Data Protection for Kubernetes

Application-aware Backups with Bareos

Heike Jurzik, Andreas Rogge

Data Protection for Kubernetes

Application-aware Backups with Bareos

Heike Jurzik, Andreas Rogge
Bareos GmbH & Co. KG

September 29, 2022

This technical whitepaper shows how to back up stateful Kubernetes (K8s) applications with Bareos. The Bareos File Daemon (FD), i.e. the client, runs as sidecar container alongside a main container, where the actual application is deployed to. The File Daemon is configured for client-initiated connections only, to back up the application's data.

Contents

1 Introduction 2

2 Stateful Applications in K8s 2
2.1 Sidecar Containers 3

3 Bareos Client in a K8s Pod 3
3.1 Creating the Sidecar Image 3
3.2 Deploying the Image to K8s 4
3.3 Configuring the Bareos Director . 4

4 Conclusion 5

5 About Bareos 5

6 About Kubernetes 5

1 Introduction

Continuously, companies and organizations drive and transform enterprise application delivery. Some have moved their applications to containers, others have transformed existing applications into microservices or developed new, cloud-native applications.

When it comes to data protection and backups, those containerized apps are not that different from traditional applications running on virtual machines or bare metal: the backup solution should consider both an application's state and its data.

This whitepaper introduces a backup concept for Kubernetes deployments, creating file-based backups of applications running in multiple containers and pods. It makes use of Bareos' feature for client-initiated backups and offers the following benefits:

- Protection of business-critical containerized applications
- Quick restore and easy re-deployment of application data
- Flexible approach, easy to migrate within and across the cloud
- Consistent backup concept with certified Open Source software

2 Stateful Applications in K8s

A typical Kubernetes setup includes many components, for example, containers, pods, services, certificates, secrets, etc. While there are backup concepts available which handle the Kubernetes objects and their configuration itself, this paper offers a different approach: an application-aware backup secures the current state of applications at the time of the backup. This can include data in memory and also pending transactions.

In our example setup[1], we have deployed a typical WordPress blog with a MySQL database to Kubernetes. The blog software runs in one container, the database backend in another one. To make it even more interesting, those two containers belong to different pods.

The goal is to create a file-based backup of the WordPress installation as well as a logical backup of the database. The challenge: it's a closed setup and the two pods and their containers can't be accessed from the outside. The solution: Bareos supports client-initiated connections, so the File Daemon runs as a sidecar container, using the same settings as WordPress for accessing the MySQL database.

2.1 Sidecar Containers

In Kubernetes, a pod is a group of one or more containers. A sidecar container is a utility container in a pod which is linked to a main container. As a result, the sidecar and the primary application container share their resources, for example, the pod storage and network interfaces. Containers in the same pod can also share the storage volumes.

Sidecar containers are mainly used to extend the main containers' functionality—without having to change their codebase. In our example setup, the pod with the WordPress installation has a sidecar container which runs the Bareos File Daemon. The sidecar has access to the same persistent volume as the WordPress container. Thus, it can back up and restore the volume's contents. Additionally, the sidecar container includes the MySQL credentials, so it can back up and restore the database.

3 Bareos Client in a K8s Pod

Basically, three steps are required to implement the application-aware backup with Bareos:

1. Build a container image which runs the Bareos File Daemon and the MySQL client.
2. Deploy the sidecar image to Kubernetes.
3. Configure the Bareos Director and set up the new client.

[1] https://github.com/bareos/kube-bareos

3.1 Creating the Sidecar Image

First, we're going to build a container image which runs the Bareos File Daemon and the MySQL client. The image can be customized using environment variables at deployment time. The directory `container-image`[2] in our GitHub repository contains a Docker file for the initial setup:

```
FROM docker.io/almalinux:8
RUN curl -o /etc/yum.repos.d/bareos.\
repo -O https://download.bareos.org\
/bareos/release/21/EL_8/bareos.repo\
 && dnf install -y \
  bareos-filedaemon \
  mysql \
 && dnf clean all
COPY *.* /
ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "-f" ]
```

It creates an image based on Alma Linux which runs the Bareos File Daemon as well as the MySQL client. The `entrypoint.sh` script reads a Bareos configuration template (`bareos-fd.conf.in`) and a MySQL template (`mysql-defaults.cnf.in`) and replaces a few variables when deploying the container to Kubernetes. This is necessary, because you have to configure the MySQL client and the File Daemon somehow—logging into the container and adjusting the settings in a text editor is not possible.

Basically, the Bareos FD needs these 4 settings:

1. Name of the Bareos Director
2. Password of the Bareos Director
3. Address of the Bareos Director
4. Name of the File Daemon (the client)

For the MySQL client, the template sets the database backend's hostname, the username and the password.

Strictly speaking, for our example setup the script is a little exaggerated. It explains how to automate things, though: assuming you are

[2] https://github.com/bareos/kube-bareos/tree/master/container-image

running 10, 20, or more WordPress installations in Kubernetes, then you would have to configure 10, 20, or more different File Daemons manually. Automating the setup process with a script like this is a lot faster and more efficient as well as less prone to errors.

3.2 Deploying the Image to K8s

Next, the application with its sidecar image is deployed to Kubernetes. In our GitHub repository, the `k8s` directory offers some Kubernetes *kustomization* files (in YAML format) that you can adjust to your own environment:

- `kustomization.yaml`: Set the password for the MySQL client and the Bareos Director, include other resources.
- `wordpress.yaml`: Specify the WordPress container, i.e. the container image, the FD's name and the address of the Bareos Director as well as its password (defined in `kustomization.yaml`). Also set the database host and its credentials. `volumeMounts` specifies the document root (`/var/www/html`) which is used in the WordPress container; the backup container uses the same directory.
- `mysql.yaml`: Configure the MySQL container, for example, the image name, the MySQL credentials (defined in `kustomization.yaml`), the persistent volume (database in `/var/lib/mysql`), etc.
- `ingress.yaml`: The Ingress resource; Kubernetes Ingress is an API object that provides routing rules to manage external users' access to the services in a Kubernetes cluster, typically via HTTPS/HTTP.[3]

After you have deployed the application, the last step is to configure the Bareos Director to set up the new client, the FileSet and the backup job itself.

3.3 Configuring the Bareos Director

In order to set up the new File Daemon for your Bareos installation, you need three configuration files.[4] The `wordpress-fd.conf` file contains the client configuration and enables the client-initiated connection.

Using this feature (which has been available since Bareos 16.2.2), the Bareos Director doesn't have to know how to access the pod with the WordPress and the sidecar containers. If the pod starts, the file daemon will contact the Bareos Director itself:

```
Client {
  Name = wordpress-fd
  Password = "BareosFDPassword"

  # configure client-initiated \
connections only:
  Connection From Client To \
Director = yes
  Connection From Director \
To Client = no

  # address is required, but \
will not be used
  Address = localhost
}
```

The FileSet resource `wordpress-set.conf` contains the document root for the WordPress installation and calls the Bareos `bpipe` plugin[5] to stream the database dump to Bareos for backup:

```
FileSet {
  Name = "wordpress-set"
  Include {
    Options {
      Signature = MD5
    }
    File = /var/www/html
    Plugin = "bpipe:file=/MYSQL/\
all.sql:reader=mysqldump \
--all-databases:writer=mysql"
  }
}
```

[3] https://kubernetes.io/docs/concepts/services-networking/ingress/

[4] https://github.com/bareos/kube-bareos/tree/master/bareos-dir.d

[5] https://docs.bareos.org/TasksAndConcepts/Plugins.html#bpipe-plugin

The actual backup job is defined in the file `wordpress.conf`:

```
Job {
  Name = "wordpress"
  JobDefs = "DefaultJob"
  FileSet = "wordpress-set"
  Client = "wordpress-fd"
}
```

Reload the Bareos configuration and check with the `bconsole` command `status schedule` if the new backup job appears in the schedule. The scheduler itself gets updated every minute.

Your WordPress instance and its associated database will now be backed up according to the schedule you define (here: schedule for the `DefaultJob` JobDefs Resource).

4 Conclusion

When it comes to creating file-based backups, containerized apps are not that different from traditional applications running on virtual machines or bare metal. The backup software should be able to handle the application's state and its data. Following the application-aware approach, this paper offers a concept to back up Kubernetes applications running in multiple containers and pods: a stateful WordPress application with two persistence backends, i.e. the filesystem and the database backend.

During the restore process both the files as well as the database (to be more precise: its current state!) have to be restored. Even if the entire Kubernetes cluster drops out, it's easy to restore the setup with this approach: after deploying all applications to Kubernetes, Bareos is able to restore the WordPress files and the MySQL database—this takes only a few minutes.

5 About Bareos

Bareos[6] (**B**ackup **Ar**chiving **Re**covery **O**pen **S**ourced) is a cross-network open source backup solution which preserves, archives, and recovers data from all major operating systems. The Bareos project started in 2010 and is being developed under the AGPLv3 license. The company Bareos GmbH & Co. KG and their partners offer professional subscription and support services, so that customers can rely on a maintained backup environment.

Worldwide, organizations across almost all sectors use Bareos. Customers include public authorities and government departments, small and medium-sized enterprises as well as companies listed on the DAX and Fortune 500, e.g. telecommunications, cloud and internet service providers, the media, education, energy, finance, automotive and aerospace industries.

6 About Kubernetes

Kubernetes[7] is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

The name Kubernetes originates from Greek, meaning "helmsman" or "pilot". K8s as an abbreviation results from counting the eight letters between the **K** and the **s**. Google open-sourced the Kubernetes project in 2014. Kubernetes combines over 15 years of Google's experience running production workloads at scale with best-of-breed ideas and practices from the community.

[6] https://www.bareos.com/
[7] https://kubernetes.io/