4 Predefined RBAC objects

EDB Postgres for Kubernetes comes with a predefined set of resources that play an important role when it comes to RBAC policy configuration.

Command Line Tools

[Copy Login Command](#)

oc - OpenShift Command Line Interface (CLI)

With the OpenShift command line interface, you can create applications and manage OpenShift projects from a terminal.

The oc binary offers the same capabilities as the kubectl binary, but it is further extended to natively support OpenShift Container Platform features.

- [Download oc for Linux for x86_64](#)
- [Download oc for Mac for x86_64](#)
- [Download oc for Windows for x86_64](#)
- [Download oc for Linux for ARM 64](#)
- [Download oc for Linux for IBM Power, little endian](#)
- [Download oc for Linux for IBM Z](#)
- [LICENSE](#)

helm - Helm 3 CLI

Helm 3 is a package manager for Kubernetes applications which enables defining, installing, and upgrading applications packaged as Helm Charts.

[Download Helm](#)

odo - Developer-focused CLI for OpenShift

odo is a fast, iterative, and straightforward CLI tool for developers who write, build, and deploy applications on OpenShift.

odo abstracts away complex Kubernetes and OpenShift concepts, thus allowing developers to focus on what is most important to them: code.

[Download odo](#)

Fig. 9: Command Line Tools

10.4.1 Custom Resource Definitions (CRD)

The EDB Postgres for Kubernetes operator owns the following custom resourcedefinitions (CRD):

- ``Backup``

- ``Cluster``

- ``Pooler``

- ``ScheduledBackup``

You can verify this by running:

```
oc get customresourcedefinitions.apiextensions.k8s.io | grep postgresql
```

which returns something similar to:

| | |
|---|----------------------|
| backups.postgresql.k8s.enterprisedb.io | 20YY-MM-DDTHH:MM:SSZ |
| clusters.postgresql.k8s.enterprisedb.io | 20YY-MM-DDTHH:MM:SSZ |
| poolers.postgresql.k8s.enterprisedb.io | 20YY-MM-DDTHH:MM:SSZ |
| scheduledbackups.postgresql.k8s.enterprisedb.io | 20YY-MM-DDTHH:MM:SSZ |

10.4.2 Service accounts

The namespace where the operator has been installed (by default `openshift-operators`) contains the following pre-defined service accounts: `builder`, `default`, `deployer`, and most importantly `postgresql-operator-manager` (managed by the CSV).

Important: Service accounts in Kubernetes are namespaced resources. Unless explicitly authorized, a service account cannot be accessed outside the defined namespace.

You can verify this by running:

```
oc get serviceaccounts -n openshift-operators
```

which returns something similar to:

| NAME | SECRETS | AGE |
|-----------------------------|---------|-----|
| builder | 2 | ... |
| default | 2 | ... |
| deployer | 2 | ... |
| postgresql-operator-manager | 2 | ... |

The `default` service account is automatically created by Kubernetes and present in every namespace. The `builder` and `deployer` service accounts are automatically created by OpenShift (see [Default project service accounts and roles](#)).

The `postgresql-operator-manager` service account is the one used by the CloudNative PostgreSQL operator to work as part of the Kubernetes/OpenShift controlplane in managing PostgreSQL clusters.

Important: Do not delete the *postgresql-operator-manager* ServiceAccount as it can stop the operator from working.

10.4.3 Cluster roles

The Operator Lifecycle Manager (OLM) automatically creates a set of clusterrole objects to facilitate role binding definitions and granular implementation of RBAC policies. Some cluster roles have rules that apply to Custom ResourceDefinitions that are part of EDB Postgres for Kubernetes, while others that are part of the broader Kubernetes/OpenShift realm.

10.4.4 Cluster roles on EDB Postgres for Kubernetes CRDs

For every CRD owned by EDB Postgres for Kubernetes's CSV, OLM deploys some predefined cluster roles that can be used by customer facing users and service accounts. In particular:

- a role for the full administration of the resource (*admin* suffix)
- a role to edit the resource (*edit* suffix)
- a role to view the resource (*view* suffix)
- a role to view the actual CRD (*crdview* suffix)

Important: Cluster roles per se are no security threat. They are the recommended way in OpenShift to define templates for roles to be later “bound” to actual users in a specific project or globally. Indeed, cluster roles can be used in conjunction with *ClusterRoleBinding* objects for global permissions or with *RoleBinding* objects for local permissions. This makes it possible to reuse cluster roles across multiple projects while enabling customization within individual projects through local roles.

You can verify the list of predefined cluster roles by running:

```
oc get clusterroles | grep postgresql
```

which returns something similar to:

| | |
|--|----------------------|
| backups.postgresql.k8s.enterprisedb.io-v1-admin | YYYY-MM-DDTHH:MM:SSZ |
| backups.postgresql.k8s.enterprisedb.io-v1-crdview | YYYY-MM-DDTHH:MM:SSZ |
| backups.postgresql.k8s.enterprisedb.io-v1-edit | YYYY-MM-DDTHH:MM:SSZ |
| backups.postgresql.k8s.enterprisedb.io-v1-view | YYYY-MM-DDTHH:MM:SSZ |
| cloud-native-postgresql.VERSION-HASH | YYYY-MM-DDTHH:MM:SSZ |
| clusters.postgresql.k8s.enterprisedb.io-v1-admin | YYYY-MM-DDTHH:MM:SSZ |
| clusters.postgresql.k8s.enterprisedb.io-v1-crdview | YYYY-MM-DDTHH:MM:SSZ |
| clusters.postgresql.k8s.enterprisedb.io-v1-edit | YYYY-MM-DDTHH:MM:SSZ |
| clusters.postgresql.k8s.enterprisedb.io-v1-view | YYYY-MM-DDTHH:MM:SSZ |
| poolers.postgresql.k8s.enterprisedb.io-v1-admin | YYYY-MM-DDTHH:MM:SSZ |
| poolers.postgresql.k8s.enterprisedb.io-v1-crdview | YYYY-MM-DDTHH:MM:SSZ |
| poolers.postgresql.k8s.enterprisedb.io-v1-edit | YYYY-MM-DDTHH:MM:SSZ |
| poolers.postgresql.k8s.enterprisedb.io-v1-view | YYYY-MM-DDTHH:MM:SSZ |
| scheduledbackups.postgresql.k8s.enterprisedb.io-v1-admin | YYYY-MM-DDTHH:MM:SSZ |
| scheduledbackups.postgresql.k8s.enterprisedb.io-v1-crdview | YYYY-MM-DDTHH:MM:SSZ |
| scheduledbackups.postgresql.k8s.enterprisedb.io-v1-edit | YYYY-MM-DDTHH:MM:SSZ |
| scheduledbackups.postgresql.k8s.enterprisedb.io-v1-view | YYYY-MM-DDTHH:MM:SSZ |

You can inspect an actual role as any other Kubernetes resource with the `get` command. For example:

```
oc get -o yaml clusterrole clusters.postgresql.k8s.enterprisedb.io-v1-admin
```

By looking at the relevant skimmed output below, you can notice that the `clusters.postgresql.k8s.enterprisedb.io-v1-admin` cluster role enableseverything on the cluster resource defined by the `postgresql.k8s.enterprisedb.io` API group:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: clusters.postgresql.k8s.enterprisedb.io-v1-admin
rules:
- apiGroups:
  - postgresql.k8s.enterprisedb.io
  resources:
  - clusters
  verbs:
  - *
```

10.4.5 Cluster roles on Kubernetes CRDs

When installing a Subscription object in a given namespace (e.g. `openshift-operators` for cluster-wide installation of the operator), OLM also creates a cluster role that is used to grant permissions to the `postgresql-operator-manager` service account that the operator uses. The name of this cluster role varies, as it depends on the installed version of the operator and the time of installation.

You can retrieve it by running the following command:

```
oc get clusterrole --selector=olm.owner.kind=ClusterServiceVersion
```

You can then use the name returned by the above query (which should have the form of `cloud-native-postgresql.VERSION-HASH`) to look at the rules, resources and verbs via the `describe` command:

```
oc describe clusterrole cloud-native-postgresql.VERSION-HASH
```

```
Name:          cloud-native-postgresql.VERSION.HASH
Labels:         olm.owner=cloud-native-postgresql.VERSION
                olm.owner.kind=ClusterServiceVersion
                olm.owner.namespace=openshift-operators
                operators.coreos.com/cloud-native-postgresql.openshift-operators=
Annotations:    <none>
PolicyRule:
  Resources                                     Non-Resource URLs
  ↪ Resource Names  Verbs
  -----
  ↪ -----
    configmaps      [create delete get list patch update watch]  []  []
    secrets         [create delete get list patch update watch]  []  []
    services        [create delete get list patch update watch]  []  []
    ↪               [create delete get list patch update watch]
```

(continues on next page)

(continued from previous page)

| | | | |
|--|----|----|---|
| deployments.apps | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| podd disruptionbudgets.policy | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| backups.postgresql.k8s.enterprisedb.io | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| clusters.postgresql.k8s.enterprisedb.io | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| poolers.postgresql.k8s.enterprisedb.io | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| scheduledbackups.postgresql.k8s.enterprisedb.io | [] | [] | ↩ |
| ↩ [create delete get list patch update watch] | | | |
| persistentvolumeclaims | [] | [] | ↩ |
| ↩ [create delete get list patch watch] | | | |
| Pods/exec | [] | [] | ↩ |
| ↩ [create delete get list patch watch] | | | |
| Pods | [] | [] | ↩ |
| ↩ [create delete get list patch watch] | | | |
| jobs.batch | [] | [] | ↩ |
| ↩ [create delete get list patch watch] | | | |
| podmonitors.monitoring.coreos.com | [] | [] | ↩ |
| ↩ [create delete get list patch watch] | | | |
| serviceaccounts | [] | [] | ↩ |
| ↩ [create get list patch update watch] | | | |
| rolebindings.rbac.authorization.k8s.io | [] | [] | ↩ |
| ↩ [create get list patch update watch] | | | |
| roles.rbac.authorization.k8s.io | [] | [] | ↩ |
| ↩ [create get list patch update watch] | | | |
| leases.coordination.k8s.io | [] | [] | ↩ |
| ↩ [create get update] | | | |
| events | [] | [] | ↩ |
| ↩ [create patch] | | | |
| mutatingwebhookconfigurations.admissionregistration.k8s.io | [] | [] | ↩ |
| ↩ [get list update] | | | |
| validatingwebhookconfigurations.admissionregistration.k8s.io | [] | [] | ↩ |
| ↩ [get list update] | | | |
| customresourcedefinitions.apiextensions.k8s.io | [] | [] | ↩ |
| ↩ [get list update] | | | |
| namespaces | [] | [] | ↩ |
| ↩ [get list watch] | | | |
| nodes | [] | [] | ↩ |
| ↩ [get list watch] | | | |
| clusters.postgresql.k8s.enterprisedb.io/status | [] | [] | ↩ |
| ↩ [get patch update watch] | | | |
| poolers.postgresql.k8s.enterprisedb.io/status | [] | [] | ↩ |
| ↩ [get patch update watch] | | | |
| configmaps/status | [] | [] | ↩ |
| ↩ [get patch update] | | | |
| secrets/status | [] | [] | ↩ |
| ↩ [get patch update] | | | |
| backups.postgresql.k8s.enterprisedb.io/status | [] | [] | ↩ |
| ↩ [get patch update] | | | |

(continues on next page)

(continued from previous page)

| | | | |
|--|----|----|---|
| scheduledbackups.postgresql.k8s.enterprisedb.io/status | [] | [] | ↪ |
| ↪ [get patch update] | | | |
| pods/status | [] | [] | ↪ |
| ↪ [get] | | | |
| clusters.postgresql.k8s.enterprisedb.io/finalizers | [] | [] | ↪ |
| ↪ [update] | | | |
| poolers.postgresql.k8s.enterprisedb.io/finalizers | [] | [] | ↪ |
| ↪ [update] | | | |

Important: The above permissions are exclusively reserved for the operator's service account to interact with the Kubernetes API server. They are not directly accessible by the users of the operator that interact only with *Cluster* , *Pooler* , *Backup* , and *ScheduledBackup* resources (see [Cluster roles on EDB Postgres for Kubernetes CRDs](#)).

The operator automates in a declarative way a lot of operations related to PostgreSQL management that otherwise would require manual and imperative interventions. Such operations also include security related matters at RBAC (e.g. service accounts), pod (e.g. security context constraints) and Postgres levels (e.g. TLS certificates).

For more information about the reasons why the operator needs these elevated permissions, refer to the [Security / Cluster / RBAC](#) .

10.5 Users and Permissions

A very common way to use the EDB Postgres for Kubernetes operator is to rely on the `cluster-admin` role and manage resources centrally.

Alternatively, you can use the RBAC framework made available by Kubernetes/OpenShift, as with any other operator or resources.

For example, you might be interested in binding the `clusters.postgresql.k8s.enterprisedb.io-v1-admin` cluster role to specific groups or users in a specific namespace, as any other cloud native application. The following example binds that cluster role to a specific user in the `web-prod` project:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: web-prod-admin
  namespace: web-prod
subjects:
- kind: User
  apiGroup: rbac.authorization.k8s.io
  name: mario@cioni.org
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: clusters.postgresql.k8s.enterprisedb.io-v1-admin
```

The same process can be repeated with any other predefined `ClusterRole` .

If, on the other hand, you prefer not to use cluster roles, you can create specific namespaced roles like in this example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: web-prod-view
  namespace: web-prod
rules:
- apiGroups:
  - postgresql.k8s.enterprisedb.io
  resources:
  - clusters
  verbs:
  - get
  - list
  - watch
```

Then, assign this role to a given user:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: web-prod-view
  namespace: web-prod
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: web-prod-view
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: web-prod-developer1
```

This final example creates a role with administration permissions (verbs isequal to *) to all the resources managed by the operator in that namespace(web-prod):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: web-prod-admin
  namespace: web-prod
rules:
- apiGroups:
  - postgresql.k8s.enterprisedb.io
  resources:
  - clusters
  - backups
  - scheduledbackups
  - poolers
  verbs:
  - *
```

10.6 Customization of the Pooler image

By default, the Pooler resource creates pods having the `pgbouncer` container that runs with the `quay.io/enterprisedb/pgbouncer` image.

You can change the image name from the advanced interface, specifically by opening the “*Template*” section, then selecting “*Add container*” under “*Spec > Containers*”:

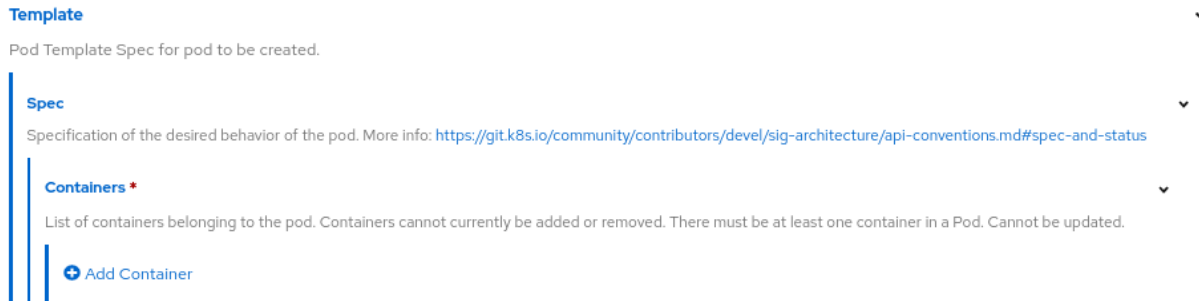


Fig. 10: Pooler template

Then:

- set `pgbouncer` as the name of the container (required field in the pod template)
- set the “*Image*” field as desired (see the image below)

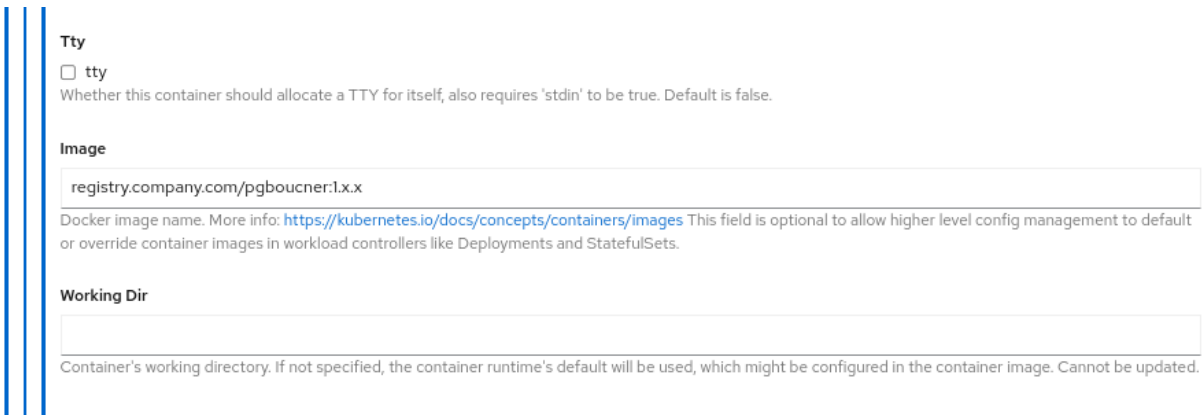


Fig. 11: Pooler image

‘FREE EVALUATION’

EDB Postgres for Kubernetes is available for a free evaluation. The process is different between Vanilla/Community PostgreSQL and EDB Postgres Advanced.

Refer to ‘License and license keys’ for terms and more details.

11.1 Evaluating PostgreSQL

By default, EDB Postgres for Kubernetes installs the latest available version of Community PostgreSQL. The operator automatically generates an implicit trial license for the cluster that lasts for 30 days.

This license is ideal for evaluation, proof of concept, integration with CI/CD pipelines, and so on.

PostgreSQL container images are available at quay.io/enterprisedb/postgresql.

11.2 Evaluating EDB Postgres Advanced Server

You can use EDB Postgres for Kubernetes with EDB Postgres Advanced Server too. You need to request a trial license key from the

[EDB website](#).

EDB Postgres Advanced container images are available at quay.io/enterprisedb/edb-postgres-advanced.

Once you have received the license key, you can use EDB Postgres Advanced Server by setting `imageName` to point to the `quay.io/enterprisedb/edb-postgres-advanced` repository in the `spec` section of the Cluster deployment configuration file.

‘LICENSE AND LICENSE KEYS’

A license key is always required for the operator to work.

The only exception is when you run the operator with Community PostgreSQL: in this case, if the license key is unset, a cluster will be started with the default trial license - which automatically expires after 30 days.

12.1 Company level license keys

A license key allows you to create an unlimited number of PostgreSQL clusters in your installation.

The license key needs to be available in a `Secret` in the same namespace where the operator is deployed (`ConfigMap` is also available, but not recommended for a license key).

Once the company level license is installed, the validity of the license key can be checked inside the cluster status.

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  [...]
  licenseStatus:
    licenseExpiration: "2021-11-06T09:36:02Z"
    licenseStatus: Trial
    valid: true
    isImplicit: false
    isTrial: true
[...]
```

12.1.1 Kubernetes installations via YAML manifest

When the operator is installed in Kubernetes using the YAML manifest, it is deployed by default in the `postgresql-operator-system` namespace.

Given the namespace name, and the license key, you can create the config map with the following command:

```
kubectl create configmap -n [NAMESPACE_NAME_HERE] \
  postgresql-operator-controller-manager-config \
  --from-literal=EDB_LICENSE_KEY=[LICENSE_KEY_HERE]
```

Operator pods will need to be recreated to apply the new configuration. You can use the following command:

```
kubectl rollout restart deployment -n [NAMESPACE_NAME_HERE] \
  postgresql-operator-controller-manager
```

12.1.2 OpenShift and Kubernetes installations with OLM

When instead OLM is used (i.e. on OpenShift or on Kubernetes from OperatorHub), you can choose to install the operator in a single namespace or to make it available in all namespaces. The second option is the default one, and the operator will be installed in `openshift-operators` on OpenShift and `operators` on OperatorHub.

You can add a given license key to an entire cluster, by creating a secret in the desired namespace (`openshift-operators`, `operators`, or the selected one) with the following command:

```
kubectl create secret generic -n [NAMESPACE_NAME_HERE] \
  postgresql-operator-controller-manager-config \
  --from-literal=EDB_LICENSE_KEY=[LICENSE_KEY_HERE]
```

You'll need to delete the current operator pods. New pods will be automatically recreated and will use the secret:

```
kubectl delete pods -n [NAMESPACE_NAME_HERE] \
  -l app.kubernetes.io/name=cloud-native-postgresql
```

12.2 Cluster level license keys

Each Cluster resource has a `licenseKey` parameter in its definition. You can find the expiration date, as well as more information about the license, in the cluster status:

```
kubectl get cluster cluster_example -o yaml
[...]
status:
  [...]
  licenseStatus:
    licenseExpiration: "2021-11-06T09:36:02Z"
    licenseStatus: Trial
    valid: true
    isImplicit: false
    isTrial: true
[...]
```

A cluster license key can be updated with a new one at any moment, to extend the expiration date or move the cluster to a production license.

12.3 License key secret at cluster level

Each `Cluster` resource can also have a `licenseKeySecret` parameter, which contains the name and key of a secret. That secret contains the license key provided by EDB.

This field will take precedence over `licenseKey` : it will be refreshed when you change the secret, in order to extend the expiration date, or switching from a trial license to a production license.

EDB Postgres for Kubernetes is distributed under the EDB Limited Usage License Agreement, available at enterprisedb.com/limited-use-license.

EDB Postgres for Kubernetes: Copyright (C) 2019-2022 EnterpriseDB Corporation.

12.4 What happens when a license expires

After the license expiration, the operator will cease any reconciliation attempt on the cluster, effectively stopping to manage its status. This also includes any self-healing and high availability capabilities, such as automated failover and switchovers.

The pods and the data will still be available.

‘API REFERENCE’

EDB Postgres for Kubernetes extends the Kubernetes API defining the following custom resources:

- *Backup*
- *Cluster*
- *Pooler*
- *ScheduledBackup*

All the resources are defined in the `postgresql.k8s.enterprisedb.io/v1` API.

Below you will find a description of the defined resources:

- *AffinityConfiguration*
- *AzureCredentials*
- *Backup*
- *BackupConfiguration*
- *BackupList*
- *BackupSource*
- *BackupSpec*
- *BackupStatus*
- *BarmanObjectStoreConfiguration*
- *BootstrapConfiguration*
- *BootstrapInitDB*
- *BootstrapPgBaseBackup*
- *BootstrapRecovery*
- *CertificatesConfiguration*
- *CertificatesStatus*
- *Cluster*
- *ClusterCondition*
- *ClusterList*
- *ClusterSpec*
- *ClusterStatus*

- *ConfigMapKeySelector*
- *ConfigMapResourceVersion*
- *DataBackupConfiguration*
- *EPASConfiguration*
- *EmbeddedObjectMetadata*
- *ExternalCluster*
- *GoogleCredentials*
- *InstanceID*
- *LDAPBindAsAuth*
- *LDAPBindSearchAuth*
- *LDAPConfig*
- *LocalObjectReference*
- *MonitoringConfiguration*
- *NodeMaintenanceWindow*
- *PgBouncerIntegrationStatus*
- *PgBouncerSecrets*
- *PgBouncerSpec*
- *PodMeta*
- *PodTemplateSpec*
- *Pooler*
- *PoolerIntegrations*
- *PoolerList*
- *PoolerSecrets*
- *PoolerSpec*
- *PoolerStatus*
- *PostgresConfiguration*
- *RecoveryTarget*
- *ReplicaClusterConfiguration*
- *RollingUpdateStatus*
- *S3Credentials*
- *ScheduledBackup*
- *ScheduledBackupList*
- *ScheduledBackupSpec*
- *ScheduledBackupStatus*
- *SecretKeySelector*
- *SecretVersion*

- *SecretsResourceVersion*
- *StorageConfiguration*
- *WalBackupConfiguration*

13.1 AffinityConfiguration

AffinityConfiguration contains the info we need to create the affinity rules for Pods

| Name | Description | Type |
|--|---|-------------------------|
| <code>enablePodAntiAffinity</code> | Activates anti-affinity for the pods. The operator will define pods anti-affinity unless this field is explicitly set to false | bool |
| <code>topologyKey</code> | TopologyKey to use for anti-affinity configuration. See k8s documentation for more info on that - <i>mandatory</i> | string |
| <code>nodeSelector</code> | NodeSelector is map of key-value pairs used to define the nodes on which the pods can run. More info: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ | map[string]string |
| <code>tolerations</code> | Tolerations is a list of Tolerations that should be set for all the pods, in order to allow them to run on tainted nodes. More info: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/ | corev1.Toleration |
| <code>podAntiAffinityType</code> | PodAntiAffinityType allows the user to decide whether pod anti-affinity between cluster instance has to be considered a strong requirement during scheduling or not. Allowed values are: “preferred” (default if empty) or “required”. Setting it to “required”, could lead to instances remaining pending until new kubernetes nodes are added if all the existing nodes don’t match the required pod anti-affinity rule. More info: https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/#inter-pod-affinity-and-anti-affinity | string |
| <code>additionalPodAntiAffinity</code> | AdditionalPodAntiAffinity allows to specify pod anti-affinity terms to be added to the ones generated by the operator if EnablePodAntiAffinity is set to true (default) or to be used exclusively if set to false. | *corev1.PodAntiAffinity |
| <code>additionalPodAffinity</code> | AdditionalPodAffinity allows to specify pod affinity terms to be passed to all the cluster’s pods. | *corev1.PodAffinity |

13.2 AzureCredentials

AzureCredentials is the type for the credentials to be used to upload files to Azure Blob Storage. The connection string contains every needed information. If the connection string is not specified, we'll need the storage account name and also one (and only one) of:

- storageKey - storageSasToken

| Name | Description | Type |
|-------------------------------|---|--------------------------|
| <code>connectionString</code> | The connection string to be used | <i>SecretKeySelector</i> |
| <code>storageAccount`</code> | The storage account where to upload data | <i>SecretKeySelector</i> |
| <code>storageKey`</code> | The storage account key to be used in conjunction with the storage account name | <i>SecretKeySelector</i> |
| <code>storageSasToken`</code> | A shared-access-signature to be used in conjunction with the storage account name | <i>SecretKeySelector</i> |

13.3 Backup

Backup is the Schema for the backups API

| Name | Description | Type |
|-----------------------|--|--------------------------|
| <code>metadata</code> | | <i>metav1.ObjectMeta</i> |
| <code>spec`</code> | Specification of the desired behavior of the backup. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | <i>BackupSpec</i> |
| <code>status`</code> | Most recently observed status of the backup. This data may not be up to date. Populated by the system. Read-only. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | <i>BackupStatus</i> |

13.4 BackupConfiguration

BackupConfiguration defines how the backup of the cluster are taken. Currently the only supported backup method is barmanObjectStore. For details and examples refer to the Backup and Recovery section of the documentation

| Name | Description | Type |
|--------------------------------|--|---------------------------------------|
| <code>barmanObjectStore</code> | The configuration for the barman-cloud tool suite | <i>BarmanObjectStoreConfiguration</i> |
| <code>retentionPolicy`</code> | RetentionPolicy is the retention policy to be used for backups and WALs (i.e. '60d'). The retention policy is expressed in the form of <code>XXu</code> where <code>XX</code> is a positive integer and <code>u</code> is in <code>[dwm]</code> - days, weeks, months. | string |

13.5 BackupList

BackupList contains a list of Backup

| Name | Description | Type |
|-----------------|--|------------------------|
| <i>metadata</i> | Standard list metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds | <i>metav1.ListMeta</i> |
| <i>items</i> | List of backups - <i>mandatory</i> | <i>Backup</i> |

13.6 BackupSource

BackupSource contains the backup we need to restore from, plus some information that could be needed to correctly restore it.

| Name | Description | Type |
|-------------------|---|--------------------------|
| <i>endpointCA</i> | EndpointCA store the CA bundle of the barman endpoint. Useful when using self-signed certificates to avoid errors with certificate issuer and barman-cloud-wal-archive. | <i>SecretKeySelector</i> |

13.7 BackupSpec

BackupSpec defines the desired state of Backup

| Name | Description | Type |
|----------------|-----------------------|-----------------------------|
| <i>cluster</i> | The cluster to backup | <i>LocalObjectReference</i> |

13.8 BackupStatus

BackupStatus defines the observed state of Backup

| Name | Description | Type |
|--------------------------|---|--------------------------|
| <i>s3Credentials</i> | The credentials to be used to upload data to S3 | <i>S3Credentials</i> |
| <i>azureCredentials</i> | The credentials to be used to upload data to Azure Blob Storage | <i>AzureCredentials</i> |
| <i>googleCredentials</i> | The credentials to use to upload data to Google Cloud Storage | <i>GoogleCredentials</i> |
| <i>endpointCA</i> | EndpointCA store the CA bundle of the barman endpoint. Useful when using self-signed certificates to avoid errors with certificate issuer and barman-cloud-wal-archive. | <i>SecretKeySelector</i> |

continues on next page

Table 8 – continued from previous page

| Name | Description | Type |
|--------------------------------|---|-------------------|
| <code>`endpointURL`</code> | Endpoint to be used to upload data to the cloud, overriding the automatic endpoint discovery | string |
| <code>`destinationPath`</code> | The path where to store the backup (i.e. <code>s3://bucket/path/to/folder</code>) this path, with different destination folders, will be used for WALs and for data - <i>mandatory</i> | string |
| <code>`serverName`</code> | The server name on S3, the cluster name is used if this parameter is omitted | string |
| <code>`encryption`</code> | Encryption method required to S3 API | string |
| <code>`backupId`</code> | The ID of the Barman backup | string |
| <code>`phase`</code> | The last backup status | BackupPhase |
| <code>`startedAt`</code> | When the backup was started | metav1.Time |
| <code>`stoppedAt`</code> | When the backup was terminated | metav1.Time |
| <code>`beginWal`</code> | The starting WAL | string |
| <code>`endWal`</code> | The ending WAL | string |
| <code>`beginLSN`</code> | The starting xlog | string |
| <code>`endLSN`</code> | The ending xlog | string |
| <code>`error`</code> | The detected error | string |
| <code>`commandOutput`</code> | Unused. Retained for compatibility with old versions. | string |
| <code>`commandError`</code> | The backup command output in case of error | string |
| <code>`instanceID`</code> | Information to identify the instance where the backup has been taken from | <i>InstanceID</i> |

13.9 BarmanObjectStoreConfiguration

BarmanObjectStoreConfiguration contains the backup configuration using Barman against an S3-compatible object storage

| Name | Description | Type |
|---------------------------------|---|--------------------------|
| <code>`s3Credentials`</code> | The credentials to use to upload data to S3 | <i>S3Credentials</i> |
| <code>`azureCredentials`</code> | The credentials to use to upload data to Azure Blob Storage | <i>AzureCredentials</i> |
| <code>googleCredentials</code> | The credentials to use to upload data to Google Cloud Storage | <i>GoogleCredentials</i> |
| <code>`endpointURL`</code> | Endpoint to be used to upload data to the cloud, overriding the automatic endpoint discovery | string |
| <code>`endpointCA`</code> | EndpointCA store the CA bundle of the barman endpoint. Useful when using self-signed certificates to avoid errors with certificate issuer and barman-cloud-wal-archive | <i>SecretKeySelector</i> |
| <code>`destinationPath`</code> | The path where to store the backup (i.e. <code>s3://bucket/path/to/folder</code>) this path, with different destination folders, will be used for WALs and for data - <i>mandatory</i> | string |
| <code>`serverName`</code> | The server name on S3, the cluster name is used if this parameter is omitted | string |

continues on next page

Table 9 – continued from previous page

| Name | Description | Type |
|----------------------------|---|--------------------------------|
| <code>`wal`</code> | The configuration for the backup of the WAL stream. When not defined, WAL files will be stored uncompressed and may be unencrypted in the object store, according to the bucket default policy. | <i>WalBackupConfiguration</i> |
| <code>`data`</code> | The configuration to be used to backup the data files. When not defined, base backups files will be stored uncompressed and may be unencrypted in the object store, according to the bucket default policy. | <i>DataBackupConfiguration</i> |
| <code>`tags`</code> | Tags is a list of key value pairs that will be passed to the Barman <code>--tags</code> option. | map[string]string |
| <code>`historyTags`</code> | HistoryTags is a list of key value pairs that will be passed to the Barman <code>--history-tags</code> option. | map[string]string |

13.10 BootstrapConfiguration

BootstrapConfiguration contains information about how to create the PostgreSQL cluster. Only a single bootstrap method can be defined among the supported ones. `initdb` will be used as the bootstrap method if left unspecified. Refer to the Bootstrap page of the documentation for more information.

| Name | Description | Type |
|----------------------------|--|------------------------------|
| <code>`initdb`</code> | Bootstrap the cluster via initdb | <i>BootstrapInitDB</i> |
| <code>`recovery`</code> | Bootstrap the cluster from a backup | <i>BootstrapRecovery</i> |
| <code>pg_basebackup</code> | Bootstrap the cluster taking a physical backup of another compatible PostgreSQL instance | <i>BootstrapPgBaseBackup</i> |

13.11 BootstrapInitDB

BootstrapInitDB is the configuration of the bootstrap process when `initdb` is used. Refer to the Bootstrap page of the documentation for more information.

| Name | Description | Type |
|-------------------------|---|-----------------------------|
| <code>`database`</code> | Name of the database used by the application. Default: <i>app. - mandatory</i> | string |
| <code>`owner`</code> | Name of the owner of the database in the instance to be used by applications. Defaults to the value of the <i>database</i> key. - mandatory | string |
| <code>`secret`</code> | Name of the secret containing the initial credentials for the owner of the user database. If empty a new secret will be created from scratch | <i>LocalObjectReference</i> |
| <code>`redwood`</code> | If we need to enable/disable Redwood compatibility. Requires EPAS and for EPAS defaults to true | <i>*bool</i> |
| <code>`options`</code> | The list of options that must be passed to <code>initdb</code> when creating the cluster. Deprecated: This could lead to inconsistent configurations, please use the explicit provided parameters instead. If defined, explicit values will be ignored. | string |

continues on next page

Table 11 – continued from previous page

| Name | Description | Type |
|-------------------------------------|--|--------------|
| <code>`dataChecksums`</code> | Whether the <code>-k</code> option should be passed to initdb, enabling checksums on data pages (default: <i>false</i>) | <i>*bool</i> |
| <code>`encoding`</code> | The value to be passed as option <code>--encoding</code> for initdb (default: <i>UTF8</i>) | string |
| <code>`localeCollate`</code> | The value to be passed as option <code>--lc-collate</code> for initdb (default: <i>C</i>) | string |
| <code>`localeCTYPE`</code> | The value to be passed as option <code>--lc-ctype</code> for initdb (default: <i>C</i>) | string |
| <code>`walSegmentSize`</code> | The value in megabytes (1 to 1024) to be passed to the <code>--wal-segsize</code> option for initdb (default: empty, resulting in PostgreSQL default: 16MB) | int |
| <code>`postInitSQL`</code> | List of SQL queries to be executed as a superuser immediately after the cluster has been created - to be used with extreme care (by default empty) | string |
| <code>postInitApplicationSQL</code> | List of SQL queries to be executed as a superuser in the application database right after is created - to be used with extreme care (by default empty) | string |
| <code>`postInitTemplateSQL`</code> | List of SQL queries to be executed as a superuser in the <i>template1</i> after the cluster has been created - to be used with extreme care (by default empty) | string |

13.12 BootstrapPgBaseBackup

BootstrapPgBaseBackup contains the configuration required to take a physical backup of an existing PostgreSQL cluster

| Name | Description | Type |
|---------------------|--|--------|
| <code>source</code> | The name of the server of which we need to take a physical backup - <i>mandatory</i> | string |

13.13 BootstrapRecovery

BootstrapRecovery contains the configuration required to restore the backup with the specified name and, after having changed the password with the one chosen for the superuser, will use it to bootstrap a full cluster cloning all the instances from the restored primary. Refer to the Bootstrap page of the documentation for more information.

| Name | Description | Type |
|-----------------------|--|---------------------|
| <code>`backup`</code> | The backup we need to restore | <i>BackupSource</i> |
| <code>`source`</code> | The external cluster whose backup we will restore. This is also used as the name of the folder under which the backup is stored, so it must be set to the name of the source cluster | string |

continues on next page

Table 13 – continued from previous page

| Name | Description | Type |
|-----------------------|---|-----------------------|
| <i>recoveryTarget</i> | By default, the recovery process applies all the available WAL files in the archive (full recovery). However, you can also end the recovery as soon as a consistent state is reached or recover to a point-in-time (PITR) by specifying a <i>RecoveryTarget</i> object, as expected by PostgreSQL (i.e., timestamp, transaction Id, LSN, ...). More info: https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET | <i>RecoveryTarget</i> |

13.14 CertificatesConfiguration

CertificatesConfiguration contains the needed configurations to handle server certificates.

| Name | Description | Type |
|---------------------------------|---|--------|
| <code>serverCASecret`</code> | The secret containing the Server CA certificate. If not defined, a new secret will be created with a self-signed CA and will be used to generate the TLS certificate <code>ServerTLSSecret</code> .
Contains:
- <code>ca.crt</code> : CA that should be used to validate the server certificate, used as <code>sslrootcert</code> in client connection strings.
- <code>ca.key</code> : key used to generate Server SSL certs, if <code>ServerTLSSecret</code> is provided, this can be omitted. | string |
| <code>serverTLSSecret`</code> | The secret of type <code>kubernetes.io/tls</code> containing the server TLS certificate and key that will be set as <code>ssl_cert_file</code> and <code>ssl_key_file</code> so that clients can connect to postgres securely. If not defined, <code>ServerCASecret</code> must provide also <code>ca.key</code> and a new secret will be created using the provided CA. | string |
| <i>replicationTLSSecret</i> | The secret of type <code>kubernetes.io/tls</code> containing the client certificate to authenticate as the <i>streaming_replica</i> user. If not defined, <code>ClientCASecret</code> must provide also <code>ca.key</code> , and a new secret will be created using the provided CA. | string |
| <code>clientCASecret`</code> | The secret containing the Client CA certificate. If not defined, a new secret will be created with a self-signed CA and will be used to generate all the client certificates.
Contains:
- <code>ca.crt</code> : CA that should be used to validate the client certificates, used as <code>ssl_ca_file</code> of all the instances.
- <code>ca.key</code> : key used to generate client certificates, if <code>ReplicationTLSSecret</code> is provided, this can be omitted. | string |
| <code>serverAltDNSNames`</code> | The list of the server alternative DNS names to be added to the generated server TLS certificates, when required. | string |

13.15 CertificatesStatus

CertificatesStatus contains configuration certificates and related expiration dates.

| Name | Description | Type |
|--------------------|--|-------------------|
| <i>expirations</i> | Expiration dates for all certificates. | map[string]string |

13.16 Cluster

Cluster is the Schema for the PostgreSQL API

| Name | Description | Type |
|-----------------|---|-----------------------------------|
| <i>metadata</i> | | metav1.ObjectMeta |
| <i>spec</i> | Specification of the desired behavior of the cluster. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | ClusterSpec |
| <i>status</i> | Most recently observed status of the cluster. This data may not be up to date. Populated by the system. Read-only. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | ClusterStatus |

13.17 ClusterCondition

ClusterCondition describes the state of a cluster object at a certain point

| Name | Description | Type |
|---------------------------|---|-----------------------------|
| <i>type</i> | Type of the condition. | ClusterConditionType |
| <i>status</i> | Status of the condition, one of True, False, Unknown. | ConditionStatus |
| <i>lastTransitionTime</i> | Last time the condition transitioned from one status to another. | metav1.Time |
| <i>reason</i> | The reason for the condition's last transition. | string |
| <i>message</i> | A human readable message indicating details about the transition. | string |

13.18 ClusterList

ClusterList contains a list of Cluster

| Name | Description | Type |
|-----------------------|--|------------------------------|
| <code>metadata</code> | Standard list metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds | <code>metav1.ListMeta</code> |
| <code>items</code> | List of clusters - <i>mandatory</i> | <i>Cluster</i> |

13.19 ClusterSpec

ClusterSpec defines the desired state of Cluster

| Name | Description | Type |
|--------------------------------|--|------------------------------------|
| <code>description</code> | Description of this PostgreSQL cluster | string |
| <code>inheritedMetadata</code> | Metadata that will be inherited by all objects related to the Cluster | <i>EmbeddedObjectMetadata</i> |
| <code>imageName</code> | Name of the container image, supporting both tags (<code><image>:<tag></code>) and digests for deterministic and repeatable deployments (<code><image>:<tag>@sha256:<digestValue></code>) | string |
| <code>imagePullPolicy</code> | Image pull policy. One of <i>Always</i> , <i>Never</i> or <i>IfNotPresent</i> . If not defined, it defaults to <i>IfNotPresent</i> . Cannot be updated. More info: https://kubernetes.io/docs/concepts/containers/images#updating-images | <code>corev1.PullPolicy</code> |
| <code>postgresUID</code> | The UID of the <i>postgres</i> user inside the image, defaults to 26 | int64 |
| <code>postgresGID</code> | The GID of the <i>postgres</i> user inside the image, defaults to 26 | int64 |
| <code>instances</code> | Number of instances required in the cluster - <i>mandatory</i> | int32 |
| <code>minSyncReplicas</code> | Minimum number of instances required in synchronous replication with the primary. Undefined or 0 allow writes to complete when no standby is available. | int32 |
| <code>maxSyncReplicas</code> | The target value for the synchronous replication quorum, that can be decreased if the number of ready standbys is lower than this. Undefined or 0 disable synchronous replication. | int32 |
| <code>postgresql</code> | Configuration of the PostgreSQL server | <i>PostgresConfiguration</i> |
| <code>bootstrap</code> | Instructions to bootstrap this cluster | <i>BootstrapConfiguration</i> |
| <code>replica</code> | Replica cluster configuration | <i>ReplicaClusterConfiguration</i> |
| <code>superuserSecret</code> | The secret containing the superuser password. If not defined a new secret will be created with a randomly generated password | <i>LocalObjectReference</i> |

continues on next page

Table 19 – continued from previous page

| Name | Description | Type |
|------------------------------------|---|----------------------------------|
| <i>enableSuperuserAccess</i> | When this option is enabled, the operator will use the <i>SuperuserSecret</i> to update the <i>postgres</i> user password (if the secret is not present, the operator will automatically create one). When this option is disabled, the operator will ignore the <i>SuperuserSecret</i> content, delete it when automatically created, and then blank the password of the <i>postgres</i> user by setting it to <i>NULL</i> . Enabled by default. | *bool |
| <code>`certificates`</code> | The configuration for the CA and related certificates | <i>CertificatesConfiguration</i> |
| <code>`imagePullSecrets`</code> | The list of pull secrets to be used to pull the images. If the license key contains a pull secret that secret will be automatically included. | <i>LocalObjectReference</i> |
| <code>`storage`</code> | Configuration of the storage of the instances | <i>StorageConfiguration</i> |
| <code>`startDelay`</code> | The time in seconds that is allowed for a PostgreSQL instance to successfully start up (default 30) | int32 |
| <code>`stopDelay`</code> | The time in seconds that is allowed for a PostgreSQL instance to gracefully shutdown (default 30) | int32 |
| <code>`switchoverDelay`</code> | The time in seconds that is allowed for a primary PostgreSQL instance to gracefully shutdown during a switchover. Default value is 40000000, greater than one year in seconds, big enough to simulate an infinite delay | int32 |
| <code>`affinity`</code> | Affinity/Anti-affinity rules for Pods | <i>AffinityConfiguration</i> |
| <code>`resources`</code> | Resources requirements of every generated Pod. Please refer to https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ for more information. | corev1.ResourceRequirements |
| <i>primaryUpdateStrategy</i> | Strategy to follow to upgrade the primary server during a rolling update procedure, after all replicas have been successfully updated: it can be automated (<i>unsupervised</i> - default) or manual (<i>supervised</i>) | PrimaryUpdateStrategy |
| <code>`primaryUpdateMethod`</code> | Method to follow to upgrade the primary server during a rolling update procedure, after all replicas have been successfully updated: it can be with a switchover (<i>switchover</i> - default) or in-place (<i>restart</i>) | PrimaryUpdateMethod |
| <code>`backup`</code> | The configuration to be used for backups | <i>BackupConfiguration</i> |
| <i>nodeMaintenanceWindow</i> | Define a maintenance window for the Kubernetes nodes | <i>NodeMaintenanceWindow</i> |
| <code>`licenseKey`</code> | The license key of the cluster. When empty, the cluster operates in trial mode and after the expiry date (default 30 days) the operator will cease any reconciliation attempt. For details, please refer to the license agreement that comes with the operator. | string |
| <code>`licenseKeySecret`</code> | The reference to the license key. When this is set it take precedence over LicenseKey. | corev1.SecretKeySelector |
| <code>`monitoring`</code> | The configuration of the monitoring infrastructure of this cluster | <i>MonitoringConfiguration</i> |
| <code>`externalClusters`</code> | The list of external clusters which are used in the configuration | <i>ExternalCluster</i> |

continues on next page

Table 19 – continued from previous page

| Name | Description | Type |
|-------------------------|---|--------|
| <code>`logLevel`</code> | The instances' log level, one of the following values: error, warning, info (default), debug, trace | string |

13.20 ClusterStatus

ClusterStatus defines the observed state of Cluster

| Name | Description | Type |
|---|--|---------------------------------|
| <code>`instances`</code> | Total number of instances in the cluster | int32 |
| <code>`readyInstances`</code> | Total number of ready instances in the cluster | int32 |
| <code>`instancesStatus`</code> | Instances status | map[utils.PodStatus]string |
| <code>`latestGeneratedNode`</code> | ID of the latest generated node (used to avoid node name clashing) | int32 |
| <code>`currentPrimary`</code> | Current primary instance | string |
| <code>`targetPrimary`</code> | Target primary instance, this is different from the previous one during a switchover or a failover | string |
| <code>`pvcCount`</code> | How many PVCs have been created by this cluster | int32 |
| <code>`jobCount`</code> | How many Jobs have been created by this cluster | int32 |
| <code>`danglingPVC`</code> | List of all the PVCs created by this cluster and still available which are not attached to a Pod | string |
| <code>`resizingPVC`</code> | List of all the PVCs that have ResizingPVC condition. | string |
| <code>`initializingPVC`</code> | List of all the PVCs that are being initialized by this cluster | string |
| <code>`healthyPVC`</code> | List of all the PVCs not dangling nor initializing | string |
| <code>`licenseStatus`</code> | Status of the license | licensekey.Status |
| <code>`writeService`</code> | Current write pod | string |
| <code>`readService`</code> | Current list of read pods | string |
| <code>`phase`</code> | Current phase of the cluster | string |
| <code>`phaseReason`</code> | Reason for the current phase | string |
| <code>`secretsResourceVersion`</code> | The list of resource versions of the secrets managed by the operator. Every change here is done in the interest of the instance manager, which will refresh the secret data | <i>SecretsResourceVersion</i> |
| <code>`configMapResourceVersion`</code> | The list of resource versions of the configmaps, managed by the operator. Every change here is done in the interest of the instance manager, which will refresh the configmap data | <i>ConfigMapResourceVersion</i> |
| <code>`certificates`</code> | The configuration for the CA and related certificates, initialized with defaults. | <i>CertificatesStatus</i> |
| <code>`firstRecoverability-Point`</code> | The first recoverability point, stored as a date in RFC3339 format | string |
| <code>`cloudNativePostgres-CommitHash`</code> | The commit hash number of which this operator running | string |
| <code>`currentPrimaryTimestamp`</code> | The timestamp when the last actual promotion to primary has occurred | string |
| <code>`targetPrimaryTimestamp`</code> | The timestamp when the last request for a new primary has occurred | string |

continues on next page

Table 20 – continued from previous page

| Name | Description | Type |
|---|---|---------------------------|
| <code>`poolerIntegrations`</code> | The integration needed by poolers referencing the cluster | <i>PoolerIntegrations</i> |
| <code>cloudNativePostgresql-OperatorHash</code> | The hash of the binary of the operator | string |
| <code>`onlineUpdateEnabled`</code> | OnlineUpdateEnabled shows if the online upgrade is enabled inside the cluster | bool |
| <code>`azurePVCUpdateEnabled`</code> | AzurePVCUpdateEnabled shows if the PVC online upgrade is enabled for this cluster | bool |
| <code>`conditions`</code> | Conditions for cluster object | <i>ClusterCondition</i> |

13.21 ConfigMapKeySelector

ConfigMapKeySelector contains enough information to let you locate the key of a ConfigMap

| Name | Description | Type |
|------------------|--------------------------------------|--------|
| <code>key</code> | The key to select - <i>mandatory</i> | string |

13.22 ConfigMapResourceVersion

ConfigMapResourceVersion is the resource versions of the secrets managed by the operator

| Name | Description | Type |
|----------------------|---|-------------------|
| <code>metrics</code> | A map with the versions of all the config maps used to pass metrics. Map keys are the config map names, map values are the versions | map[string]string |

13.23 DataBackupConfiguration

DataBackupConfiguration is the configuration of the backup of the data directory

| Name | Description | Type |
|----------------------------|--|-----------------|
| <code>`compression`</code> | Compress a backup file (a tar file per tablespace) while streaming it to the object store. Available options are empty string (no compression, default), <i>gzip</i> , <i>bzip2</i> or <i>snappy</i> . | CompressionType |
| <code>`encryption`</code> | Whenever to force the encryption of files (if the bucket is not already configured for that). Allowed options are empty string (use the bucket policy, default), <i>AES256</i> and <i>aws:kms</i> | EncryptionType |

continues on next page

Table 23 – continued from previous page

| Name | Description | Type |
|----------------------------|--|--------|
| <i>immediateCheckpoint</i> | Control whether the I/O workload for the backup initial checkpoint will be limited, according to the <i>checkpoint_completion_target</i> setting on the PostgreSQL server. If set to true, an immediate checkpoint will be used, meaning PostgreSQL will complete the checkpoint as soon as possible. <i>false</i> by default. | bool |
| <i>jobs</i> | The number of parallel jobs to be used to upload the backup, defaults to 2 | *int32 |

13.24 EPASConfiguration

EPASConfiguration contains EDB Postgres Advanced Server specific configurations

| Name | Description | Type |
|--------------|-----------------------------------|------|
| <i>audit</i> | If true enables edb_audit logging | bool |

13.25 EmbeddedObjectMetadata

EmbeddedObjectMetadata contains metadata to be inherited by all resources related to a Cluster

| Name | Description | Type |
|--------------------|-------------|-------------------|
| <i>labels</i> | | map[string]string |
| <i>annotations</i> | | map[string]string |

13.26 ExternalCluster

ExternalCluster represents the connection parameters to an external cluster which is used in the other sections of the configuration

| Name | Description | Type |
|-----------------------------|--|---------------------------------------|
| <i>name</i> | The server name, required - <i>mandatory</i> | string |
| <i>connectionParameters</i> | The list of connection parameters, such as dbname, host, username, etc | map[string]string |
| <i>sslCert</i> | The reference to an SSL certificate to be used to connect to this instance | corev1.SecretKeySelector |
| <i>sslKey</i> | The reference to an SSL private key to be used to connect to this instance | corev1.SecretKeySelector |
| <i>sslRootCert</i> | The reference to an SSL CA public key to be used to connect to this instance | corev1.SecretKeySelector |
| <i>password</i> | The reference to the password to be used to connect to the server | corev1.SecretKeySelector |
| <i>barmanObjectStore</i> | The configuration for the barman-cloud tool suite | <i>BarmanObjectStoreConfiguration</i> |

13.27 GoogleCredentials

GoogleCredentials is the type for the Google Cloud Storage credentials. This needs to be specified even if we run inside a GKE environment.

| Name | Description | Type |
|-------------------------------------|---|--|
| <code>gkeEnvironment</code> | If set to true, will presume that it's running inside a GKE environment, default to false. - <i>mandatory</i> | bool |
| <code>applicationCredentials</code> | The secret containing the Google Cloud Storage JSON file with the credentials | <i>SecretKeySelector</i> |

13.28 InstanceID

InstanceID contains the information to identify an instance

| Name | Description | Type |
|--------------------------|------------------|--------|
| <code>podName</code> | The pod name | string |
| <code>ContainerID</code> | The container ID | string |

13.29 LDAPBindAsAuth

LDAPBindAsAuth provides the required fields to use the bind authentication for LDAP

| Name | Description | Type |
|---------------------|---|--------|
| <code>prefix</code> | Prefix for the bind authentication option | string |
| <code>suffix</code> | Suffix for the bind authentication option | string |

13.30 LDAPBindSearchAuth

LDAPBindSearchAuth provides the required fields to use the bind+search LDAP authentication process

| Name | Description | Type |
|------------------------------|--|---|
| <code>baseDN</code> | Root DN to begin the user search | string |
| <code>bindDN</code> | DN of the user to bind to the directory | string |
| <code>bindPassword</code> | Secret with the password for the user to bind to the directory | <i>corev1.SecretKeySelector</i> |
| <code>searchAttribute</code> | Attribute to match against the username | string |
| <code>searchFilter</code> | Search filter to use when doing the search+bind authentication | string |

13.31 LDAPConfig

LDAPConfig contains the parameters needed for LDAP authentication

| Name | Description | Type |
|-----------------------------|---|------------------------------------|
| <code>server`</code> | LDAP hostname or IP address | string |
| <code>port`</code> | LDAP server port | int |
| <code>scheme`</code> | LDAP schema to be used, possible options are <i>ldap</i> and <i>ldaps</i> | LDAPScheme |
| <code>tls`</code> | Set to 1 to enable LDAP over TLS | bool |
| <code>bindAsAuth`</code> | Bind as authentication configuration | LDAPBindAsAuth |
| <code>bindSearchAuth</code> | Bind+Search authentication configuration | LDAPBindSearchAuth |

13.32 LocalObjectReference

LocalObjectReference contains enough information to let you locate a local object with a known type inside the same namespace

| Name | Description | Type |
|-------------------|--|--------|
| <code>name</code> | Name of the referent. - <i>mandatory</i> | string |

13.33 MonitoringConfiguration

MonitoringConfiguration is the type containing all the monitoring configuration for a certain cluster

| Name | Description | Type |
|-------------------------------------|---|--|
| <code>disableDefaultQueries`</code> | Whether the default queries should be injected. Set it to <i>true</i> if you don't want to inject default queries into the cluster. Default: false. | <i>*bool</i> |
| <code>customQueriesConfigMap</code> | The list of config maps containing the custom queries | ConfigMapKeySelector |
| <code>customQueriesSecret`</code> | The list of secrets containing the custom queries | SecretKeySelector[<i>(#SecretKeySelector)</i>] |
| <code>enablePodMonitor`</code> | Enable or disable the <i>PodMonitor</i> | bool |

13.34 NodeMaintenanceWindow

NodeMaintenanceWindow contains information that the operator will use while upgrading the underlying node.

This option is only useful when the chosen storage prevents the Pods from being freely moved across nodes.

| Name | Description | Type |
|-------------------|--|--------------|
| <i>inProgress</i> | Is there a node maintenance activity in progress? - <i>mandatory</i> | bool |
| <i>reusePVC</i> | Reuse the existing PVC (wait for the node to come up again) or not (recreate it elsewhere - when <i>instances</i> >1) - <i>mandatory</i> | <i>*bool</i> |

13.35 PgBouncerIntegrationStatus

PgBouncerIntegrationStatus encapsulates the needed integration for the pgbouncer poolers referencing the cluster

| Name | Description | Type |
|----------------|-------------|--------|
| <i>secrets</i> | | string |

13.36 PgBouncerSecrets

PgBouncerSecrets contains the versions of the secrets used by pgbouncer

| Name | Description | Type |
|------------------|-------------------------------|----------------------|
| <i>authQuery</i> | The auth query secret version | <i>SecretVersion</i> |

13.37 PgBouncerSpec

PgBouncerSpec defines how to configure PgBouncer

| Name | Description | Type |
|------------------------|--|-----------------------------|
| <i>poolMode</i> | The pool mode - <i>mandatory</i> | PgBouncerPoolMode |
| <i>authQuerySecret</i> | The credentials of the user that need to be used for the authentication query. In case it is specified, also an AuthQuery (e.g. “SELECT username, passwd FROM pg_shadow WHERE username=\$1”) has to be specified and no automatic CNP Cluster integration will be triggered. | <i>LocalObjectReference</i> |

continues on next page

Table 37 – continued from previous page

| Name | Description | Type |
|-------------------------|--|-------------------|
| <code>authQuery</code> | The query that will be used to download the hash of the password of a certain user. Default: “SELECT username, passwd FROM user_search(\$1)”. In case it is specified, also an AuthQuerySecret has to be specified and no automatic CNP Cluster integration will be triggered. | string |
| <code>parameters</code> | Additional parameters to be passed to PgBouncer - please check the CNP documentation for a list of options you can configure | map[string]string |
| <code>paused</code> | When set to <i>true</i> , PgBouncer will disconnect from the PostgreSQL server, first waiting for all queries to complete, and pause all new client connections until this value is set to <i>false</i> (default). Internally, the operator calls PgBouncer’s <i>PAUSE</i> and <i>RESUME</i> commands. | bool |

13.38 PodMeta

PodMeta is a structure similar to the `metav1.ObjectMeta`, but still parseable by controller-gen to create a suitable CRD for the user. The comment of `PodTemplateSpec` has an explanation of why we are not using the core data types.

| Name | Description | Type |
|--------------------------|---|-------------------|
| <code>labels</code> | Map of string keys and values that can be used to organize and categorize (scope and select) objects. May match selectors of replication controllers and services. More info: http://kubernetes.io/docs/user-guide/labels | map[string]string |
| <code>annotations</code> | Annotations is an unstructured key value map stored with a resource that may be set by external tools to store and retrieve arbitrary metadata. They are not queryable and should be preserved when modifying objects. More info: http://kubernetes.io/docs/user-guide/annotations | map[string]string |

13.39 PodTemplateSpec

PodTemplateSpec is a structure allowing the user to set a template for Pod generation.

Unfortunately we can’t use the `corev1.PodTemplateSpec` type because the generated CRD won’t have the field for the metadata section.

References: <https://github.com/kubernetes-sigs/controller-tools/issues/385> <https://github.com/kubernetes-sigs/controller-tools/issues/448> <https://github.com/prometheus-operator/prometheus-operator/issues/3041>

| Name | Description | Type |
|-----------------------|--|----------------|
| <code>metadata</code> | Standard object’s metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata | <i>PodMeta</i> |

continues on next page

Table 39 – continued from previous page

| Name | Description | Type |
|-------------------|--|-----------------------------|
| <code>spec</code> | Specification of the desired behavior of the pod. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | <code>corev1.PodSpec</code> |

13.40 Pooler

Pooler is the Schema for the poolers API

| Name | Description | Type |
|---------------------|-------------|--------------------------|
| <i>metadata</i> | | <i>metav1.ObjectMeta</i> |
| <code>spec</code> | | <i>PoolerSpec</i> |
| <code>status</code> | | <i>PoolerStatus</i> |

13.41 PoolerIntegrations

PoolerIntegrations encapsulates the needed integration for the poolers referencing the cluster

| Name | Description | Type |
|-----------------------------|-------------|-----------------------------------|
| <i>pgBouncerIntegration</i> | | <i>PgBouncerIntegrationStatus</i> |

13.42 PoolerList

PoolerList contains a list of Pooler

| Name | Description | Type |
|--------------------|--|------------------------|
| <i>metadata</i> | | <i>metav1.ListMeta</i> |
| <code>items</code> | <ul style="list-style-type: none"> <i>mandatory</i> | <i>Pooler</i> |

13.43 PoolerSecrets

PoolerSecrets contains the versions of all the secrets used

| Name | Description | Type |
|-------------------------|--|-------------------------|
| <code>serverTLS</code> | The server TLS secret version | <i>SecretVersion</i> |
| <code>serverCA</code> | The server CA secret version | <i>SecretVersion</i> |
| <code>clientCA</code> | The client CA secret version | <i>SecretVersion</i> |
| <i>pgBouncerSecrets</i> | The version of the secrets used by PgBouncer | <i>PgBouncerSecrets</i> |

13.44 PoolerSpec

PoolerSpec defines the desired state of Pooler

| Name | Description | Type |
|------------------------|---|---|
| <code>cluster`</code> | This is the cluster reference on which the Pooler will work. Pooler name should never match with any cluster name within the same namespace. - <i>mandatory</i> | <i>LocalObjectReference</i> |
| <code>type`</code> | Which instances we must forward traffic to? - <i>mandatory</i> | PoolerType |
| <code>instances</code> | The number of replicas we want - <i>mandatory</i> | int32 |
| <code>template`</code> | The template of the Pod to be created | <i>PodTemplateSpec</i> |
| <code>pgbouncer</code> | The PgBouncer configuration - <i>mandatory</i> | <i>PgBouncerSpec</i> |

13.45 PoolerStatus

PoolerStatus defines the observed state of Pooler

| Name | Description | Type |
|------------------------|---|--------------------------------------|
| <code>secrets`</code> | The resource version of the config object | <i>PoolerSecrets</i> |
| <code>instances</code> | The number of pods trying to be scheduled | int32 |

13.46 PostgresConfiguration

PostgresConfiguration defines the PostgreSQL configuration

| Name | Description | Type |
|---------------------------------------|--|--|
| <code>parameters`</code> | PostgreSQL configuration options (postgresql.conf) | map[string]string |
| <code>pg_hba`</code> | PostgreSQL Host Based Authentication rules (lines to be appended to the pg_hba.conf file) | string |
| <code>epas`</code> | EDB Postgres Advanced Server specific configurations | <i>EPASConfiguration</i> |
| <code>promotionTimeout`</code> | Specifies the maximum number of seconds to wait when promoting an instance to primary. Default value is 40000000, greater than one year in seconds, big enough to simulate an infinite timeout | int32 |
| <code>shared_preload_libraries</code> | Lists of shared preload libraries to add to the default ones | string |
| <code>ldap`</code> | Options to specify LDAP configuration | <i>LDAPConfig</i> |

13.47 RecoveryTarget

RecoveryTarget allows to configure the moment where the recovery process will stop. All the target options except TargetTLI are mutually exclusive.

| Name | Description | Type |
|---------------------------|---|--------------|
| <code>`targetTLI`</code> | The target timeline (“latest” or a positive integer) | string |
| <code>`targetXID`</code> | The target transaction ID | string |
| <code>`targetName`</code> | The target name (to be previously created with <i>pg_create_restore_point</i>) | string |
| <code>`targetLSN`</code> | The target LSN (Log Sequence Number) | string |
| <code>`targetTime`</code> | The target time, in any unambiguous representation allowed by PostgreSQL | string |
| <i>targetImmediate</i> | End recovery as soon as a consistent state is reached | <i>*bool</i> |
| <code>`exclusive`</code> | Set the target to be exclusive (defaults to true) | <i>*bool</i> |

13.48 ReplicaClusterConfiguration

ReplicaClusterConfiguration encapsulates the configuration of a replica cluster

| Name | Description | Type |
|-----------------------|--|--------|
| <i>enabled</i> | If replica mode is enabled, this cluster will be a replica of an existing cluster. Replica cluster can be created from a recovery object store or via streaming through <i>pg_basebackup</i> . Refer to the Replication page of the documentation for more information. - <i>mandatory</i> | bool |
| <code>`source`</code> | The name of the external cluster which is the replication origin - <i>mandatory</i> | string |

13.49 RollingUpdateStatus

RollingUpdateStatus contains the information about an instance which is being updated

| Name | Description | Type |
|------------------|--|--------------------|
| <i>imageName</i> | The image which we put into the Pod - <i>mandatory</i> | string |
| <i>startedAt</i> | When the update has been started | <i>metav1.Time</i> |

13.50 S3Credentials

S3Credentials is the type for the credentials to be used to upload files to S3. It can be provided in two alternative ways:

- explicitly passing `accessKeyId` and `secretAccessKey`
- inheriting the role from the pod environment by setting `inheritFromIAMRole` to `true`

| Name | Description | Type |
|---------------------------------|---|--------------------------|
| <code>accessKeyId</code> | The reference to the access key id | <i>SecretKeySelector</i> |
| <code>secretAccessKey</code> | The reference to the secret access key | <i>SecretKeySelector</i> |
| <code>sessionToken</code> | The references to the session key | <i>SecretKeySelector</i> |
| <code>inheritFromIAMRole</code> | Use the role based authentication without providing explicitly the keys. - <i>mandatory</i> | bool |

13.51 ScheduledBackup

ScheduledBackup is the Schema for the scheduledbackups API

| Name | Description | Type |
|---------------------|---|------------------------------|
| <i>metadata</i> | | <i>metav1.ObjectMeta</i> |
| <code>spec</code> | Specification of the desired behavior of the ScheduledBackup. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | <i>ScheduledBackupSpec</i> |
| <code>status</code> | Most recently observed status of the ScheduledBackup. This data may not be up to date. Populated by the system. Read-only. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status | <i>ScheduledBackupStatus</i> |

13.52 ScheduledBackupList

ScheduledBackupList contains a list of ScheduledBackup

| Name | Description | Type |
|--------------------|--|------------------------|
| <i>metadata</i> | Standard list metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds | <i>metav1.ListMeta</i> |
| <code>items</code> | List of clusters - <i>mandatory</i> | <i>ScheduledBackup</i> |

13.53 ScheduledBackupSpec

ScheduledBackupSpec defines the desired state of ScheduledBackup

| Name | Description | Type |
|-------------------------|--|-----------------------------|
| <code>`suspend`</code> | If this backup is suspended or not | <i>*bool</i> |
| <i>immediate</i> | If the first backup has to be immediately start after creation or not | <i>*bool</i> |
| <code>`schedule`</code> | The schedule follows the same format used in Kubernetes CronJobs, see https://pkg.go.dev/github.com/robfig/cron#hdr-CRON_Expression_Format - <i>mandatory</i> | string |
| <code>`cluster`</code> | The cluster to backup | <i>LocalObjectReference</i> |

13.54 ScheduledBackupStatus

ScheduledBackupStatus defines the observed state of ScheduledBackup

| Name | Description | Type |
|------------------------------|--|--------------------|
| <code>`lastCheckTime`</code> | The latest time the schedule | <i>metav1.Time</i> |
| <i>lastScheduleTime</i> | Information when was the last time that backup was successfully scheduled. | <i>metav1.Time</i> |
| <i>nextScheduleTime</i> | Next time we will run a backup | <i>metav1.Time</i> |

13.55 SecretKeySelector

SecretKeySelector contains enough information to let you locate the key of a Secret

| Name | Description | Type |
|------------|--------------------------------------|--------|
| <i>key</i> | The key to select - <i>mandatory</i> | string |

13.56 SecretVersion

SecretVersion contains a secret name and its ResourceVersion

| Name | Description | Type |
|---------------------|-----------------------------------|--------|
| <code>`name`</code> | The name of the secret | string |
| <i>version</i> | The ResourceVersion of the secret | string |

13.57 SecretsResourceVersion

SecretsResourceVersion is the resource versions of the secrets managed by the operator

| Name | Description | Type |
|---------------------------------------|---|-------------------|
| <code>superuserSecretVersion</code> | The resource version of the “postgres” user secret | string |
| <code>replicationSecretVersion</code> | The resource version of the “streaming_replica” user secret | string |
| <code>applicationSecretVersion</code> | The resource version of the “app” user secret | string |
| <code>caSecretVersion</code> | Unused. Retained for compatibility with old versions. | string |
| <code>clientCaSecretVersion</code> | The resource version of the PostgreSQL client-side CA secret version | string |
| <code>serverCaSecretVersion</code> | The resource version of the PostgreSQL server-side CA secret version | string |
| <code>serverSecretVersion</code> | The resource version of the PostgreSQL server-side secret version | string |
| <code>barmanEndpointCA</code> | The resource version of the Barman Endpoint CA if provided | string |
| <code>metrics</code> | A map with the versions of all the secrets used to pass metrics. Map keys are the secret names, map values are the versions | map[string]string |

13.58 StorageConfiguration

StorageConfiguration is the configuration of the storage of the PostgreSQL instances

| Name | Description | Type |
|---------------------------------|---|--|
| <code>storageClass</code> | StorageClass to use for database data (<i>PGDATA</i>). Applied after evaluating the PVC template, if available. If not specified, generated PVCs will be satisfied by the default storage class | string |
| <code>size</code> | Size of the storage. Required if not already specified in the PVC template. Changes to this field are automatically reapplied to the created PVCs. Size cannot be decreased. - <i>mandatory</i> | string |
| <code>resizeInUseVolumes</code> | Resize existent PVCs, defaults to true | bool |
| <code>pvcTemplate</code> | Template to be used to generate the Persistent Volume Claim | corev1.PersistentVolumeClaimSpec |

13.59 WalBackupConfiguration

WalBackupConfiguration is the configuration of the backup of the WAL stream

| Name | Description | Type |
|--------------------|---|-----------------|
| <i>compression</i> | Compress a WAL file before sending it to the object store. Available options are empty string (no compression, default), <i>gzip</i> , <i>bzip2</i> or <i>snappy</i> . | CompressionType |
| <i>encryption</i> | Whenever to force the encryption of files (if the bucket is not already configured for that). Allowed options are empty string (use the bucket policy, default), <i>AES256</i> and <i>aws:kms</i> | EncryptionType |
| <i>maxParallel</i> | Number of WAL files to be either archived in parallel (when the PostgreSQL instance is archiving to a backup object store) or restored in parallel (when a PostgreSQL standby is fetching WAL files from a recovery object store). If not specified, WAL files will be processed one at a time. It accepts a positive integer as a value - with 1 being the minimum accepted value. | int |