Pivotal Greenplum©for Kubernetes: Demonstration of Managing Greenplum Database on Kubernetes

Massive Parallel Processing Relational Database in the Cloud

Jemish Patel Goutam Tadi Oz Basarir Lawrence Hamel David Sharp Fei Yang Xin Zhang
Pivotal Software

Abstract

Greenplum Database (GPDB) has many features designed to enable data scientists. Before a data scientist can use GPDB, a database administrator (DBA) must provision a cluster and install any required data science packages. Provisioning a GPDB cluster on bare metal requires a lengthy setup process. Scaling, recovering, and securing the cluster post-deployment are also complex. Greenplum for Kubernetes (GP4K) abstracts away these complexities, simplifying and automating the process for users.

In this demonstration, we introduce GP4K with an opinionated deployment and a declarative manifest. We provide a brief overview of GP4K's architecture and discuss its implementation. We also demonstrate a full life cycle of managing a cluster from birth to retirement, including scale-up and self-healing all with minimal DBA inputs, as shown in a video here. [1]

ACM Reference Format:
Jemish Patel Goutam Tadi Oz Basarir Lawrence Hamel David Sharp Fei Yang Xin Zhang. 2019. Pivotal Greenplum©for Kubernetes: Demonstration of Managing Greenplum Database on Kubernetes: Massive Parallel Processing Relational Database in the Cloud. In *2019 International Conference on Management of Data (SIGMOD '19), June 30–July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3299869.3320229

[1]https://bit.ly/2YPwhxp

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3320229

1 Introduction

1.1 Kubernetes

Kubernetes is a portable,extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.[1]. Kubernetes is becoming the de facto standard for deploying containerized applications at scale in a multi-cloud environment.

Distributed databases are an attractive class of stateful applications [5] to adopt K8s, as customers increasingly ask cloud providers for database solutions to analyze troves of application generated data. In particular, users of Greenplum Database, a distributed database based on PostgreSQL, stand to gain from leveraging K8s.

1.2 Greenplum Database

Greenplum Database (GPDB) is a massively parallel processing (MPP) database server based on a shared nothing architecture, enabling it to distribute the workload of multi-terabyte data warehouses and process a query in parallel utilizing all of a system's resources [3]. It is designed to serve large enterprise customers querying both managed data as well as data stored on external sources such as HDFS, AWS S3, Kafka, and Spark.

There are two types of nodes in Greenplum Database cluster, masters and segments [7]. The master handles all incoming connections, queries compilation, and coordinates execution across all the segments, while segments are individual PostgreSQL instances handling processing on distributed data.

2 Greenplum For Kubernetes

At Pivotal, we have developed GP4K, utilizing the K8s platform to help our users, such as data scientists, data engineers, and database administrators (DBAs), to address common challenges associated with deploying and managing distributed databases. GP4K aims to solve problems in areas of speed, scalability and stability, for the following scenarios:

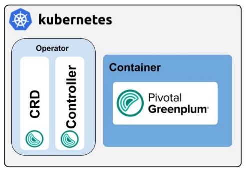

Figure 1: Greenplum for Kubernetes Architecture

(1) Instead of following pages of instructions to carefully configure a system, DBAs can deploy a cluster in minutes using pre-packaged and factory-tested images.
(2) Rather than following a painful process to recover from failures, GP4K enables automatic recovery. For example, when a pod dies, it is recreated automatically and attached to the same storage such that its state is preserved.
(3) In contrast to physically setting up additional hardware to scale the database as workload increases, GP4K leverages elastic infrastructure resources by detaching storage from its associated container and attaching it to a new container with more memory and CPU resources.

3 Related Work

Adapting an existing MPP system for cloud environments has been done at Vertica [9] in Eon mode. Also Snowflake [2] is designed to only run from the cloud as managed service.

GP4K is the first K8s project utilizing the operator pattern to manage the deployment and production life cycle of an open-source MPP distributed system.

4 Greenplum for Kubernetes Details

4.1 Architecture

GP4K consists of three components: a custom resource, an operator and a cluster instance. Greenplum cluster instances are deployed as custom resources to K8s clusters and managed by Greenplum Operator, as illustrated in Figure 1.

`GreenplumCluster` *custom resource:* We use the CustomResourceDefinition (CRD) API exposed by K8s to define how a K8s user may create a Greenplum cluster instance. The custom resource defines all of the parameters the Greenplum Operator needs to create a Greenplum cluster instance, such as number of primary segments, and limits for CPU and memory. The user can then construct a YAML or JSON document to specify the desired state of the Greenplum clusters and submit it to the K8s API with kubectl. All user input parameters are validated by the Greenplum Operator.

`greenplum-operator` *pod:* Running in its own K8s pod, the Greenplum Operator runs the custom `GreenplumCluster` controller that continually monitors `Greenplum Cluster` resources and takes action to attain the user's desired state. Once a `Greenplum Cluster` resource is created, the Greenplum Operator is then responsible for maintaining the state of the Greenplum cluster according to the properties defined. If a `GreenplumCluster` resource is modified, the Greenplum Operator reacts to those changes by updating the underlying Greenplum cluster instance, such as by adding or removing pods. However the controller may also reject requested changes that it cannot make to the Greenplum cluster and report an error. If the underlying Greenplum cluster instance has unexpected changes (e.g. a pod dying), then the operator is responsible for reconciling the cluster with the specifications in the resource (e.g. by restarting the pod and ensuring it will rejoin the cluster instance).

`GreenplumCluster` *instance:* This includes master, standby, and segments, in separate K8s pods that contain the Greenplum Database software, along with additional, optional Greenplum components, such as the MADlib open-source library for scalable in-database analytics [6]. The `Greenplum Cluster` instance also exposes a load balanced K8s service for psql access on port 5432 and sets up SSH keys for intra-cluster communication and management.

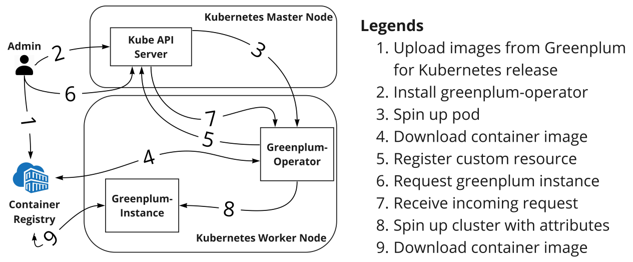
Kubernetes Master Node
Admin
Kube API Server
Greenplum-Operator
Container Registry
Greenplum-Instance
Kubernetes Worker Node
1
2
3
4
5
6
7
8
9
Legends
1. Upload images from Greenplum for Kubernetes release
2. Install greenplum-operator
3. Spin up pod
4. Download container image
5. Register custom resource
6. Request greenplum instance
7. Receive incoming request
8. Spin up cluster with attributes
9. Download container image

Figure 2: DBA Workflow using Greenplum Operator

Figure 2 shows a DBA's workflow to deploy Greenplum Operator and request a specific Greenplum instance.

4.2 Implementation

Kubernetes resources, such as `StatefulSets`, `ConfigMaps`, `Secrets`, `PersistentVolumeClaims` and `PersistentVolumes`, are used in implementing the system to resolve the challenges associated with deploying and maintaining distributed Greenplum server components as a single Greenplum cluster instance. Figure 3 illustrates the implementation details of GP4K

Persistent Volumes Claims (PVCs) and Persistent Volumes (PVs) : Greenplum uses built-in K8s storage classes that enable provisioning volumes in a dynamic fashion, when a Greenplum segment is spun up for the first time. Persistent Volume Claims specify the amount of storage, and the class

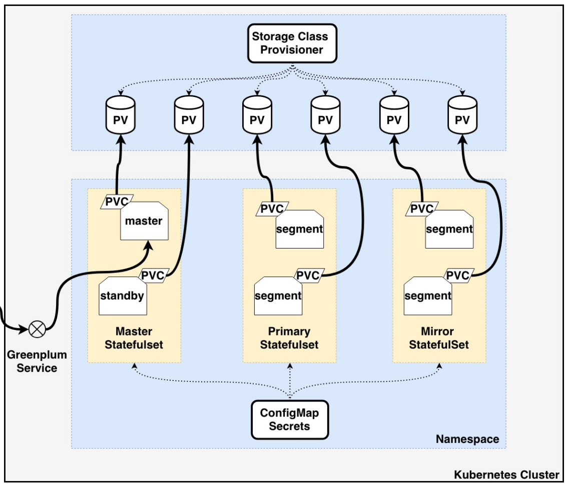

Figure 3: Greenplum for Kubernetes Implementation

of storage, needed by each segment. Containers can come and go within the cluster, and any re-created container will remount any persistent disk that was previously created for the same container, as long as the cluster lives. This automated provisioning of storage is very flexible.

StatefulSets : A `StatefulSet` is a Kubernetes workload API object used to manage stateful application. GP4K uses three StatefulSets: one for master database instances, one for primaries, and one for mirror segment databases. The master StatefulSet is configured with two replicas: active master and standby master. The segment StatefulSets are created with the same number of replicas by the `GreenplumCluster` controller. The associated primary and mirror of a given segment share the same index digit. For example, a primary/mirror segment pair would consist of the pods `segment-a-1` and `segment-b-1`.

5 Demonstration Plan

This demonstration illustrates how using GP4K simplifies the handling of multiple scenarios in a data warehouse management life cycle from birth to retirement with a consistent experience regardless of private or public infrastructure.

5.1 Birth: Deployment

After a database administrator (DBA) uploads the Greenplum Operator and Greenplum Instance images to an accessible container registry, the DBA declares the container image registry to the Greenplum Operator using the operator's Helm chart. For example, if the DBA put the images in a `gcr.io` container image registry, then the entries in the operator-values-overrides.yaml would look like below:

```
operatorImageRepository: |
  gcr.io/some-project-name/greenplum-operator
greenplumImageRepository: |
  gcr.io/some-project-name/greenplum-for-kubernetes
```

Install Greenplum Operator as shown below:

```
helm install greenplum-operator/operator \
  --name greenplum-operator \
  -f operator-values-overrides.yaml
kubectl create -f my-gp-instance.yaml
```

GP4K validates user input for correctness and rejects requests if the request is incorrect

```
# recreate Greenplum with same name/properties
kubectl create -f workspace/my-gp-instance.yaml
> Error from server ... "my-greenplum" already exists
```

5.2 Grow Bigger: Scale-Out

As the total amount of data grows, it can saturate a given segment's capacity as increasing segment's memory or cpu is no longer cost-effective. Therefore, increasing the capacity of the Greenplum cluster can only be done by adding more segments. Currently, increasing the number of segments will increase the computing capacity as well as the storage capacity. In the future, we plan to support scaling storage and compute independently using dynamic expansion of persistent volume a native K8s feature [4]. The following scenario demonstrates scaling out a Greenplum cluster by increasing segment counts.

```
apiVersion: "greenplum.pivotal.io/v1"
kind: "GreenplumCluster"
metadata:
  name: my-greenplum
spec:
 segments:
    primarySegmentCount: 16 -> 24
```

From: an initial deployment of 16 segments **To**: scaling out the cluster to 24 segments

5.3 Grow Stronger: Self-Healing

In order to provide an highly available service in a distributed database system, GP4K relies on K8s self-healing capabilities to keep the service running despite failures in different parts of the system.

5.3.1 Primary Pod Failure If a primary segment crashes for any reason, Greenplum fails over to a mirror segment automatically. Greenplum `gprecoverseg` utility can be used to recover the failed segment once it is brought back online. However bringing it back online is a lengthy and costly process [8]. This scenario demonstrate how the failed pod is automatically re-created, allowing a DBA to run `gprecoverseg -a` to reactivate the failed segment instance and identifies the changed database files that require re-synchronization.

```
psql -c "begin; select pg_sleep(300); end"
kubectl delete pod segment-a-0
kubectl exec -it master-0 bash -- -c \
  "source /opt/gpdb/greenplum_path.sh; \
  gprecoverseg -a"
```

Furthermore, because the primary and mirror segments have the same guaranteed resources, there is no need to rebalance the segments.

5.3.2 Master Pod Failure Failure of the master host directly causes service interruptions, which do not necessarily occur during segment failures. GP4K enables rapid service restoration by running `gpstart` when the failed master is re-created automatically, leading to low recovery time as demonstrated in the following scenario.

```
psql -c "begin; select pg_sleep(300); end"
kubectl delete pod master-0
kubectl exec -it master-0 bash -- -c \
  "source /opt/gpdb/greenplum_path.sh; gpstart -a"
```

5.3.3 Master Data Corruption For example, one of the most laborious failure scenarios occurs when the master fails. Without a K8s environment, a DBA will need to initiate a series of actions: 1) activate the standby master to become the active master, 2) update all client connections to that new active master, 3) and finally reactivate the original master as a standby. Unsurprisingly, this is one of the most costly and painful processes for DBAs to handle.

Support to automate this process is on the near-term GP4K roadmap. Reactivating the master as a new standby becomes a comparatively straightforward task. Furthermore, the clients do not need to change their connections as master and standby sit behind a load balanced service. The scenario below demonstrates the streamlined master to standby failover process without losing any data.

```
# start a transaction
psql -c "begin; select pg_sleep(300); end"
# Simulate a failure: delete the master and its data
kubectl delete pod master-0
kubectl delete pvc my-greenplum-master-0
# activate standby master
kubectl exec -it master-1 bash -- \
    -c "source /opt/gpdb/greenplum_path.sh; \
    gpactivatestandby -d /greenplum/data-1  -f"
# patch load balancer service
kubectl patch service greenplum -p \
     '{"spec":{"selector": \
     {"statefulset.kubernetes.io/pod-name": "master-1"}}}'
# initialize new standby
kubectl exec master-1  -- /bin/bash -c \
    "source /opt/gpdb/greenplum_path.sh; \
    /home/gpadmin/tools/sshKeyScan"
kubectl exec master-1  -- /bin/bash -c \
    "source /opt/gpdb/greenplum_path.sh; \
    gpinitstandby -a -s master-0"
```

5.4 Retire: Decommission

Once a cluster is no longer needed, a single command as shown below releases the compute resources from the cluster back to K8s.

```
kubectl delete -f my-gp-instance.yaml
```

Because, by default, any existing data is not deleted to prevent data loss, a separate command has to be run to cleanup Persistent Volumes.

```
kubectl delete pvc -l "app=greenplum"
```

6 Conclusion

As we have seen, the DBA experience when running GP4K is simplified by abstracting away bare-metal aspects associated with activities including deploying, scaling, healing and even decommissioning. The orchestration and automation capabilities provided by Greenplum Operator leveraging K8s free DBAs from the physical aspects of performing and managing the scenarios we have demonstrated. GP4K enpowers DBAs to focus on the needs of database users rather than deployment details.

Acknowledgments

The authors would like to thank Venkatesh Raghavan and Ashwin Agrawal for generously offering their time to review this paper and providing valuable comments and helpful suggestions.

References

[1] The Kubernetes Authors. 2018. What is Kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[2] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 215–226. https://doi.org/10.1145/2882903.2903741

[3] David DeWitt and Jim Gray. 1992. Parallel Database Systems: The Future of High Performance Database Systems. *Commun. ACM* 35, 6 (June 1992), 85–98. https://doi.org/10.1145/129888.129894

[4] Hemant Kumar (Red Hat). 2018. Resizing Persistent Volumes using Kubernetes. https://kubernetes.io/blog/2018/07/12/resizing-persistent-volumes-using-kubernetes/

[5] Cockroach Labs. 2018. https://www.cockroachlabs.com/docs/stable/orchestrate-cockroachdb-with-kubernetes.html

[6] Apache MADlib. 2018. http://madlib.apache.org/docs/latest/index.html

[7] Pivotal Software. 2018. https://gpdb.docs.pivotal.io/5130/admin_guide/intro/arch_overview.html

[8] Pivotal Software. 2018. https://gpdb.docs.pivotal.io/590/utility_guide/admin_utilities/gprecoverseg.html

[9] Ben Vandiver, Shreya Prasad, Pratibha Rana, Eden Zik, Amin Saeidi, Pratyush Parimal, Styliani Pantela, and Jaimin Dave. 2018. Eon Mode: Bringing the Vertica Columnar Database to the Cloud. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 797–809. https://doi.org/10.1145/3183713.3196938