Kubernetes 集群资源数据的备份、恢复和自动化

1、概述

1.1 Kubernetes 是什么？

“Kubernetes（常简称为 K8s）是用于自动部署、扩展和管理容器化（containerized）应用程序的开源系统。该系统由 Google 设计并捐赠给 Cloud Native Computing Foundation（今属 Linux 基金会）来使用。”这是维基百科上对 Kubernetes 的介绍。

“它将组成应用程序的容器组合成逻辑单元，以便于管理和服务发现，Kubernetes 构建在 Google 15 年生产环境经验基础之上,并结合来自社区的最佳创意和实践。”这是 Kubernetes 官网中文版写在扉页上的寄语。

简单来说，Kubernetes 能帮用户把应用封装在容器里，想让它运行多少个副本就运行多少个。你的应用也许服务于 1 万个客户，部署 10 个应用容器，就能满足需求。也许你的应用服务于 1000 万客户，可能部署 1000 个应用容器才能满足需求。Kubernetes 能帮你实现业务规模的自动水平伸缩。当然，Kubernetes 能做的远不止这些。

Kubernetes 对云计算的重要性，不亚于二十年前 Java 语言随第一波互联网浪潮掀起的革命性风暴。Java 语言能让程序员的源代码“编写一次，运行在任何地方。”而 Kubernetes 引领的云计算革命，能让应用程序经过编排、调度组合，组成收放自如的复杂计算机应用系统。就像一台全自动绣花机，它把一根根线头、一个个色块、一个个印象元素组合在一起，编织出绚丽的彩色图案。

某年双十一前，某宝的高管豪言，系统已经就绪，只需一杯清茶，再大的访问流量（N 亿级用户），运维团队也能应对。难道他们没有用到可伸缩的容器编排？难以想象没有容器云他能如此豪言。

1.2 为什么备份？

备份的目的是防止数据永久丢失，快速恢复系统至正常可用。数据为什么会丢失呢？用户手潮误操作、不完美的版本升级的软件故障，单点失效的硬件故障，地震、台风、洪水、

海啸等自然灾害，都可能影响系统正常运行，造成数据部分丢失甚至全部灭失。

Kubernetes 也是软件，软件运行的中间状态和运行结果也是以资源数据形式保存的。Kubernetes 的运行数据保存在 etcd 存储服务，etcd 是以键值对 Key/Value 格式存储，并对外提供数据读和数据写服务。

Etcd 一般以多点集群形式出现，发生单点失效的可能性很小，但是用户误操作、升级失败的可能性还是存在的。

1.3 备份方法介绍

数据备份大致可以分为两类：物理备份和逻辑备份。

物理备份不区分数据的内在逻辑关系，把数据存储作为一个整体来备份，恢复时也是作为整体恢复，不可能只恢复一部分数据。

逻辑备份按照数据的内在逻辑关系，选择性提取部分数据或全部数据，恢复时可以选择恢复一部分数据。

数据的物理备份，一般来说对人类是不可读的，只有原软件系统才能读取识别。Oracle、MySQL 的 Dump 文件、重做日志和数据快照，etcd、Redis 数据文件的直接拷贝或者数据快照，都可以看做是物理备份。

从物理备份恢复数据的优点是明显的，要恢复完整数据非常快。缺点也很明显，可能会丢失上次备份以来的数据更新，但是可以通过全量备份配合更频繁的增量备份来弥补，缩小数据丢失的范围。物理备份也不能做部分恢复，那怕只是一小点数据更新。

从逻辑备份恢复数据的优点也是明显的，恢复部分数据很快，想要恢复哪部分就恢复哪部分。缺点是，逻辑备份的速度慢一些，遇到数据量大时耗时相对较长。

物理备份与逻辑备份的优缺点正好相反，在实践中不妨结合起来使用，扬长补短，发挥各自的优势。物理备份适合系统崩溃后的快速恢复重建，而逻辑备份适合更加精细化的局部修补。

本小文要介绍的 Kubernetes 数据备份是一种逻辑备份方法，需要深入到 Kubernetes 内部的逻辑结构，条分缕析，探寻奥秘，类似于庖丁解牛，尝试找到一种逻辑备份新途径。

2、模型

2.1 Kubernetes 命名空间和资源

Kubernetes 集群能支撑庞大而又复杂的应用系统，许多用户和团队共享集群，难免会相互影响或冲突。为了避免用户之间的冲突，Kubernetes 引入命名空间概念， 在同一个命名空间下各种资源的不能重名，在不同的命名空间下允许重名，实现资源安全隔离。用户在分配给他的命名空间下操作，不用担心影响到别人，也不用担心受别人影响，因为每个用户或者每个团队都有独立的命名空间。

Kubernetes 拥有和保留系统级的命名空间 Kube-system 和 kube-public，未经授权不允许普通用户使用系统命名空间。默认命名空间 default 是公共的，如果没有指定命名空间，用户新建的资源都将建立在 default 命名空间。

Kubernetes 的资源类型包括：

- ✓ 服务 service
- ✓ 部署 deploy
- ✓ 配置 configmap
- ✓ 加密配置 secret
- ✓ 任务 job
- ✓ 定时任务 cronjob
- ✓ 副本集 replicaset
- ✓ 驻留任务集 daemonset
- ✓ 有状态集 statefulset

此处不一一列举。

Kubernetes 的资源配置告诉 Kubernetes 系统，部署（deploy）哪些应用，对外提供哪些服务（service），应用配置参数（configmap）存在哪儿，敏感参数（secret）需要加密吗，运行一次就结束的任务（job）如何调度，像闹钟一样的定时任务（cronjob）怎么安排，一个节点运行一个且只运行一个的驻留任务（daemonset）支持吗，运行过后希望保留数据及状态的任务（statefulset）。资源配置赋予 Kubernetes 丰富的资源创建能力，能适应复杂多变的应用环境。

同一个资源类型下，可以配置多个不同的资源实例，而同一个资源实例允许运行多个完全相同的副本，拓展了系统服务能力。例如：供客户使用的 nginx/http 服务和供内部运营人员使用 nginx/http 服务，可以分别配置、分别部署。供客户使用的 nginx/http 服务可以多副本运行，并且按应用负载自适应扩展或者缩减副本数量。

2.2 Kubernetes 集群层级模型

如上所述，Kubernetes 系统可以归纳为多层级的分层模型，从上到下分别是：

- ✓ Kubernete 平台
- ✓ 命名空间 namespace
- ✓ 资源类型 resource type
- ✓ 资源实例 resource instance。

上一级和下一级之间是一对多的关系，而下一级从属于唯一的上级。例如，资源类型为 deploy 的应用实例 nginx-web 从属于类型 deploy，而 deploy 从属于某个命名空间 ns-cmft。命名空间可以容纳 deploy、service 和 job 等多种类型的资源。资源实例的副本（实例），是运行时的概念，可以动态创建、动态销毁。

Kubernetes 资源配置构成容器编排的身体骨架，而血肉在运行时填充进去并塑造形状。或者说，资源配置就像是一个具体 Kubernetes 运行时实例的 DNA、基因组。如果所有资源配置确定了，系统的内部结构也基本确定了。这个 Kubernetes 基因组的另一部分是容器镜像，以及容器运行时产生的数据。

2.2 数据备份模型

Kubernetes 的资源配置可以从系统内导出来，存为 yaml 格式的文本文件，也可以根据导出来的 yaml 文件，重新建立与导出时完全一样的同名资源。这是 Kubernetes 备份和恢复的核心原理和技术基础，也是本文描述的基本处理过程。

前文说到 Kubernetes 资源是分层模型，从上到下是一对多的关系。那我们从上到下，按图索骥、顺藤摸瓜，一级一级往下探索，就能找到所有的命名空间、资源类型、资源实例，达成全面备份资源配置的目的。Kubectl 是 Kubernetes 提供给管理用户使用的实用命令，它能查询命名空间、资源实例等，资源类型和资源名称是固定的，作为查询参数传入。kubectl 命令正好能满足我们的需求。

查询全部命名空间

```
# kubectl get ns
## 查询某类型下的资源列表
# kubectl -n mynamespace get deploy
## 查询具体资源的详细配置数据，以 yaml 格式导出为文本文件：
# kubectl -n mynamespace get deploy nginx-web -o yaml > deploy_nginx-web.yaml
## 从 yaml 文件重建资源实例
# kubectl -n mynamespace create -f deploy_nginx-web.yaml
```

代码块中的 mynamespace 和 nginx-web 需要替换为具体命名空间和资源名称。

2.3 数据恢复模型

数据恢复是从已备份的数据副本恢复到正在运行的 Kubernetes 系统。

这里的数据副本是指此前备份导出来的 yaml 格式文本文件。Yaml 文件是人类肉眼可读的，也是可以修改的。如果管理员想恢复某个历史时点备份的资源文件，且修改某个错误值，那么从备份的 yaml 文件恢复是比较好的选择。

数据恢复可以是全量恢复，或者部分恢复。全量恢复是从某个时刻的全量备份数据恢复至 Kubernetes 系统。部分恢复只恢复部分数据。

相比数据备份的豪气、大方，数据恢复则要小心谨慎得多。数据恢复一般以部分恢复为主，只对发生故障，而且确认备份副本数据正确有效时才会恢复。在系统发生不可逆转的全面崩溃时，会优先考虑从物理备份恢复，只有在物理备份不可用时，此时才考虑逻辑备份（本文所说的 yaml 备份）。

有时物理备份与逻辑备份配合使用恢复系统也是不错的选项。

部分恢复应当把经过仔细审查通过的 yaml 文件复制到专门的恢复目录（restore），以便于按顺序批量执行，恢复系统数据。

数据恢复应当记录详细的日志，以便事后查询、审计。

3、数据备份

3.1 全量备份

备份脚本默认是全量备份，也就是备份 K8s 集群下的所有命名空间下的系统资源数据。有些生产系统，命名空间比较多，运行的服务、容器和 Pod 也多，系统资源的数据量比较大，备份时间也会比较长。

3.2 优先备份

优先备份是指选择一部分优先级高的重要资源数据、或者经常变化的资源数据，并提供频率更高的备份。

同一用户下的资源数据经常配置在相同的命名空间，按命名空间来识别高优先级的资源数据是一条简便途径。本文的方法能用脚本传入参数，只备份指定命名空间的资源数据，而不会备份其他命名空间下的资源数据，可以用作优先备份。

3.3 备份流程和代码详解

备份流程的主体是三重嵌套的循环，从外到内分别是：命名空间、资源类型、同类型下的资源条目。具体步骤如下：

S1、查询集群内的全部命名空间，作为待备份的命名空间：

```
ns_list=`kubectl get ns | awk '{print $1}' | grep -v NAME`
```

如果脚本带有命令行参数，默认会把参数当做待备份的命名空间集合。

S2、对所有待备份的命名空间作循环处理。

S3、对一个待备份的命名空间下的所有资源类型作循环处理。资源类型是常量集合，以后还会补充新的资源类型：

```
CONFIG_TYPE="service deploy configmap secret job cronjob replicaset daemonset statefulset"
```

S4、查询一个资源类型下的所有资源条目，并对资源条目作循环处理：

```
item_list=`kubectl -n $ns get $type | awk '{print $1}' | grep -v NAME  | grep -v "No "`
```

S5、将一个资源条目的详细资源数据导出为 yaml 文本格式，输出到本地目录。备份文件的路径和文件名符合一定的规则，便于查询和恢复数据。

路径按备份时间，同一批备份的文件放在同一目录下，文件名以“命名空间_资源类型_条目名称.yaml”格式命名，以名度义，比较直观。

```
file_name=$BACKUP_PATH_DATA/${ns}_${type}_${item}.yaml
kubectl -n $ns get $type $item -o yaml > $file_name
```

S6、返回 S4 直到处理完某一类型下的所有资源条目。

S7、返回 S3 直到处理完某一命名空间下的所有资源类型。

S8、返回 S2 直到处理完所有待备份的命名空间。

公共处理：

1、在外中内三层嵌套循环分别加计数器，统计执行备份数量。在运行中输出状态、计数值和错误信息，便于观察备份进度。

2、同时保存运行时信息到日志文件。对自动化备份任务便于事后了解备份情况，例如什么时间备份了哪些资源项，备份是否成功等等。

3.4 数据备份源代码清单

```
#!/usr/bin/bash


## show usage
echo "usage: $0 [namespace]"


## define variable
BACKUP_PATH=/data/k8s-backup-restore
BACKUP_PATH_BIN=$BACKUP_PATH/bin
BACKUP_PATH_DATA=$BACKUP_PATH/data/backup/`date +%Y%m%d%H%M%S`
BACKUP_PATH_LOG=$BACKUP_PATH/log
BACKUP_LOG_FILE=$BACKUP_PATH_LOG/k8s-backup.log


## set K8s type
CONFIG_TYPE="service deploy configmap secret job cronjob replicaset daemonset statefulset"


## make dir
mkdir -p $BACKUP_PATH_BIN
mkdir -p $BACKUP_PATH_DATA
mkdir -p $BACKUP_PATH_LOG
```

```
cd $BACKUP_PATH_DATA


## set namespace list
ns_list=`kubectl get ns | awk '{print $1}' | grep -v NAME`
if [ $# -ge 1 ]; then
        ns_list="$@"
fi


## define counters
COUNT0=0
COUNT1=0
COUNT2=0
COUNT3=0


## print hint
echo "`date` Backup kubernetes config in [namespaces: ${ns_list}] now."
echo "`date` Backup kubernetes config for [type: ${CONFIG_TYPE}]."
echo "`date` If you want to read the record of backup, please input command ' tail -100f
${BACKUP_LOG_FILE} '"


## ask and answer
message="This will backup resources of kubernetes cluster to yaml files."
echo ${message} 2>&1 >> $BACKUP_LOG_FILE
echo ${message}
read -n 1 -p "Do you want to continue? [yes/no] " input_char && printf "\n"
if [ "${input_char}" != 'y'     ]; then
      message="`date` Exit by user's selection."
      echo $message 2>&1 >> $BACKUP_LOG_FILE
      echo $message
```

```
    exit 1
fi


## loop for namespaces
for ns in $ns_list; do
    COUNT0=`expr $COUNT0 + 1`
    echo "`date` Backup No.${COUNT0} namespace [namespace: ${ns}]." 2>&1 >> $BACKUP_LOG_FILE
    COUNT2=0
    ## loop for types
    for type in $CONFIG_TYPE; do
            echo "`date` Backup type [namespace: ${ns}, type: ${type}]." 2>&1 >> $BACKUP_LOG_FILE
        item_list=`kubectl -n $ns get $type | awk '{print $1}' | grep -v NAME  | grep -v "No "`
        COUNT1=0
        ## loop for items
        for item in $item_list; do
            file_name=$BACKUP_PATH_DATA/${ns}_${type}_${item}.yaml
              echo "`date` Backup kubernetes config yaml [namespace: ${ns}, type: ${type}, item: ${item}] to file: ${file_name}" 2>&1 >> $BACKUP_LOG_FILE
            kubectl -n $ns get $type $item -o yaml > $file_name
            COUNT1=`expr $COUNT1 + 1`
            COUNT2=`expr $COUNT2 + 1`
            COUNT3=`expr $COUNT3 + 1`
            echo "`date` Backup No.$COUNT3 file done."  2>&1 >> $BACKUP_LOG_FILE
        done;
    done;
    echo "`date` Backup ${COUNT2} files done in [namespace: ${ns}]." 2>&1 >> $BACKUP_LOG_FILE
```

```
done;


## show stats
message="`date` Backup ${COUNT3} yaml files in all."
echo ${message}
echo ${message} 2>&1 >> $BACKUP_LOG_FILE
echo "`date` kubernetes Backup completed, all done." 2>&1 >> $BACKUP_LOG_FILE
exit 0
```

3.5 手动备份

在命令行下输入备份脚本，默认开始全量备份：

cd k8s-backup-restore/bin

./k8s_backup.sh

在命令行输入带参数的备份脚本，开始部分备份（优先备份）：

cd k8s-backup-restore/bin

./k8s_backup.sh wisecloud-controller wisecloud-agent

3.6 自动备份

备份工作最好加入定时任务，定时由系统自动触发备份。

将备份脚本加入到 Linux/Unix 的 crontab 定时任务：

crontab -e

输入定时任务计划，每天凌晨 0:05 触发自动备份运行一次。

5 0 * * * sh -x /data/k8s-backup-restore/k8s_backup.sh

3.7 备份机

备份机可以是 K8s 集群内的任意一台主机，也可以是集群外的主机。只要网络互通，备份机能访问到 K8s 集群的 Kube-apiserver，备份机可以部署在任何地方。备份机需要持有 K8s 集群的数字证书，也就是 kubeconfig 文件，否则无法访问 K8s 集群，也就不能执行任何备份操作。

3.8 备份的备份

备份数据需不需要备份，要不要考虑备份数据的安全性。备份数据的备份是管理员要考虑的事项之一。如果要提高备份数据的安全性，可以拷贝一份备份数据，存放到安全的地方。

4、数据恢复

4.1 准备数据

数据恢复会对 K8s 集群正常运行产生影响，需要谨慎执行。数据恢复没有采用全面恢复的策略，而是设置专用的数据恢复目录，只有恢复目录下的 yaml 文件才会被恢复。

从备份目录复制 yaml 文件到恢复目录，文件名应替换为待恢复的文件。

```
cd    k8s-backup-restore/data/backup/20190812xxyy
cp    wisecloud-controller_comfigmap_wisecloud-config.yaml    ../../restore/
```

4.2 执行恢复

数据恢复以批处理方式进行，对数据恢复专用目录下的所有 yaml 文件逐个执行，先删除旧资源条目，再创建新的资源条目。数据恢复的次序是按照 yaml 文件名升序先后排列。

```
cd    k8s-backup-restore/bin/
./k8s_restore.sh
```

4.3 恢复检测

数据恢复以后，最好再查询一次已恢复的资源项，验证资源项是否正确恢复了。检测的方式可以是重新做一次备份，导出 yaml 文件，再做数据比对。命令 diff 比较两个文本文件内容，并列出两个文件的差异行。

如果不想导出太多 yaml 文件，也可以导出单个资源项的 yaml 文件，然后与之前的文件做比对。例如：

```
kubectl -n wisecloud-controller get cm wisecloud-config -o yaml
```

4.4 恢复流程和代码详解

恢复流程的主要工作是：列出恢复目录下的 yaml 文件，按顺序逐个执行删除、创建资源项的工作。具体步骤如下：

S1、查询恢复目录下的 yaml 文件列表，按文件名排序，查询文件个数。

```
file_list=`ls -n ${BACKUP_PATH_DATA}/*.yaml | awk '{print $9}'`
```

file_count=`echo ${file_list} | wc -w`

S2、向用户提示潜在风险，询问是否继续恢复？

S3、对 yaml 文件名集合作循环处理。

S4、删除旧的资源项：

kubectl delete -f ${file_yaml}

S5、创建新的资源项：

kubectl create -f ${file_yaml}

S6、返回 S3 直到处理完所有的 yaml 文件。

公共处理：

1、在循环中设置计数器，统计执行恢复的文件数量。在运行中输出状态、计数值和错误信息，便于观察恢复进度。

2、同时保存日志信息到日志文件。

4.5 数据恢复源代码清单

```
#!/usr/bin/bash

## define variable
BACKUP_PATH=/data/k8s-backup-restore
BACKUP_PATH_BIN=$BACKUP_PATH/bin
BACKUP_PATH_DATA=$BACKUP_PATH/data/restore
BACKUP_PATH_LOG=$BACKUP_PATH/log
RESTORE_LOG_FILE=$BACKUP_PATH_LOG/k8s-restore.log

## make dir
mkdir -p $BACKUP_PATH_BIN
mkdir -p $BACKUP_PATH_DATA
mkdir -p $BACKUP_PATH_LOG
cd $BACKUP_PATH_DATA

## print hint
message="`date` Kubernetes Restore start now. All yaml files which located in path ${BACKUP_PATH_DATA} will be applied."
echo ${message} 2>&1 >> $RESTORE_LOG_FILE
echo ${message}
echo "`date` If you want to read the record of restore, please input command ' tail -100f ${RESTORE_LOG_FILE} '"
```

```
## list yaml files
file_list=`ls -n ${BACKUP_PATH_DATA}/*.yaml | awk '{print $9}'`
file_count=`echo ${file_list} | wc -w`

## ask and answer
message="WARNING!!! This will create ${file_count} yaml files to kubernetes cluster. While same name resources will be deleted. Please consider it carefully!"
echo ${message} 2>&1 >> $RESTORE_LOG_FILE
echo ${message}
read -n 1 -p "Do you want to continue? [yes/no/show] " input_char && printf "\n"
if [ "${input_char}" == 's'    ]; then
    message="`date` Show yaml files list."
    echo $message 2>&1 >> $RESTORE_LOG_FILE
    echo $message
    ls -n ${BACKUP_PATH_DATA}/*.yaml
    exit 1
elif [ "${input_char}" != 'y'    ]; then
    message="`date` Exit by user's selection."
    echo $message 2>&1 >> $RESTORE_LOG_FILE
    echo $message
    exit 2
fi

## loop for file list
COUNT=0
for file_yaml in $file_list; do
      COUNT=`expr $COUNT + 1`
      echo "`date` Restore No.${COUNT} yaml file: ${file_yaml}..." 2>&1 >> $RESTORE_LOG_FILE
      cmd_delete="kubectl delete -f ${file_yaml}"
      cmd_create="kubectl create -f ${file_yaml}"

      ## run delete
      echo "`date` Run shell: ${cmd_delete}." 2>&1 >> $RESTORE_LOG_FILE
      ${cmd_delete}
      result="failed"
      if [ $? -eq 0 ]; then
            result="ok"
      fi
      echo "`date` Delete resource ${result}." 2>&1 >> $RESTORE_LOG_FILE

      ## run create
      echo "`date` Run shell: ${cmd_create}." 2>&1 >> $RESTORE_LOG_FILE
      ${cmd_create}
      result="failed"
```

```
    if [ $? -eq 0 ]; then
        result="ok"
    fi
    echo "`date` Create resource ${result}." 2>&1 >> $RESTORE_LOG_FILE
done;

## show stats
message="`date` Restore ${COUNT} yaml files in all."
echo ${message}
echo ${message} 2>&1 >> $RESTORE_LOG_FILE
echo "`date` Kubernetes Restore completed, all done." 2>&1 >> $RESTORE_LOG_FILE
exit 0;
```

5、应用

5.1 故障恢复

系统出现故障后，恢复被破坏或者丢失的资源数据，这是备份与恢复的基本功能。

5.2 历史跟踪

在一个动态运行的 K8s 集群中，变化是常态，如果能连续记录各个时间点的资源数据快照，对于资源配置历史变化的跟踪溯源，有一定意义。

5.3 时点比对

在一些重要时间节点前、后，例如版本升级、部署新应用等，分别做资源数据备份，事后比较两个同名备份文件的内容，可以找出二者的差异，便于故障定位，或者判断变化是否符合预期。

diff datetime1/wisecloud-controller_comfigmap_wisecloud-config.yaml datetime2/wisecloud-controller_comfigmap_wisecloud-config.yaml

6、结语

6.1 工作小结

本文介绍了利用 kubectl 命令开发 K8s 资源数据备份、恢复程序的原理和实现方法。本

文的备份方法是逻辑方法，已备份的数据文件人类肉眼可读、可识别、可理解，支持全量备份和部分备份（优先备份）。自动化备份有利于降低运维工作量，节约人力成本。

本文介绍的方法已在某大型国有金融企业的容器云管理平台实际应用。

6.2 未来展望

对于大型企业、事业单位或者政府部门，一个 K8s 集群难以满足实际需要，一般会部署多个生产集群。能否只部署一套 K8s 备份程序同时支持多个 K8s 集群，是下一步改进的方向。集中备份有利于集中管理，节约 IT 管理成本，降低管理复杂度。

6.3 源码下载

本项目源代码托管在 github.com 代码仓库，欢迎下载使用。

https://github.com/solomonxu/k8s-backup-restore