EDB™ Postgres on Kubernetes

Release 2.7

Integration with Google Kubernetes Engine

Feb 04, 2020
EDB Postgres Platform for Containers allows you to use a docker-formatted container to deploy and manage EDB Postgres Advanced Server (Advanced Server) and supporting components in a Google Kubernetes Engine (GKE) environment. GKE automation provides an environment in which you can easily:

- Deploy or disable Advanced Server instances as needed.
- Automatically scale an Advanced Server instance to meet application requirements.
- Easily ensure Failover Manager protection for your data.
- Utilize load balancing to distribute read/write requests across available servers.
- Deploy Advanced Server instances with custom configurations in a container environment.

The EDB Postgres Platform for Containers automates the deployment of containers that include Advanced Server and the following supporting components:

- EDB Failover Manager (EFM)
- EDB Postgres Backup and Recovery Tool (BART)
- pgPool (connection pooling for Postgres databases)
- EDB Postgres Enterprise Manager (PEM)

For detailed information and documentation for each component, please visit the EnterpriseDB website.
The following changes are added to EDB™ Postgres integration with Google Kubernetes Engine guide to create version 2.7:

- Containers can be deployed in GKE using one of the two following methods:
  - Helm chart
  - EDB Postgres Advanced Server (EPAS) Operator (installed via helm chart)
  - Locale support for Advanced Server
For deployment using the Advanced Server Operator, refer to the Advanced Server Operator Guide.

- EDB Postgres Containers use the UBI (Universal Base Image) image from Red Hat as the base OS. For more information about UBI images, refer to https://developers.redhat.com/products/rhel/ubi/.
To deploy a container, you must have access to the registry in which the containers are stored (containers.enterprisedb.com). To receive credentials that allow you to access the container registry, please submit a request at: https://www.enterprisedb.com/general-inquiry-form. The following containers are available in the EDB container registry at containers.enterprisedb.com:

EDB Postgres Advanced Server with Failover Management and Monitoring components:
- containers.enterprisedb.com/edb/edb-as:v12
- containers.enterprisedb.com/edb/edb-as:v11
- containers.enterprisedb.com/edb/edb-as-gis:v11
- containers.enterprisedb.com/edb/edb-as-gis:v12
- containers.enterprisedb.com/edb/edb-as:v10
- containers.enterprisedb.com/edb/edb-as-gis:v10
- containers.enterprisedb.com/edb/edb-as-gis23:v10
- containers.enterprisedb.com/edb/edb-as:v9.6

EDB Postgres Advanced Server only:
- containers.enterprisedb.com/edb/edb-as-lite:v12
- containers.enterprisedb.com/edb/edb-as-lite:v11
- containers.enterprisedb.com/edb/edb-as-lite:v10

EDB Backup and Recovery Tool (BART):
- containers.enterprisedb.com/edb/edb-bart:v2.5

EDB Postgres Enterprise Manager (PEM):

EDB PgPool:
- containers.enterprisedb.com/edb/edb-pgpool:v4.0
Note: You can also download all of these images as tarballs from the EDB website.

The containers use UBI (Universal Base Image) image from Red Hat as the base OS. For more information about UBI images, refer to: https://developers.redhat.com/products/rhel/ubi/
Preparing to Deploy an Advanced Server Container

Google Kubernetes Engines (GKE) is an open source container application platform based on the Kubernetes container orchestrator for enterprise application development and deployment.

GKE supports multi-node Advanced Server clusters that include deployments of Advanced Server (with EDB Failover Manager), EDB BART, and pgPool.

For information about GKE, click here.
3.1 Deployment Prerequisites

For this release, containers will be deployed using helm charts in the Google Kubernetes Environment Engine (GKE). Before deploying the EDB Postgres Platform for Containers, you must have:

- A fully functional GKE environment with at least three worker nodes
- Install latest Helm Client
You can use a Helm chart to deploy EDB Postgres Advanced Server, EDB Postgres BART, EDB Postgres Enterprise Manager, and PgPool. The following sections provide details.

### 4.1 Deploying Containers with Advanced Server, BART, and PgPool

The following steps outline the deployment process for containers with Advanced Server, BART, and PgPool:

1. Visit the EnterpriseDB website to sign up for credentials that allow you to access the containers repository at:
   
   containers.enterprisedb.com
2. Download the following files:
   - Sample helm chart `edb-2.7.0.tgz`
   - Sample values.yaml file `sample-values.yaml`
3. Create your own values.yaml file with customized input values using the sample values.yaml as a template (e.g. `myvalues.yaml`).
4. (Recommended) Create a namespace. For information about creating namespaces, please see the Kubernetes documentation.
5. Create a registry secret using the container registry credentials received from EnterpriseDB. For information on how to create registry secrets, please see the Kubernetes documentation.
6. Enter the following command to deploy the containers (i.e. install the helm chart) with your custom inputs:
   `helm install edb-2.7.0.tgz -f myvalues.yaml`
   
   Where `myvalues.yaml` is the user supplied values.yaml file containing the desired parameter values.
7. Once the helm chart is successfully installed, the following EDB containers should be available in your environment:
• **EDB Postgres Advanced Server pod (includes EDB Failover Manager)**  
  Container image: edb-as:v12/edb-as:v11/edb-as:v10  
  Number of replicas: 3  
  Deployed as: Stateful Set  
  Each EPAS pod is deployed on a separate node (anti-affinity rule applied)

• **EDB PgPool pod**  
  Container image: edb-pgpool:v4.0  
  Number of replicas: 2  
  Deployed as: Replica Set  
  Each PgPool container is set on a separate node (anti-affinity rule applied)

• **(Optionally) EDB BART (Backup and Recovery Tool) pod**  
  Container image: edb-bart:v2.5  
  Number of replicas: 1  
  Deployed as: Replica Set


---

**4.2 Using a Helm Chart to Deploy a PEM Server**

Download the following sample files:

- Sample helm chart: edb-pemserver-7.12.0.tgz
- Sample values.yaml file: sample-values.yaml

Create your pemserver-values.yaml file with customized input values using the sample values.yaml file as a template (e.g. my-pemserver-values.yaml).

Enter the following command to deploy the containers with your custom inputs in your namespace:

```
helm install <helm-chart-name> edb-pemserver-7.12.0.tgz -f <pemserver-values.yaml> --namespace <namespace>
```

**Note:** In the example, myvalues.yaml is the user supplied values.yaml file containing the desired parameter values.
The following sections provide some use cases for EDB Containers.

5.1 Customizing Advanced Server Configuration Files

To provide custom `postgresql.conf` or `pg_hba.conf` settings for Advanced Server, perform the following steps before deploying the containers:

1. Create a directory, e.g. `initconf`:
   ```sh
mkdir initconf
   ```

2. In the `initconf` directory, store custom `postgresql.conf` and `pg_hba.conf` properties in files named `postgresql.conf.in` and `pg_hba.conf.in` respectively.

3. Create a configmap volume using the `initconf` directory:
   ```sh
   kubectl create configmap edb1-initconf-vol --from-file=initconf/
   where edb1 is the name of the database cluster in this example.
   ```

4. In the database section of the values yaml file, set the `initConfigmapVolume` property to the name the configmap volume created in the step above, i.e., `edb1-initconf-vol`. 
5.2 Providing Remote Access to a Container

Access to your database cluster is provided through a kubernetes service backed by pgPool containers which provide load balancing and connection pooling. To access the database remotely, expose pgPool via an external service:

```
kubectl expose deployment edb1-pgpool --port=5444 --target-port=9999 --name=edb1-ext-service --type=LoadBalancer
```

where `edb1` is the name of the database cluster and 5444 is the database port.

Once an external IP address is allocated for the server, you can obtain the service via the command:

```
kubectl get service edb1-ext-service
```

The database cluster (edb1) will now be accessible remotely using the external ip address and port.
5.3 Using EDB Failover Manager Functionality

For detailed information about Failover Manager (EFM), refer to the EFM 3.8 Guide.

- To enable EFM, ensure that the `enableHAMode` property is set to `Yes` in the `database` section of the values yaml file.

**Note:** EFM is enabled by default on the EPAS containers.

- Optionally, the pgPool pods can serve as witness nodes by setting the `enableHAMode` property in the `queryrouter` section of the values yaml file.

**Note:** EFM is disabled by default on pgPool containers.

- For EFM to function properly, the `replicas` property in the `database` section has to be at least 3 i.e., 1 master and 2 standbys.

- To customize EFM settings, create a file named `efm.properties.in` and include it in the `configmap` volume `edbl-initconf-vol` by placing it in the `initconf` directory prior to creating the `configmap` volume.
5.3.1 Performing a Failover

1. Run the following command to check the status of the database pods:
   ```
kubectl get pods -o wide -L cluster, role
   ```

   ![Fig. 5.1: Pods status](image)

   **Note:** The master node is `edb-as-0`.

2. Check status of the database cluster:
   ```
kubectl exec edb-as-0 -- efm cluster-status edb
   ```

   ![Cluster Status: edb](image)

   **Cluster Status: edb**

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Address</th>
<th>Agent DB</th>
<th>VIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standby</td>
<td>10.44.0.143</td>
<td>UP</td>
<td>UP</td>
</tr>
<tr>
<td>Witness</td>
<td>10.44.3.102</td>
<td>UP</td>
<td>N/A</td>
</tr>
<tr>
<td>Witness</td>
<td>10.44.4.117</td>
<td>UP</td>
<td>N/A</td>
</tr>
<tr>
<td>Promoting</td>
<td>10.44.7.121</td>
<td>UP</td>
<td>UP</td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.8.118</td>
<td>UP</td>
<td>UP</td>
</tr>
</tbody>
</table>

   **Allowed node host list:**
   10.44.8.141

   **Membership coordinator:** 10.44.3.102

   **Standby priority host list:**
   10.44.8.118 10.44.0.143

   **Promote Status:**

<table>
<thead>
<tr>
<th>DB Type</th>
<th>Address</th>
<th>WAL Received LSN</th>
<th>WAL Replayed LSN</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>10.44.7.121</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.8.118</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.8.143</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
<td>0/C0001A8</td>
</tr>
</tbody>
</table>

   **Standby database(s) in sync with master. It is safe to promote.**

   ![Fig. 5.2: View cluster](image)

3. Delete the master pod to simulate a crash which would initiate an automatic failover:
   ```
kubectl delete pod edb-as-0
   ```

   ![Fig. 5.3: Master pod is created again](image)
4. Re-check status of the database cluster to is the failover occurred:

   kubectl exec edb-as-0 -- efm cluster-status edb

**Note:** Node 10.48.1.97 (edb1-as-1) has transitioned from standby to master, and the newly created node 10.48.2.97 (edb-as-0) is the new standby.

---

**Fig. 5.4:** EFM database cluster

---

5.3. Using EDB Failover Manager Functionality
### 5.3.2 Performing a Switchover

Switchover manually forces a standby node to transition to master, and the master node transitions to become a standby node. Assuming the DB cluster state denoted in the section above, The current master is 10.44.8.118.

```
kubectl exec edb-as-0 -- efm promote edb -switchover
```

![Cluster Status: edb](image)

<table>
<thead>
<tr>
<th>Agent Type</th>
<th>Address</th>
<th>Agent</th>
<th>DB</th>
<th>VIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>10.44.0.141</td>
<td>UP</td>
<td>UP</td>
<td></td>
</tr>
<tr>
<td>Witness</td>
<td>10.44.3.102</td>
<td>UP</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Witness</td>
<td>10.44.4.117</td>
<td>UP</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.7.121</td>
<td>UP</td>
<td>UP</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.8.118</td>
<td>UP</td>
<td>UP</td>
<td></td>
</tr>
</tbody>
</table>

**Allowed node host list:**
10.44.0.141

**Membership coordinator:** 10.44.0.141

**Standby priority host list:**
10.44.8.118 10.44.7.121

**Promote Status:**

<table>
<thead>
<tr>
<th>DB Type</th>
<th>Address</th>
<th>WAL Received LSN</th>
<th>WAL Replayed LSN</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>10.44.0.141</td>
<td>0/7001C68</td>
<td>0/7001C68</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.8.118</td>
<td>0/7001C68</td>
<td>0/7001C68</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>10.44.7.121</td>
<td>0/7001C68</td>
<td>0/7001C68</td>
<td></td>
</tr>
</tbody>
</table>

Standby database(s) in sync with master. It is safe to promote.

**Fig. 5.5: View cluster details**

**Note:** The new master is 10.44.7.121 (edb-as-2) and the previous master 10.44.8.118 (edb-as-1) is now a standby node.
5.4 Using Backup and Recovery Functionality

For detailed information about the EDB Backup and Recovery Tool (BART), refer to the BART 2.5.2 Guide.

To enable backup (and recovery) using BART, ensure that the enabled property is set to `true` in the backup section of the values yaml file (the BART pod will only get deployed if backup is enabled).

5.4.1 Taking a Full Backup

To take a manual backup:

- Set the `automateBackup` property to `No`
- To take compressed backups, set the `compressBackup` property to `Yes`
- For manual backup, run the following command:
  
  `kubectl bart-podname exec -- bart backup -s edb1`

To take an automated backup:

- Set the `automateBackup` property to `Yes`
- To take compressed backups, set the property `compressBackup` to `Yes`
- Set the `backupSchedule` and `numBackupsToKeep` properties to the desired values for your system

To list the backups, run the following command:

```
kubectl bart-podname exec -- bart show-backups -s edb1
```

To get the root directory of the backup location, run the following command:

```
kubectl bart-podname exec -- getBackupLocation
```
5.4.2 Performing a Full Recovery from Backup

To perform a full recovery from backup, complete the following steps:

1. Determine full path of the backup file to restore from:
   • For uncompressed backups, the file name will have the pattern:
     `/edbbackup/..../base.tar`
   • For compressed backups, the file name will have the pattern:
     `/edbbackup/..../base.tar.gz`

2. Set the `restoreFile` property to the fully qualified path of the backup that you wish to restore from the yaml file, and deploy.
5.5 Performing a Point-in-time Recovery

To perform a point-in-time recovery (PITR) from backup:

1. List backups:
   
   ```
kubectl bart-podname exec -- bart show-backups -s edb1
   ```

2. Restore `pgdata` into restore directory up to desired timestamp.
   
   ```
kubectl bart-podname exec -- bart restore -s edb1 -i backupid -p /
edbbackup/namespace/restore/edb1/pgdata -g 'timestamp'
   ```

3. Set the property `restoreDir` to the restored `pgdata` directory in the yaml file and deploy.
6.1 Uploading Containers to Network Isolated Environments

If you want to deploy containers to a network isolated environment, perform the following steps:

1. Using a browser, download EDB Container images as tarballs (.tgz files) from the EDB website.
2. Internally transfer the image tarball to the isolated environment.
3. Load the image in the isolated environment:
   
   \texttt{docker load < image.tgz}

For more information, refer to the Docker Documentation.
CHAPTER 7

Conclusion

EDB™ Postgres Containers and Integration with Google Kubernetes Engine Copyright © 2007 - 2020 EnterpriseDB Corporation. All rights reserved.

EnterpriseDB® Corporation 34 Crosby Drive, Suite 201, Bedford, MA 01730, USA
T +1 781 357 3390 F +1 978 467 1307 E info@enterprisedb.com www.enterprisedb.com

• EnterpriseDB and Postgres Enterprise Manager are registered trademarks of EnterpriseDB Corporation. EDB and EDB Postgres are trademarks of EnterpriseDB Corporation. Oracle is a registered trademark of Oracle, Inc. Other trademarks may be trademarks of their respective owners.

• EDB designs, establishes coding best practices, reviews, and verifies input validation for the logon UI for EDB Postgres products where present. EDB follows the same approach for additional input components, however the nature of the product may require that it accepts freeform SQL, WMI or other strings to be entered and submitted by trusted users for which limited validation is possible. In such cases it is not possible to prevent users from entering incorrect or otherwise dangerous inputs.

• EDB reserves the right to add features to products that accept freeform SQL, WMI or other potentially dangerous inputs from authenticated, trusted users in the future, but will ensure all such features are designed and tested to ensure they provide the minimum possible risk, and where possible, require superuser or equivalent privileges.

• EDB does not warrant that we can or will anticipate all potential threats and therefore our process cannot fully guarantee that all potential vulnerabilities have been addressed or considered.
Index

C
Conclusion, 19
Customizing Advanced Server Configuration Files, 9

E
EDB Containers, 3

P
Performing a Failover, 12
Performing a Full Recovery from Backup, 16
Performing a Point-in-time Recovery, 17
Performing a Switchover, 14
Preparing to Deploy an Advanced Server Container, 5
Providing Remote Access to a Container, 10

R
Reference, 18

T
Taking a Full Backup, 15

U
Use Cases, 9
Using Backup and Recovery Functionality, 15
Using EDB Failover Manager Functionality, 11
Using Helm Charts to Deploy Containers, 7

W
What’s New, 2