# Conquering Data Consistency

## With Kasten K10 and Kanister.io Blueprints

January 2024

# Conquering Data Consistency with Kasten K10 by Veeam and Kanister.io Blueprints

This tutorial is a guide for a good development methodology with Kanister.io blueprints. By understanding Kasten K10 and Kanister.io integration, blueprint authors can overcome many application orchestration challenges for data management. In this tutorial, the Elasticsearch operator is used as an example to manage backup and replication.

There are many tutorial steps that are easy to understand, but they should be followed sequentially.

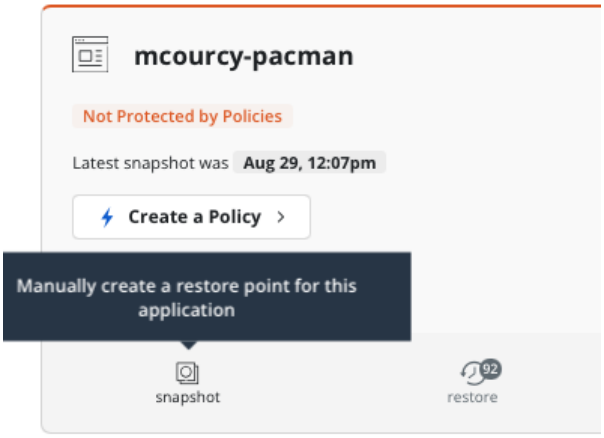
```
White highlights identify changes in a script or a command.
```

## Disclaimer

Kasten does not officially support this tutorial blueprint for backing up an Elasticsearch cluster with Elasticsearch operator.

## Can’t we just click the “snapshot” button?

Kasten K10 makes backups as simple as possible. In practice, the only thing you have to do is find your namespace in the UI and click “snapshot.”




Kasten K10 takes care of everything –- all the data and metadata in the namespace are “snapshotted” and exported to an off-site location for future restoration. Therefore, many databases, such as Postgres, MySQL or MongoDB that write transactions to a log file before committing them to support crash recovery are mostly compatible with this approach.

But a common workload such as Elasticsearch that documentation that states backing up an Elasticsearch cluster with snapshots is not supported:

> *A copy of the data directories of a cluster’s nodes does not work as a backup, because it is not a consistent representation of their contents at a single point in time. You cannot fix this by shutting down nodes while making the copies, nor by taking atomic filesystem-level snapshots, because Elasticsearch has consistency requirements that span the whole cluster. You must use the built-in snapshot functionality for cluster backups.*
>
> *If you try to restore a cluster from such a backup, it may fail with reports of corruption or missing files, or other data inconsistencies. Alternatively, it may appear to have succeeded but silently lost some of your data.*

For these reasons, we must extend Kasten K10 to support this workload. **Enter Kanister.**

# Kanister in 5 Minutes

Kanister is a data protection workflow management tool that lets you abstract away tedious details around executing data operations on Kubernetes, such a backup and restore.

There are three concepts that make up the Kanister framework:

- **Execution**: Controller, Blueprint and ActionSet
- **Profile**: Where you push or pull data
- **Artifacts**: What your operations produce and consume

## Execution: Blueprint and Actionset

Using programming language analogies, you can think of a **Blueprint** as a library of **Functions,** and an **ActionSet** as a **Call** to one of the library functions in the blueprint.

Execute Kanister by creating an Actionset. In the ActionSet, define the:

- Blueprint action(s) you want to execute
- Object(s) the blueprint targets
- Profile(s) you want to use

The Kanister controller detects the creation of an ActionSet and launches the execution.

Blueprint and ActionSet are Kubernetes objects that you will create.

## A Quick Example

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   phases:
   - func: KubeTask
     name: multielasticdump
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: alpine:latest
       command:
       - /bin/sh
       - -c
       - |
         echo "Number of elastic nodes : {{ (index .Object.spec.nodeSets 0).count }}"
```

This simple blueprint has one action (backup), containing one phase (multielasticdump), which is a KubeTask launched on the object namespace (*e.g.:* we create a pod for executing the task. Remember Kanister is deployed on the same Kubernetes cluster)..

The object you target can be dynamically used as a go-template parameter. For instance, you can render the namespace of the object with `{{ .Object.metadata.namespace }}` or even a more complex go-template expression will give you the size of the Elasticsearch cluster: `{{ (index .Object.spec.nodeSets 0).count }}`

To execute this blueprint, create an ActionSet that targets the Elasticsearch object with the blueprint name and action name in the blueprint.

```
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
 generateName: backup-
 namespace: kasten-io
spec:
 actions:
 - blueprint: elastic-bp
   name: backup
   object:
     apiVersion: v1
     group: elasticsearch.k8s.elastic.co/
     kind: ""
     name: quickstart
     namespace: test-es1
     resource: elasticsearches
```

You can create multiple actions in the ActionSet, and you can coordinate multiple actions across multiple blueprints, but this is beyond the scope of this tutorial.

It is easy to confuse actions in a blueprint and actions in an ActionSet, but they are not the same. Actions in an ActionSet are for executing multiple blueprints in a specific order. Most of the time an ActionSet has only one action. Actions in a blueprint are a set of consistent operations with each other, *e.g.:* backup with restore or preHook with postHook.

## Profiles

One of the strengths of Kanister is its ability to switch from one profile type to another. You can decide to stop backing up to an AWS S3 bucket and start backing up in an Azure container without changing a single line of code. All you need to do is change the profile name in the

ActionSet.

Today, Kanister has 3 types of profiles:

1. Azure
2. GCP
3. S3 compliant object storage.

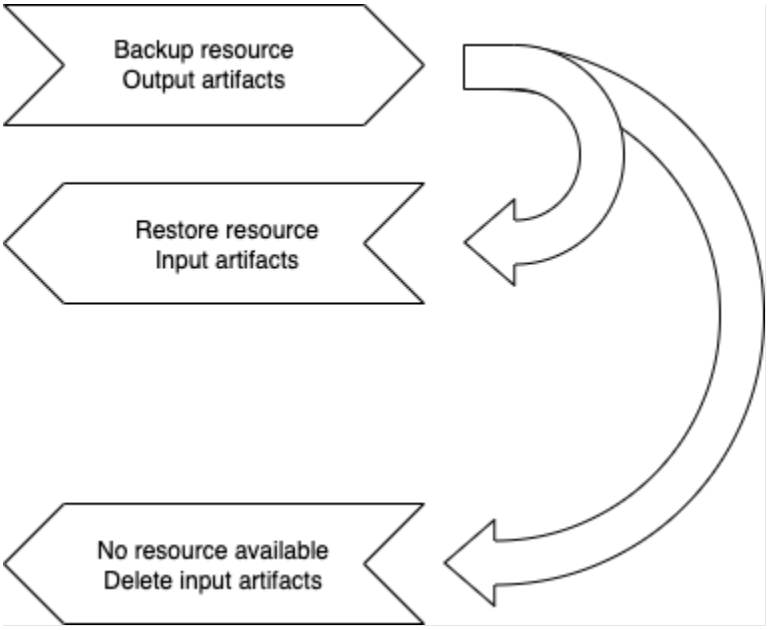
Kasten K10 adds two extra profile types: NFS and VBR.

## Producing and Consuming Artifacts

The next important concept is the Artifact. An artifact points to the physical backup. Examples of artifacts include:

- A path on a Profile
- A backup date
- A Kopia snapshot identifier
- A Postgres wall-e timeline number
- A Cockroach db backup identifier
- An AWS RDS snapshot URN
- Anything that can participate in the coordinate of a backup

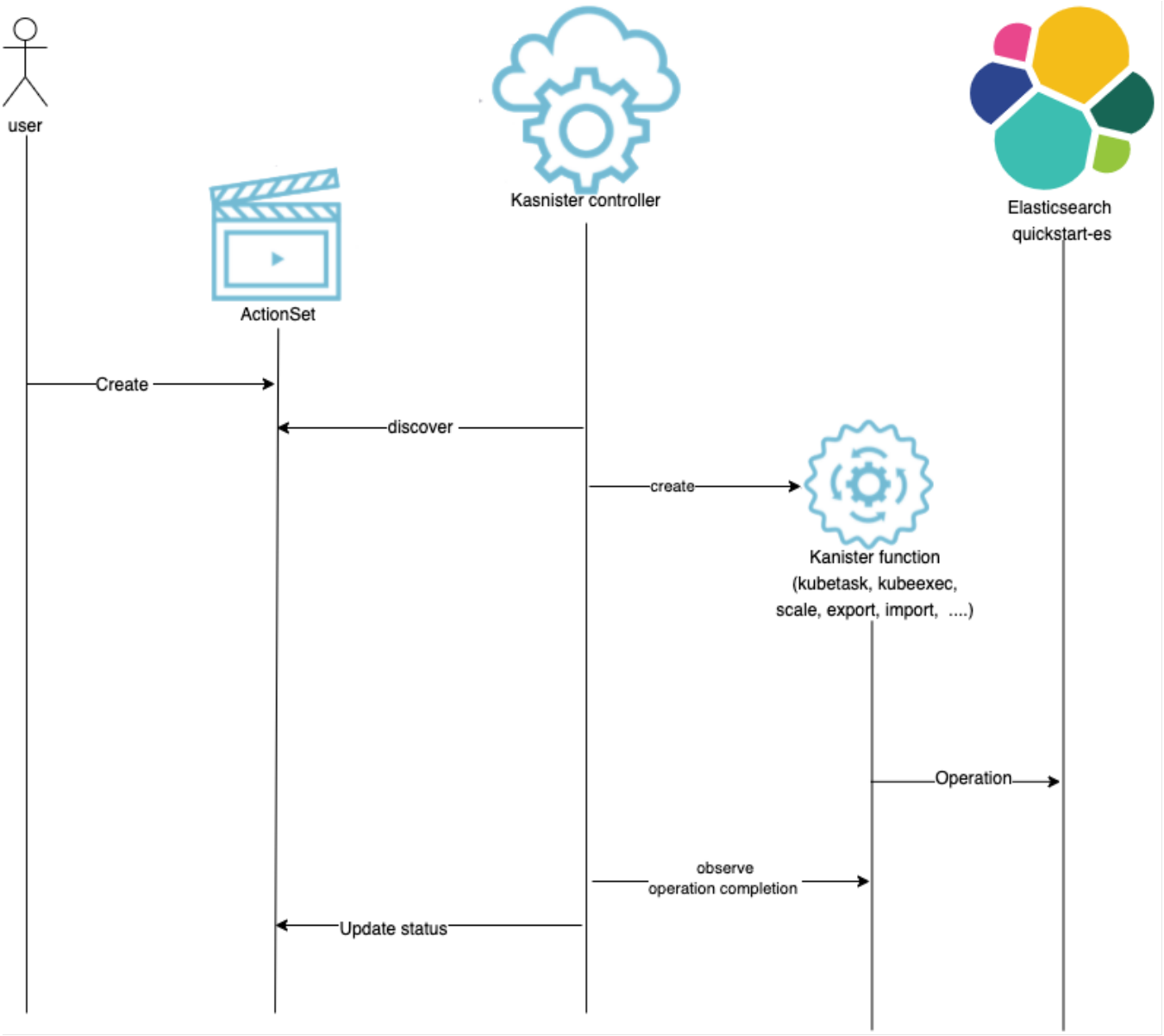
The artifact will be output in the status of the ActionSet and can be reused by another ActionSet as an input artifact. For instance, to execute a restore, you would consume the artifact produced by a backup.




Similarly, deleting a backup to reclaim space would require the backup artifact to coordinate to remove.

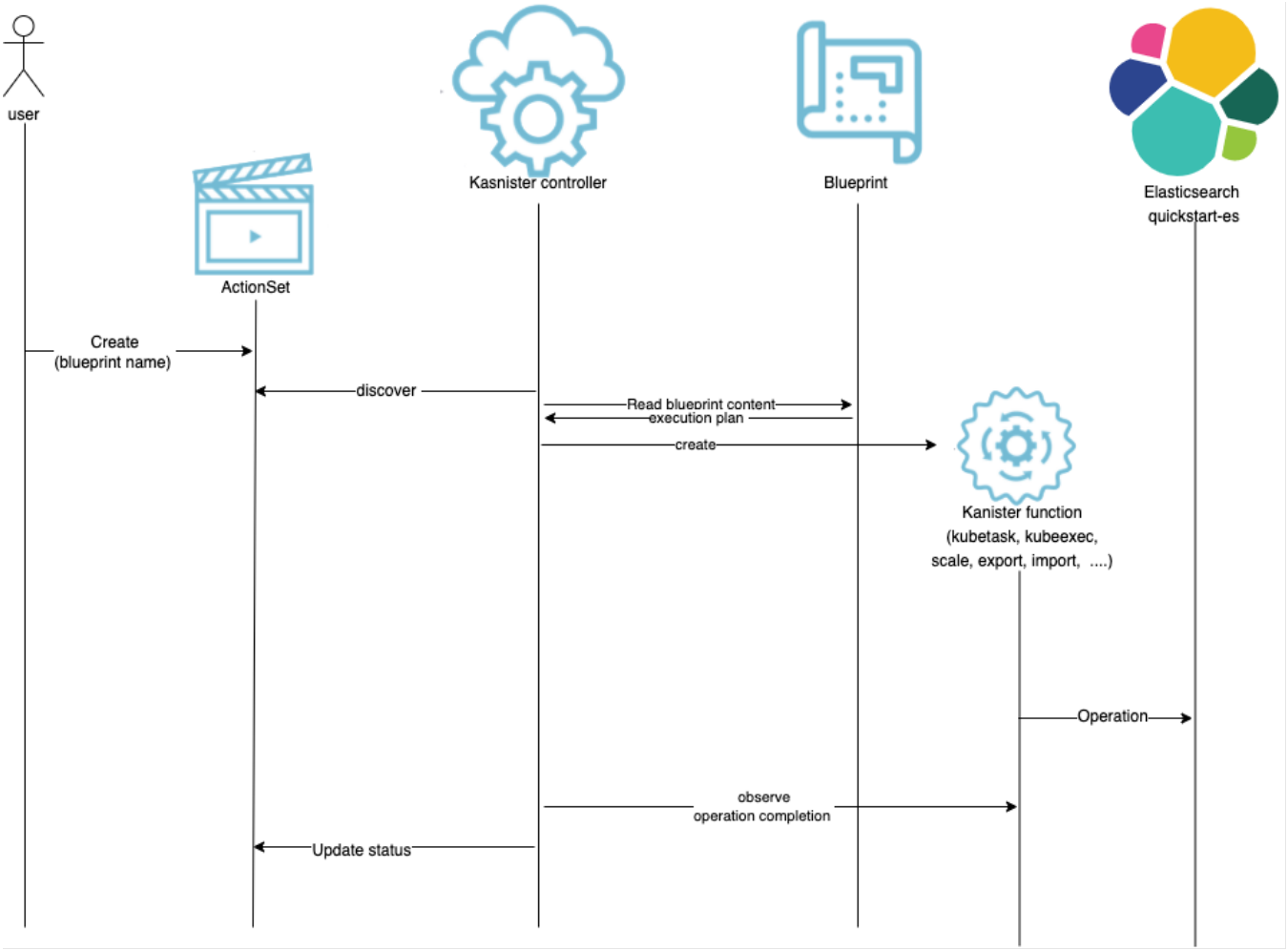
## Diagram Sequence Summary

1. In this simplified sequence, the blueprint and artifact are not shown:

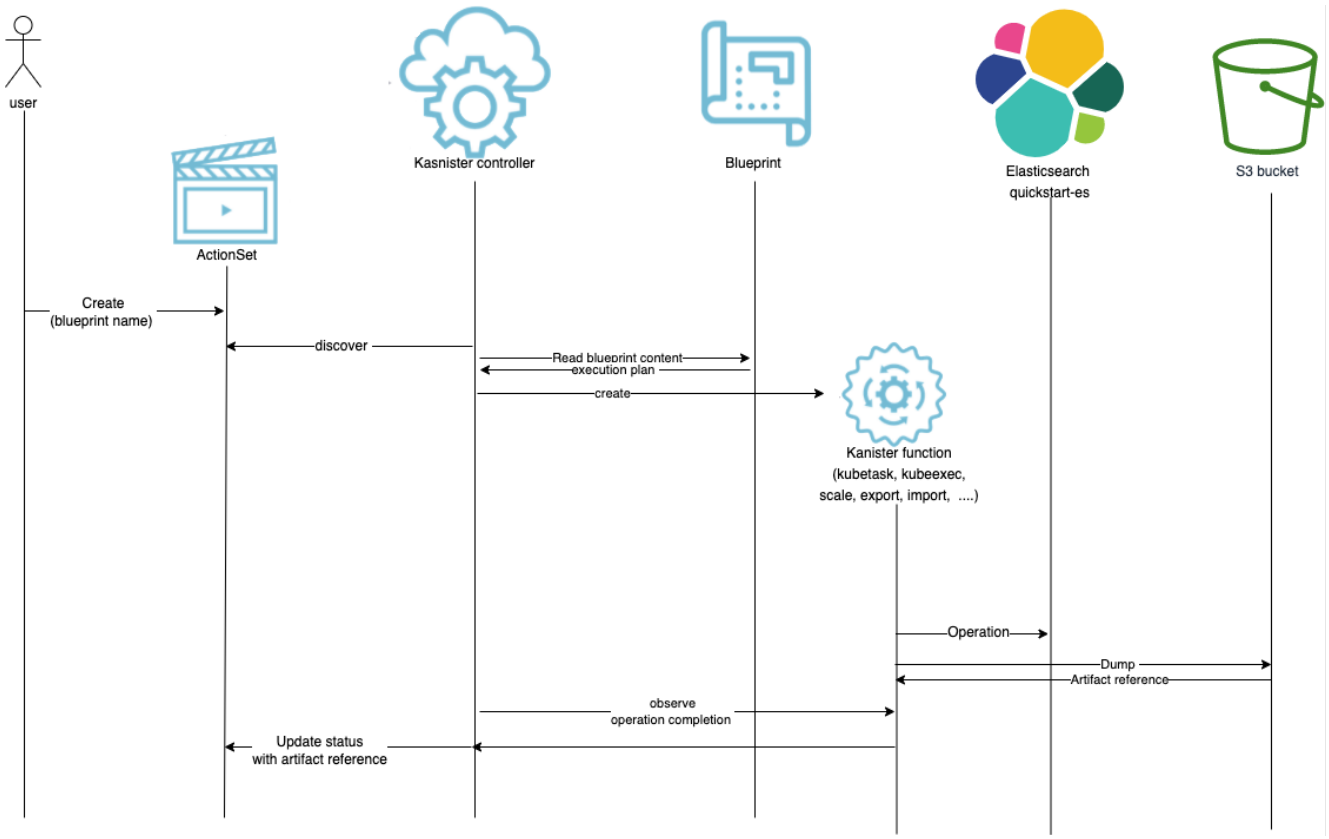



The user creates an ActionSet that will be handled by the Kanister Controller. The Kanister Controller will create operations on the workload, thanks to the Kanister Functions defined in the ActionSet.

2. But ActionSets are repeatedly created by the user, which is why, instead of defining the operation in the ActionSet, we define them in a blueprint and reference the blueprint in the ActionSet.




3. Finally those operations generate artifacts (for instance a dump on an S3 bucket) that need to be referenced in the Status of the ActionSet for future use.



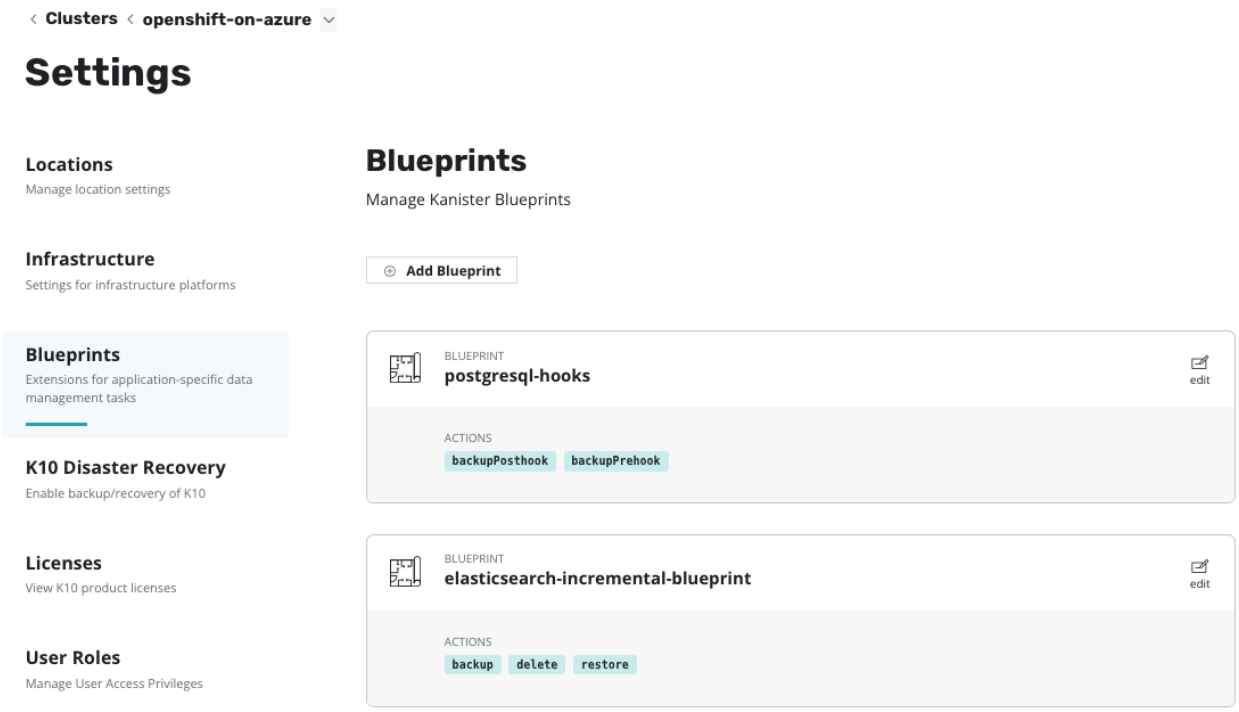

# Extending Kasten K10 with Kanister

We can extend Kasten K10 operations with Kanister. Let's replace the user in the diagram above with Kasten K10 to manage the execution and artifacts of the Kanister ActionSet. When Kasten backs up or restores a resource, it checks if the resource has the following specific annotation:

```
kanister.kasten.io/blueprint='<name of the blueprint>'
```

If this annotation is present, Kasten K10 will create a Kanister ActionSet including:

- A blueprint name in the annotation
- The action name, depending on what Kasten K10 is doing (backup, restore or delete)
- An annotated object
- The profile defined in the Kasten K10 policy




Blueprints can be managed directly from Kasten.

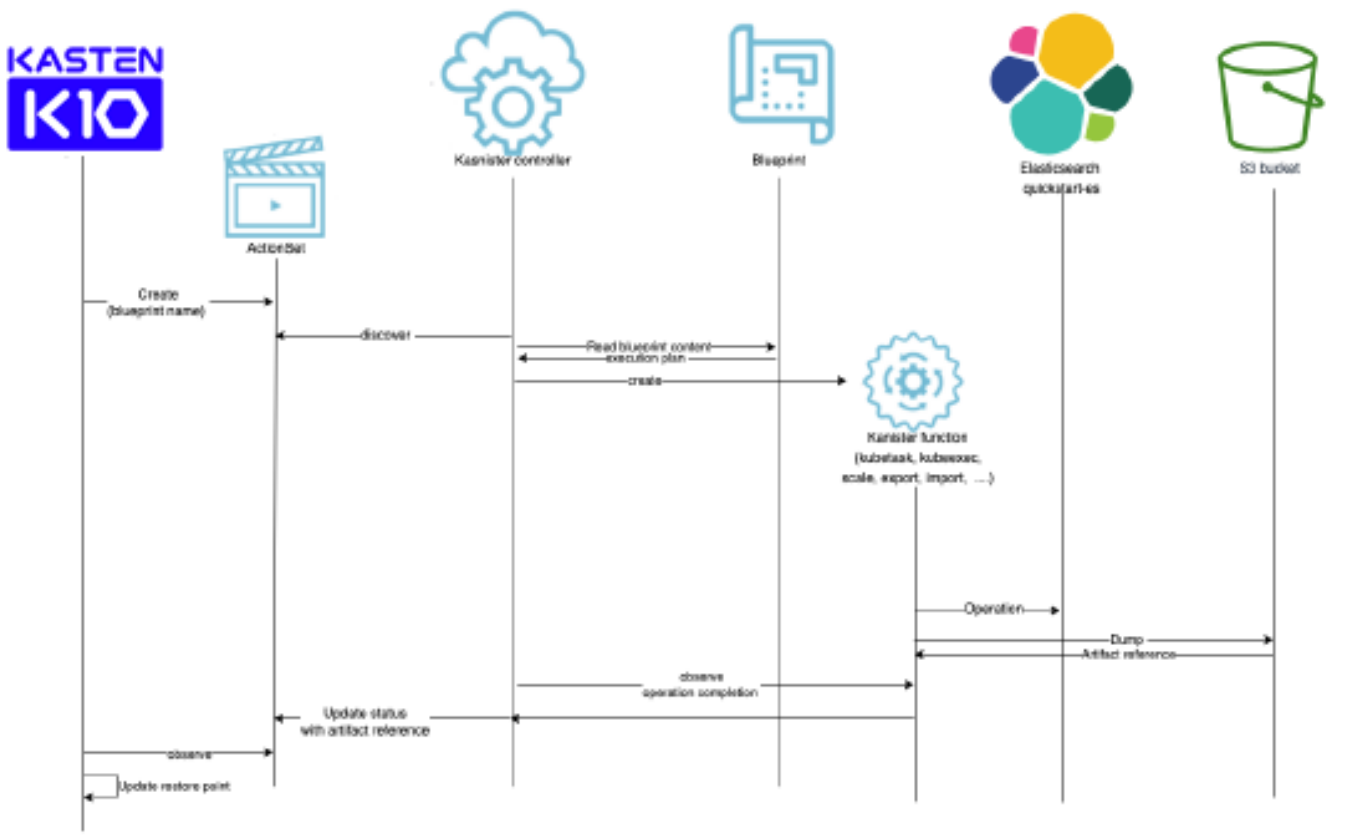
## Specific Action Names

Since the action name depends on the Kasten K10 action – backup, restore or delete – all three must be present in the blueprint. If the blueprint does not define delete for instance, Kasten K10 will not create the *ActionSet when it deletes the restore point.*

Note that any other action name is ignored by Kasten K10.

## Deleting the ActionSet

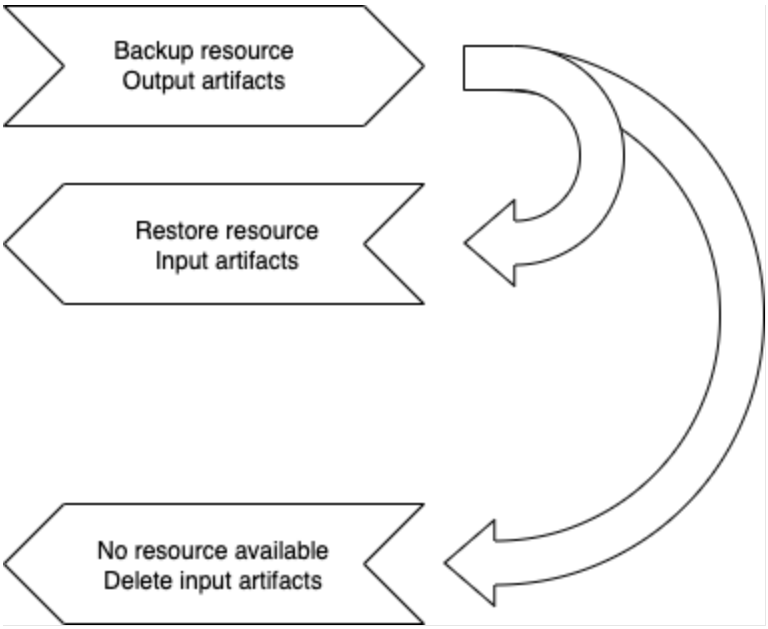
Once the ActionSet is successful, Kasten K10 will store the artifacts in the restore point.




## Developing the Blueprint

As we can see from the two previous diagrams, you don't need Kasten K10 to write and validate your blueprint. If your blueprint works in a standalone Kanister installation, it will also work with Kasten K10. The only real constraint is the name of the actions:

- **backup:** When Kasten K10 backs up the resource into a restore point, the target object will be the annotated object, and this operation will output artifacts.
- **restore:** When Kasten K10 restores the resource from a restore point, the target object will be the annotated object. This operation will input artifacts (created previously during backup).
- **delete:** When Kasten K10 deletes the restore point, we input the artifact reference for clean up (*e.g.* S3 removal), but we don't have the target object. Contrary to backup or restore, this object may not exist anymore (*e.g.* the namespace has been deleted).




# Execute Manually first

Before we write the blueprint, it is necessary that we prepare an environment and execute operations manually to make sure we have the right process.

This is a very important part of the activity, and you should not try to do it directly from Kanister in the first place. Why is that? Because Kanister adds another layer of complexity when automating this process. If you directly try to use Kanister, then you have to troubleshoot the combined issues coming from your Kanister development and your external process in actions. It's important to first validate your backup/restore process without export/import to the backup location. Once you're confident it works properly, you can encapsulate it in a Kanister blueprint. Many blueprint authors become stuck, because they don't take this first step, and they were unsure of where the problem came from.

## What are we going to test?

We are going to use `elasticdump` to create a backup of the Elastic indices. We'll test the restore from this backup, but we won't test export/import to/from the location profile, because this is managed by Kanister.

## Create a Kanister tool image

In our context we need:

- Elasticdump: the utility that creates `elasticdump`, which we'll have to move to a location profile.

- Kando: This is a Kanister tool that will handle data moving for the different backup location types.
- Two data movers: Kopia.io and Restic.net; Kopia is used in version 2 blueprint (discussed later).

Don't worry! It's not as hard as it sounds, thanks to Docker, which makes adding tools to various images much easier.

Installing `kando` is very simple. Simply copy a binary from the elasticdump image and add `kando`, `kopia`, and `restic` in the Dockerfile.

```
# docker build . -t michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
# docker push michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
FROM elasticdump/elasticsearch-dump:v6.88.0

LABEL org.opencontainers.image.authors="michael@kasten.io"

# we need to install curl for a minimum certificate in the
# truststore and for sending command to the cluster
RUN apt-get update && apt-get -y install curl

COPY --from=ghcr.io/kanisterio/kanister-tools:0.81.0 /usr/local/bin/kando
/usr/local/bin/kando
COPY --from=restic/restic:0.11.0 /usr/bin/restic /usr/local/bin/restic
COPY --
from=ghcr.io/kanisterio/kopia@sha256:87648ef24ce47f1d74ef5fa70bff96080f686b849dd0d787e
1699d4c05807c4b \
 /kopia/kopia /usr/local/bin/kopia
```

Next, build and push, and replace *michaelcourcy* with your choice of container repository.

```
docker build . -t michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
docker push michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
```

That's all it takes to build our images.

## Create an Elasticsearch cluster

Before testing the image, you need the Elasticsearch operator and a running cluster. Follow the [Elasticsearch documentation].

```
kubectl create -f https://download.elastic.co/downloads/eck/2.4.0/crds.yaml
kubectl apply -f https://download.elastic.co/downloads/eck/2.4.0/operator.yaml
```

The Elasticsearch instance is createds  with the Elasticsearch custom resource. Here we'll create it in the test-es1 namespace:

```
kubectl create ns test-es1
kubectl config set-context --current --namespace test-es1
cat <<EOF | kubectl create -n test-es1 -f -
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
 name: quickstart
spec:
 version: 8.4.1
 nodeSets:
 - name: default
   count: 2
   volumeClaimTemplates:
   - metadata:
       name: elasticsearch-data
     spec:
       accessModes:
       - ReadWriteOnce
       resources:
         requests:
           storage: 100Gi
   config:
     node.store.allow_mmap: false
EOF
```

Check that the cluster is ready, and after some time, health should turn green:

```
kubectl get -n test-es1 elasticsearch
NAME         HEALTH   NODES   VERSION   PHASE   AGE
quickstart   green    2       8.4.1     Ready   6m34s
```

## Create data

Now create a client in a terminal. We'll call the terminal "es client terminal":

```
PASSWORD=$(kubectl get -n test-es1 secret quickstart-es-elastic-user \
    -o go-template='{{.data.elastic | base64decode}}')
ES_URL="https://quickstart-es-http:9200"
kubectl run -n test-es1 curl -it \
```

```
   --restart=Never \
   --rm --image ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0 \
   --env="PASSWORD=$PASSWORD" --env="ES_URL=$ES_URL" --command bash
```

Test the connectivity to the cluster:

```
curl -u "elastic:$PASSWORD" -k "${ES_URL}"
{
  "name" : "quickstart-es-default-1",
  "cluster_name" : "quickstart",
  "cluster_uuid" : "GNFxLabyTHypBW9JpDDzsw",
  "version" : {
    "number" : "8.4.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "2bd229c8e56650b42e40992322a76e7914258f0c",
    "build_date" : "2022-08-26T12:11:43.232597118Z",
    "build_snapshot" : false,
    "lucene_version" : "9.3.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Create two indices and a document:

```
# List the index
curl  -k -u "elastic:$PASSWORD" -X GET "${ES_URL}/*?pretty"
# Create an index
curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/my-index-000001?pretty"
#  add an index and a document
curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/my-index-
000002/_doc/1?timeout=5m&pretty" -H 'Content-Type: application/json' -d'
{
 "@timestamp": "2099-11-15T13:12:00",
 "message": "GET /search HTTP/1.1 200 1070000",
 "user": {
   "id": "kimchy-2"
 }
}
```

```
'
# retrieve document
curl -k -u "elastic:$PASSWORD" -X GET "${ES_URL}/my-index-000002/_doc/1?pretty"
```

## Test a backup

Now we are ready to test our process. Let's create a temporary pod with the image we just built in another terminal that we will call the "elasticdump terminal":

```
PASSWORD=$(kubectl get -n test-es1 secret quickstart-es-elastic-user \
   -o go-template='{{.data.elastic | base64decode}}')
ES_URL="https://elastic:$PASSWORD@quickstart-es-http:9200"

kubectl run --restart=Never -n test-es1 --rm -it --image=michaelcourcy/elasticdump-
kanister:v6.88.0-0.81.0 elasticdump \
   --env="ES_URL=$ES_URL"\
   --command -- bash
```

Let's execute a backup of all the indices:

```
mkdir /tmp/es_backup
NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
 --direction=dump \
 --match='^.*$' \
 --input=$ES_URL \
 --output=/tmp/es_backup
```

The output should look like this, and you should find your index in /tmp/es_backup:

```
Thu, 29 Sep 2022 13:48:18 GMT | got 1 objects from source elasticsearch (offset: 0)
Thu, 29 Sep 2022 13:48:18 GMT | sent 1 objects to destination file, wrote 1
Thu, 29 Sep 2022 13:48:18 GMT | got 0 objects from source elasticsearch (offset: 1)
Thu, 29 Sep 2022 13:48:18 GMT | Total Writes: 1
Thu, 29 Sep 2022 13:48:18 GMT | dump complete
Thu, 29 Sep 2022 13:48:18 GMT |  dumping all done
Thu, 29 Sep 2022 13:48:18 GMT |  bye
root@elasticdump:/# ls /tmp/es_backup/
my-index-000001.json          my-index-000001.settings.json  my-index-000002.json
my-index-000002.settings.json
my-index-000001.mapping.json  my-index-000001.template.json  my-index-
000002.mapping.json  my-index-000002.template.json
```

## Test a Restore

Now let's test a restore. According to the official documentation, you will need to execute some operations first:

- Turn off GeoIP database downloader
- Turn off ILM
- Turn off Machine Learning
- Turn off Monitoring
- Turn off Watcher
- Allow the deletions of all the indices using a wildcard
- Delete all the indices and data streams

Return to the "es client terminal" and execute those commands:

```
curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "ingest.geoip.downloader.enabled": false
 }
}
'
curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_ilm/stop?pretty"
curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL}/_ml/set_upgrade_mode?enabled=true&pretty"
curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "xpack.monitoring.collection.enabled": false
 }
}
'
curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_watcher/_stop?pretty"
curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "action.destructive_requires_name": false
 }
}
```

```
'
curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL}/_data_stream/*?expand_wildcards=all&pretty"
curl  -k -u "elastic:$PASSWORD" -X DELETE "${ES_URL}/*?expand_wildcards=all&pretty"
```

Now you can restore, return to the “elasticdump terminal” and execute the restore command:

```
NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
 --direction=load \
 --match='^.*$' \
 --input=/tmp/es_backup \
 --output=$ES_URL
```

Of course, you need to turn on all the things that you turned off, and return to the “es client terminal”:

```
curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "ingest.geoip.downloader.enabled": true
 }
}
'
curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_ilm/start?pretty"
curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL}/_ml/set_upgrade_mode?enabled=false&pretty"
curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "xpack.monitoring.collection.enabled": true
 }
}
'
curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_watcher/_start?pretty"
curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "action.destructive_requires_name": null
```

```
 }
}
'
```

Let's test that we get back our data in the "es client terminal"

```
curl -k -u "elastic:$PASSWORD" -X GET "${ES_URL}/my-index-000002/_doc/1?pretty"
```

## An Unexpected Effect

On the second backup, I experienced an unexpected effect. The backup took much longer to complete… and I saw that `elasticdump` was backing up the ".monitoring" index. What is this new index?

```
dumping https://elastic:irUuQ0eJ7898e084xTBpC8D9@quickstart-es-http:9200/.monitoring-es-7-2022.09.30 to /tmp/es_backup/.monitoring-es-7-2022.09.30.json
Mon, 03 Oct 2022 13:20:16 GMT [debug] | fork: /usr/local/lib/node_modules/elasticdump/bin/elasticdump --type=data,--input=https://elastic:irUuQ0eJ7898e084xTBpC8D9@quickstart-es-http:9200/.monitoring-es-7-2022.09.30,--output=/tmp/es_backup/.monitoring-es-7-2022.09.30.json,--scrollId=null,--scrollTime=10m,--limit=100,--offset=0,--size=-1,--searchBody=null,--searchWithTemplate=null,--prefix=,--suffix=,--support-big-int=false,--big-int-fields=,--headers=null,--cert=null,--key=null,--pass=null,--ca=null,--tlsAuth=false,--input-cert=null,--input-key=null,--input-pass=null,--input-ca=null,--output-cert=null,--output-key=null,--output-pass=null,--output-ca=null,--httpAuthFile=null,--concurrency=1,--carryoverConcurrencyCount=true,--intervalCap=5,--concurrencyInterval=5000,--overwrite=false,--fsCompress=false,--awsChain=false,--awsAccessKeyId=null,--awsSecretAccessKey=null,--awsIniFileProfile=null,--awsService=null,--awsRegion=null,--awsUrlRegex=null,--s3AccessKeyId=null,--s3SecretAccessKey=null,--s3Region=null,--s3Endpoint=null,--s3SSLEnabled=true,--s3ForcePathStyle=false,--s3Compress=false,--s3ServerSideEncryption=null,--s3SSEKMSKeyId=null,--s3ACL=null,--quiet=false,--scroll-with-post=false
Mon, 03 Oct 2022 13:20:16 GMT | starting dump
(node:96) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to '0' makes TLS connections and HTTPS requests insecure by disabling certificate verification.
(Use `node --trace-warnings ...` to show where the warning was created)
Mon, 03 Oct 2022 13:20:16 GMT | got 100 objects from source elasticsearch (offset: 0)
Mon, 03 Oct 2022 13:20:16 GMT | sent 100 objects to destination file, wrote 100
Mon, 03 Oct 2022 13:20:16 GMT | got 100 objects from source elasticsearch (offset: 100)
```

```
...
```

When we turned  monitoring off and on, it triggered the creation of the “.monitoring” index.

```
curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty" -H
'Content-Type: application/json' -d'
{
 "persistent": {
   "xpack.monitoring.collection.enabled": true
 }
}
'
```

Depending on your use case, you may or may not want to capture the “system” indices starting with a dot (.).

In my case, I don’t want those indices, so I changed the command to exclude any indices starting with a dot (.).

```
# we don't want to capture indices starting with a dot "." like ".monitoring"
# --match='^[^.].+'
NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
 --direction=dump \
 --match='^[^.].+' \
 --input=$ES_URL \
 --output=/tmp/es_backup
```

**Manually testing your process before encapsulating it in a blueprint is a mandatory step. It’s also important to run backup and restore multiple times to insure that you capture all the side effects to make your process idempotent. This is a best practice to minimize new testing concerns before moving on to Kanister blueprint development.**

# Creating the Blueprint

We have properly validated our backup and restore process.Now iIt's time to encapsulate it in a blueprint. Let’s begin to implement the backup action first.

## Implement the Backup Action

For the backup action, we are going to make sure that we can obtain the secret generated by the operator:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   phases:
   - func: KubeTask
     name: multielasticdump
     # We introduce a secret <-------------------------------
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
         NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
         --direction=dump \
         --match='^[^.].*$' \
         --input=$ES_URL \
         --output=/tmp/es_backup
         sleep 4
```

## Working with Secrets

When working with data services, it's very common to work with secrets to connect and operate on data service APIs. When secrets are generated by Helm charts or operators, they may have their own way to create secrets. Being able to evaluate secrets in a template allows you to handle multiple use cases.

We introduce the secret object in the multielasticdump phase with the help of Golang templating:

```
    name: multielasticdump
    objects:
      elasticSecret:
        kind: Secret
        name: '{{ .Object.metadata.name }}-es-elastic-user'
        namespace: '{{ .Object.metadata.namespace }}'
```

If the Elasticsearch object is named quickstart, then the operator will create the secret quickstart-es-elastic-user. For this reason, the template `{{ .Object.metadata.name }}-es-elastic-user` will render dynamically this secret in the blueprint.

Now the content of the secret can be used in the different phases of the action:

```
PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data "elastic" |
toString }}"
```

We are using go-template primitive `index` to navigate the phases `.Phases.multielasticdump` and obtain the content of the secret `.Secrets.elasticSecret.Data`. You can use this `{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data "elastic" | toString }}` in any phase of the same action, because phases are shared in memory. If you want to share information between actions you need to create artifacts.

## Test the Backup Action

Let's create the blueprint, assuming kubectl is on your $PATH and configured with your Kubernetes cluster.

```
kubectl create -f elastic-bp.yaml
```

To run Kanister, you need a backup profile. It won't be used immediately, but it is mandatory to run Kanister.

To facilitate the creation of profile and ActionSet, we use a tool called `kanctl`. Download and install this tool here and add it to your $PATH for execution.

```
curl https://raw.githubusercontent.com/kanisterio/kanister/master/scripts/get.sh |
bash
```

Your Kubernetes context must point to your cluster before you run `kanctl`:

```
kanctl create profile s3compliant -n kasten-io \
  --access-key $AWS_S3_ACCESS_KEY_ID \
  --secret-key $AWS_S3_SECRET_ACCESS_KEY \
  --region eu-west-3 \
  --bucket michael-kopia
```

You should see a result like this:

```
secret 's3-secret-hmiakj' created
profile 's3-profile-vfqll' created
```

`Kanctl` helps you create the profile, which is a custom resource that you could create manually:

```
kubectlget profiles.cr.kanister.io s3-profile-vfqll -n kasten-io -o yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Profile
metadata:
 name: s3-profile-vfqll
 namespace: kasten-io
credential:
 keyPair:
   idField: aws_access_key_id
   secret:
     apiVersion: ""
     group: ""
     kind: ""
     name: s3-secret-hmiakj
     namespace: kasten-io
     resource: ""
   secretField: aws_secret_access_key
 type: keyPair
location:
 bucket: michael-kopia
```

```
 endpoint: ""
 prefix: ""
 region: eu-west-3
 type: s3Compliant
skipSSLVerify: false
```

In my example, I created a profile corresponding to an AWS S3 bucket, to see the possible options used:

```
kanctl create profile --help
```

You may also use Azure and GCP. Kasten K10 adds NFS capabilities.

Now, let's create the ActionSet:

```
kanctl create actionset \
  --action backup --namespace kasten-io \
  --blueprint elastic-bp \
  --profile kasten-io/s3-profile-vfqll \
  --objects elasticsearch.k8s.elastic.co/v1/elasticsearches/test-es1/quickstart
```

The output gives you the name of the ActionSet:

```
actionset backup-mq7wh created
```

You can check the status of the ActionSet:

```
kubectlget actionset backup-464xq -n kasten-io -o jsonpath='{.status}' |jq
{
 "actions": [
   {
     "blueprint": "elastic-bp",
     "deferPhase": {
       "name": "",
       "state": ""
     },
     "name": "backup",
     "object": {
       "apiVersion": "v1",
       "group": "elasticsearch.k8s.elastic.co/",
       "kind": "",
       "name": "quickstart",
       "namespace": "test-es1",
```

```
        "resource": "elasticsearches"
      },
      "phases": [
        {
          "name": "multielasticdump",
          "state": "complete"
        }
      ]
    }
  ],
  "error": {
    "message": ""
  },
  "progress": {
    "lastTransitionTime": "2022-09-30T18:11:28Z",
    "percentCompleted": "100.00"
  },
  "state": "complete"
}
```

The status gives you a report of the action:

- The blueprint and action that was used
- The object that was the target of the ActionSet
- The current state

In this case, the action state is “complete,” which means successful execution of the action.

## Debugging

It is difficult to write a Kanister blueprint from scratch the first time, and you will need a way to debug what’s going on. Debugging is easy: execute a kubetask (a task pod) in the test-es1 namespace with the `bash` parameter “-x” and add “sleep 4” to let the pod live a little longer after process completion. These useful additions allow capturing the execution logs.

In the first terminal, run this command to capture the logs of any Kanister pod in the test-es1 namespace:

```
while true; do kubectl  -n test-es1 logs -f -l createdBy=kanister; sleep 2; done
```

You should see this output because we have not yet launched an ActionSet:

```
No resources found in test-es1 namespace.
No resources found in test-es1 namespace.
No resources found in test-es1 namespace.
```

In the second terminal, launch the ActionSet again:

```
kanctl create actionset \
  --action backup --namespace kasten-io \
  --blueprint elastic-bp \
  --profile kasten-io/s3-profile-d69lg \
  --objects elasticsearch.k8s.elastic.co/v1/elasticsearches/test-es1/quickstart
actionset backup-dzks8 created
```

In the first terminal, the output changes to:

```
while true; do kubectl  -n test-es1 logs -f -l createdBy=kanister; sleep 2; done
No resources found in test-es1 namespace.
No resources found in test-es1 namespace.
Error from server (BadRequest): container "container" in pod "kanister-job-p59d4" is
waiting to start: ContainerCreating
Mon, 03 Oct 2022 22:12:40 GMT | starting dump
(node:40) Warning: Setting the NODE_TLS_REJECT_UNAUTHORIZED environment variable to
'0' makes TLS connections and HTTPS requests insecure by disabling certificate
verification.
(Use `node --trace-warnings ...` to show where the warning was created)
Mon, 03 Oct 2022 22:12:40 GMT | got 1 objects from source
....
....
Mon, 03 Oct 2022 22:12:42 GMT | got 0 objects from source elasticsearch (offset: 1)
Mon, 03 Oct 2022 22:12:42 GMT | Total Writes: 1
Mon, 03 Oct 2022 22:12:42 GMT | dump complete
Mon, 03 Oct 2022 22:12:43 GMT |  dumping all done
Mon, 03 Oct 2022 22:12:43 GMT |  bye
+ sleep 4
```

## Create an Artifact

The backup action is not finished. The next steps are:

- Compress the backup in an archive
- Define a unique path on the backup location profile
- Send the archive to the profile at this path

To define a unique path, we can rely on the name of the namespace and date functions. Let's add this part using the Go-template date function:

```
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
```

```
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
```

Now the blueprint looks like this:

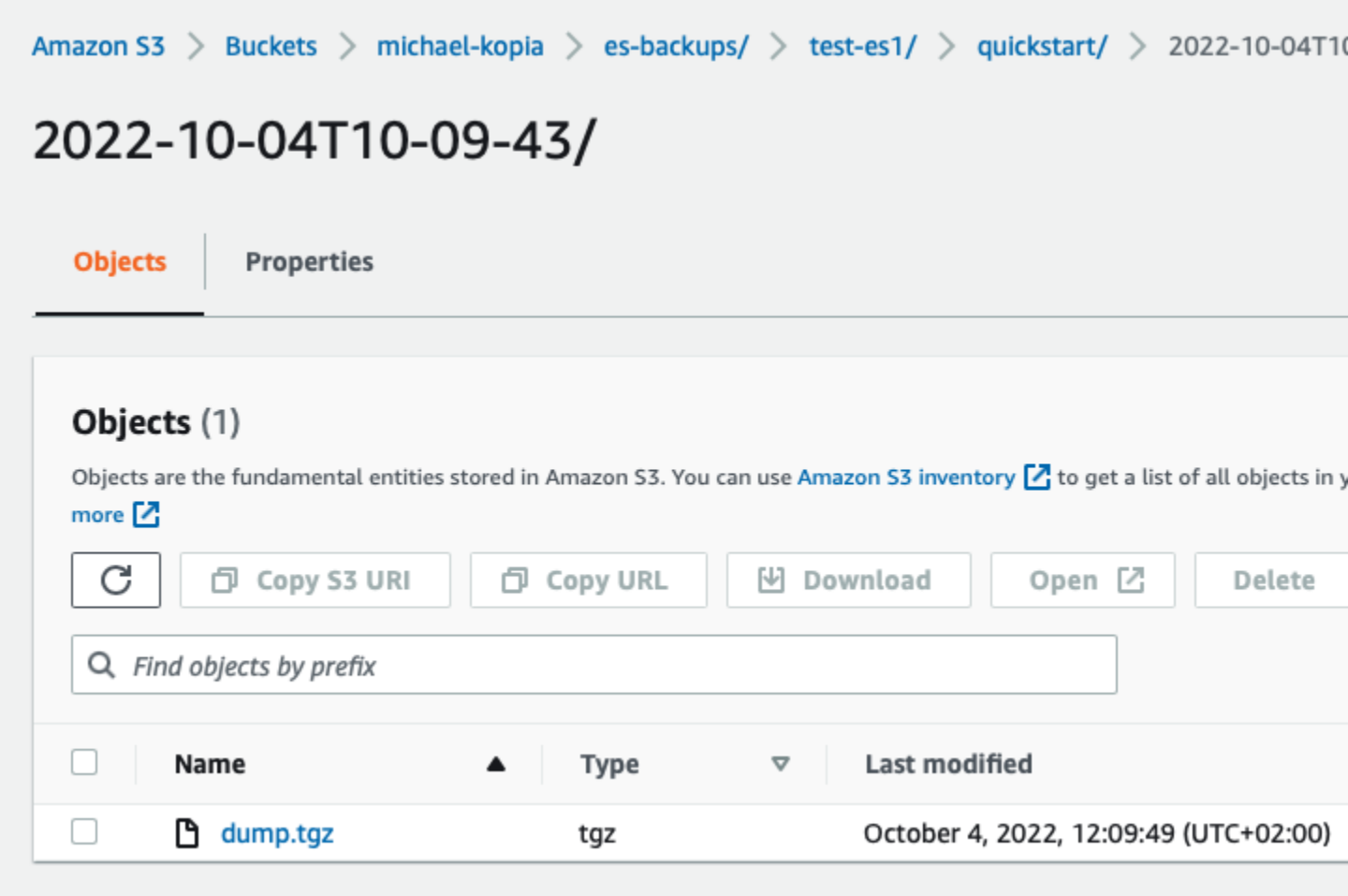
```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
         NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
         --direction=dump \
```

```
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        sleep 4
```

Execute the blueprint by running the same `kanctl` command:

```
kanctl create actionset \
  --action backup --namespace kasten-io \
  --blueprint elastic-bp \
  --profile kasten-io/s3-profile-vfqll \
  --objects elasticsearch.k8s.elastic.co/v1/elasticsearches/test-es1/quickstart
```

The dump.tgz is exported on the S3 bucket:




That's good progress, however there is no reference of this artifact in the ActionSet for handling a restore. We still need to output the artifact into the ActionSet status.

## Output an artifact in the ActionSet

Outputting artifacts in the ActionSet is important, because this is how you know the location of the backup that has been created by this ActionSet. It's how you store your operational result.

To output an artifact in the ActionSet, Kanister has a specific construct: `outputArtifacts`. Define the name of the map you want to create, for instance myObjects, and the keyValue pairs inside:

```
outputArtifacts:
 myObjects:
   keyValue:
     a: b
```

Let’s run this blueprint, which adds the artifact section:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     myObjects:
       keyValue:
         a: b
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
         NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
         --direction=dump \
         --match='^[^.].*$' \
```

```
        --input=$ES_URL \
        --output=/tmp/es_backup
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        sleep 4
```

Observe the status of the ActionSet:

```
kubectlget -o jsonpath='{.status}' actionset backup-lb7cd -n kasten-io |jq
{
 "actions": [
   {
     "artifacts": {
       "myObjects": {
         "keyValue": {
           "a": "b"
         }
       }
     },
     "blueprint": "elastic-bp",
     "deferPhase": {
       "name": "",
       "state": ""
     },
     "name": "backup",
     "object": {
       "apiVersion": "v1",
       "group": "elasticsearch.k8s.elastic.co/",
       "kind": "",
       "name": "quickstart",
       "namespace": "test-es1",
       "resource": "elasticsearches"
     },
     "phases": [
       {
         "name": "multielasticdump",
         "output": {
```

```
            "backup_path": "/es-backups/test-es1/quickstart/2022-10-04T13-10-
48/dump.tgz"
          },
          "state": "complete"
        }
      ]
    }
  ],
  "error": {
    "message": ""
  },
  "progress": {
    "lastTransitionTime": "2022-10-04T13:11:04Z",
    "percentCompleted": "100.00"
  },
  "state": "complete"
}
```

Now we have an artifact section with a keyValue a->b.

## Output Dynamic Artifact

This construct is interesting – outputting constant keyPair like a->b – but we want “b” to be evaluated dynamically.

It is possible to feed the Output map of a phase by writing to the stdout:

```
###Phase-output###: {"key":"a","value":"b"}
```

Kanister reads the stdout and identifies lines starting with `###Phase-output###:` it parses the key and the value, then adds it to the phase Output map.

To make it easier and guarantee consistent stdout writing, you should use the `kando output key value` command.

The key-value a->b is now added to the `Phases.multielasticdump.Output` map. It can be reused anywhere in the action with this template expression:
`{{ .Phases.multielasticdump.Output.a }}`

Let’s test this blueprint:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
```

```
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     myObjects:
       keyValue:
         a: "{{ .Phases.multielasticdump.Output.a }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
         NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
         --direction=dump \
         --match='^[^.].*$' \
         --input=$ES_URL \
         --output=/tmp/es_backup
         backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
```

```
"2006-01-02T15-04-05" }}/dump.tgz"
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output a b
        sleep 4
```

Now that we understand the basics of outputting an artifact, let's enhance the backup action to its final version:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       keyValue:
         backup_path: "{{ .Phases.multielasticdump.Output.backup_path }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
```

```
        PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        mkdir /tmp/es_backup
        # we don't want to capture indices starting with a dot "." like ".monitoring"
        # --match='^[^.].+'
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output backup_path $backup_path
        sleep 4
```

Testing the blueprint backup action:

```
kubectlreplace -f elastic-bp.yaml
blueprint.cr.kanister.io/elastic-bp replaced
kanctl create actionset \
  --action backup --namespace kasten-io \
  --blueprint elastic-bp \
  --profile kasten-io/s3-profile-vfqll \
  --objects elasticsearch.k8s.elastic.co/v1/elasticsearches/test-es1/quickstart
actionset backup-xp4m9 created
kubectlget -o jsonpath='{.status}' actionset backup-xp4m9 -n kasten-io |jq
{
 "actions": [
   {
     "artifacts": {
       "cloudObjects": {
         "keyValue": {
           "backup_path": "/es-backups/test-es1/quickstart/2022-10-04T14-18-
01/dump.tgz"
         }
       }
```

```
    },
    "blueprint": "elastic-bp",
    "deferPhase": {
      "name": "",
      "state": ""
    },
    "name": "backup",
    "object": {
      "apiVersion": "v1",
      "group": "elasticsearch.k8s.elastic.co/",
      "kind": "",
      "name": "quickstart",
      "namespace": "test-es1",
      "resource": "elasticsearches"
    },
    "phases": [
      {
        "name": "multielasticdump",
        "output": {
          "backup_path": "/es-backups/test-es1/quickstart/2022-10-04T14-18-
01/dump.tgz"
        },
        "state": "complete"
      }
    ]
  }
 ],
 "error": {
   "message": ""
 },
 "progress": {
   "lastTransitionTime": "2022-10-04T14:18:17Z",
   "percentCompleted": "100.00"
 },
 "state": "complete"
}
```


Congratulations, you've completed a successful backup action and created an artifact with a Kanister blueprint! Let's compliment the backup action with a restore action next.

## Implement Restore Action

An overview of the restore action operations:

- Input an artifact
- Pull the archive
- Unarchive
- Execute the operation before restoring
- Execute the restore
- Execute the operation after restoring

Let’s build up the action gradually. We’re going to implement, test, and troubleshoot the first three operations with Kanister.

## Kanister troubleshooting

Let’s add the restore action in our blueprint:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       keyValue:
         backup_path: "{{ .Phases.multielasticdump.Output.backup_path }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
```

```
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        mkdir /tmp/es_backup
        # we don't want to capture indices starting with a dot "." like ".monitoring"
        # --match='^[^.].+'
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output backup_path $backup_path
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
```

```
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        echo "pull and untar the es artifact"
        kando location pull --profile '{{ toJson .Profile }}' --path '{{
.ArtifactsIn.cloudObjects.KeyValue.backup_path }}' dump.tgz
        tar xvzf dump.tgz
        ls -R /tmp
        echo "Preparing es for restore, coming soon ...."
        sleep 4
```

Update the blueprint with the restore action, above, and in the first terminal, watch for the kube task execution:

```
while true; do kubectl  -n test-es1 logs -f -l createdBy=kanister; sleep 2; done
```

In a second terminal, launch the restore ActionSet from a successful backup ActionSet:

```
kanctl --namespace kasten-io create actionset --action restore --from "backup-wwp66"
actionset restore-backup-wwp66-cwnnm created
```

But nothing happens in the first terminale:

```
No resources found in test-es1 namespace.
No resources found in test-es1 namespace.
No resources found in test-es1 namespace.
```

Let’s check the action set:

```
kubectlget actionset -n kasten-io restore-backup-wwp66-cwnnm -o
jsonpath='{.status.error}' | jq
{
 "message": "Failed to render template: \"multielasticdump\" not found"
}
```

Aha! I renamed the phase multielasticrestore:

```
  phases:
  - func: KubeTask
    name: multielasticrestore
```

but we used multielasticbackup in the template:

```
PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data "elastic" |
toString }}"
```

That’s not going to work, so let’s fix it:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       keyValue:
         backup_path: "{{ .Phases.multielasticdump.Output.backup_path }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
```

```
        PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        mkdir /tmp/es_backup
        # we don't want to capture indices starting with a dot "." like ".monitoring"
        # --match='^[^.].+'
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output backup_path $backup_path
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
```

```
        PASSWORD="{{ index .Phases.multielasticrestore.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        echo "pull and untar the es artifact"
        kando location pull --profile '{{ toJson .Profile }}' --path '{{
.ArtifactsIn.cloudObjects.KeyValue.backup_path }}' dump.tgz
        tar xvzf dump.tgz
        echo "Preparing es for restore, coming soon ..."
        sleep 4
```

## Finishing the Restore Action

The rest of the restore action is consistent with the operations we did manually:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       keyValue:
         backup_path: "{{ .Phases.multielasticdump.Output.backup_path }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
```

```
      - -o
      - pipefail
      - -c
      - |
        PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        mkdir /tmp/es_backup
        # we don't want to capture indices starting with a dot "." like ".monitoring"
        # --match='^[^.].+'
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output backup_path $backup_path
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
```

```
      - -o
      - pipefail
      - -c
      - |
        PASSWORD="{{ index .Phases.multielasticrestore.Secrets.elasticSecret.Data
"elastic" | toString }}"
        ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
        ES_URL_CURL="https://{{ .Object.metadata.name }}-es-http:9200"
        echo "pull and untar the es artifact"
        kando location pull --profile '{{ toJson .Profile }}' --path '{{
.ArtifactsIn.cloudObjects.KeyValue.backup_path }}' dump.tgz
        tar xvzf dump.tgz
        echo "Preparing es for restore"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/stop?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=true&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_stop?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": false
         }
        }
        '
```

```
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/_data_stream/*?expand_wildcards=all&pretty"
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/*?expand_wildcards=all&pretty"
        echo "We are now ready to restore elasticsearch"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=load \
        --match='^.*$' \
        --input=/tmp/es_backup \
        --output=$ES_URL
        echo "Restore successful, let's restart the service we stopped"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/start?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=false&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_start?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": null
         }
        }
        '
        sleep 4
```

To test it, you can delete the indices before the restore:

```
kubectl run --restart=Never -n test-es1 --rm -it --image=michaelcourcy/elasticdump-
kanister:v6.88.0-0.81.0 elasticdump \
   --env="ES_URL=$ES_URL"\
   --image-pull-policy=Always \
   --command -- bash
curl -k -u "elastic:$PASSWORD" -X DELETE "${ES_URL}/my-index-000001?pretty"
curl -k -u "elastic:$PASSWORD" -X DELETE "${ES_URL}/my-index-000002?pretty"
curl -k -u "elastic:$PASSWORD" -X GET "${ES_URL}/my-index-000001?pretty"
{
 "error" : {
   "root_cause" : [
     {
       "type" : "index_not_found_exception",
       "reason" : "no such index [my-index-000001]",
       "index_uuid" : "_na_",
       "resource.type" : "index_or_alias",
       "resource.id" : "my-index-000001",
       "index" : "my-index-000001"
     }
   ],
   "type" : "index_not_found_exception",
   "reason" : "no such index [my-index-000001]",
   "index_uuid" : "_na_",
   "resource.type" : "index_or_alias",
   "resource.id" : "my-index-000001",
   "index" : "my-index-000001"
 },
 "status" : 404
}
```

Now launch the restore in another terminal:

```
kanctl --namespace kasten-io create actionset --action restore --from "backup-wwp66"
```

Check if you retrieve the index my-index-000001:

```
curl -k -u "elastic:$PASSWORD" -X GET "${ES_URL}/my-index-000001?pretty"
{
 "my-index-000001" : {
   "aliases" : { },
   "mappings" : { },
```

```
   "settings" : {
     "index" : {
       "routing" : {
         "allocation" : {
           "include" : {
             "_tier_preference" : "data_content"
           }
         }
       },
       "number_of_shards" : "1",
       "provided_name" : "my-index-000001",
       "creation_date" : "1664967371405",
       "number_of_replicas" : "1",
       "uuid" : "7gfp7AkXQnWVjPLsQEV09g",
       "version" : {
         "created" : "8040199"
       }
     }
   }
 }
}
```

Backup and restore actions are now looking good. We are able to accomplish this with just two commands to manage backup and restore of the Elasticsearch cluster.

```
kanctl create actionset \
  --action backup --namespace kasten-io \
  --blueprint elastic-bp \
  --profile kasten-io/s3-profile-vfqll \
  --objects elasticsearch.k8s.elastic.co/v1/elasticsearches/test-es1/quickstart
actionset backup-cvvf5 created
kanctl --namespace kasten-io create actionset --action restore --from backup-cvvf5
```

This Kanister blueprint applies to any other Elasticsearch cluster, in any namespace, so it was worth the effort!

## Implement the delete action

How do we remove the backup when they are not needed anymore and we want to reclaim storage space at the backup location?

Deleting is a much simpler operation; we don't need to deal with the Elasticsearch service. Kando comes with a command : `kando location delete`

Here is the new version of the blueprint with the delete action at the end:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       keyValue:
         backup_path: "{{ .Phases.multielasticdump.Output.backup_path }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
```

```
        backup_path="/es-backups/{{ .Object.metadata.namespace }}/{{
.Object.metadata.name }}/{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time  | date
"2006-01-02T15-04-05" }}/dump.tgz"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path
dump.tgz
        kando output backup_path $backup_path
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticrestore.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         ES_URL_CURL="https://{{ .Object.metadata.name }}-es-http:9200"
         echo "pull and untar the es artifact"
         kando location pull --profile '{{ toJson .Profile }}' --path '{{
```

```
.ArtifactsIn.cloudObjects.KeyValue.backup_path }}' dump.tgz
        tar xvzf dump.tgz
        echo "Preparing es for restore"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/stop?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=true&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_stop?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/_data_stream/*?expand_wildcards=all&pretty"
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/*?expand_wildcards=all&pretty"
        echo "We are now ready to restore elasticsearch"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=load \
        --match='^.*$' \
        --input=/tmp/es_backup \
        --output=$ES_URL
```

```
        echo "Restore successful, let's restart the service we stopped"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/start?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=false&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_start?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": null
         }
        }
        '
        sleep 4
 delete:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: deleteFromObjectStore
     args:
       namespace: "{{ .Namespace.Name }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
```

```
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        kando location delete --profile '{{ toJson .Profile }}' --path '{{
.ArtifactsIn.cloudObjects.KeyValue.backup_path }}'
```

Notice that we are not working anymore on any object, because there is no relation between removing the artifacts and having the service up and running: they are independent. For instance, we may have deleted all the Elasticsearch clusters, but still need to remove the artifacts at a later time.

However, you must define a namespace where the kubetask will be launched. You can add the `.Namespace` object in the template with the `namespacetargets` option:

```
kanctl --namespace kasten-io --namespacetargets test-es1 create actionset --action
delete --from backup-cvvf5
```

Render it in your action with the template: `{{ .Namespace.Name }}`.

## Validate the Blueprint with the Experts

In this blueprint, we made decisions such as not backing up system indices or stopping the machine learning feature before restoring. Are those decisions relevant? It depends. At some point, you'll have to validate your assumptions and design decisions with your company experts, such as your Database Administrator (DBA) or any responsible person in charge of the deployment and maintenance of the data service.

The diversity of data services and use cases is huge, so Kasten can't make this kind of decision or be authoritative on all the databases, for all versions, for all possible deployments and for all the possible Service Level Agreements (SLAs). When you have a working blueprint, it is best to share your blueprint with the experts for validation.

It's ideal for those experts to potentially own the blueprint and its maintenance. If you reach this point, you have accomplished your mission by assigning the responsibility to the right personas.

# Kasten K10 Integration

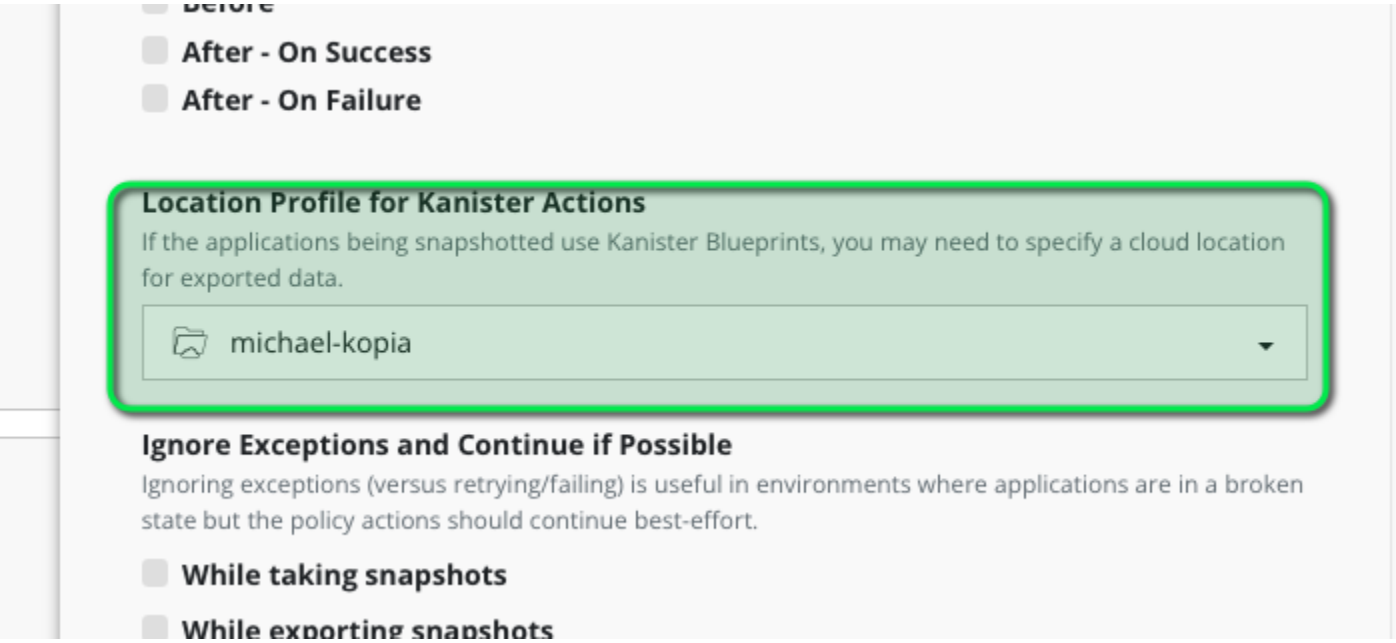
## Backup

### Let Kasten K10 execute the ActionSet for you

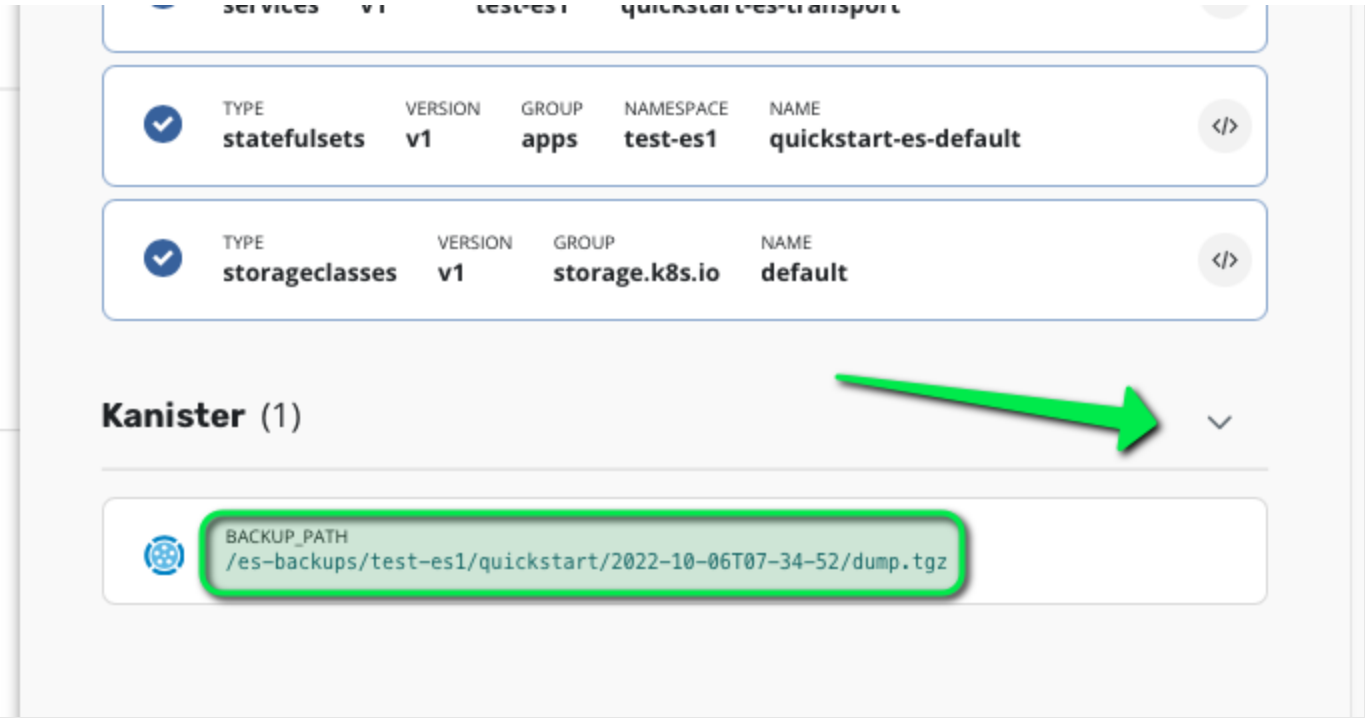
To integrate the blueprint with Kasten K10, the only thing you have to do is annotate the target object:

```
k annotate -n test-es1 elasticsearch quickstart kanister.kasten.io/blueprint=elastic-bp
```

When Kasten K10 backs up, restores, or deletes a restore point, it creates a backup, restore, or delete ActionSet, respectively. You can have this object as a target with a Kanister profile you define in the policy:




After a successful run on the policy, you should see the artifact in the restore point:




The integration with Kasten is seamless!

Next, we have two areas for improvement:

- Can I use the Kasten data mover (Kopia) instead of the default Kanister data mover (Restic)?
- Should I include the Elasticsearch PVC in the restore point if I have a dump?

Let's discuss each of these scenarios.

## Can I use the Kasten K10 data mover instead of the Kanister's?

Kasten K10 leverages a different data mover than Kanister: Kasten K10 uses Kopia.io but Kanister uses Restic.net.

There are several reason to favor Kopia over Restic:

- Kopia is more performant than Restic
- Encryption keys are managed by Kasten K10
- NFS profile support
- Using the same data mover for consistent operations

It is actually possible to switch from Restic to Kopia by explicitly defining the keyword `kopiaSnapshot` in the output artifacts. Kasten K10 then configures `kando` to use Kopia instead of Restic.

Warning: this blueprint works only with Kasten K10.

Here is the next version of the blueprint, named blueprint `elastic-bp-v2`:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp-v2
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     cloudObjects:
       # Capture the kopia snapshot information for subsequent actions
       # The information includes the kopia snapshot ID which is essential for restore
and delete to succeed
       # `kopiaOutput` is the name provided to kando using `--output-name` flag
       kopiaSnapshot: "{{ .Phases.multielasticdump.Output.kopiaOutput }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
```

```
        # we don't want to capture indices starting with a dot "." like ".monitoring"
        # --match='^[^.].+'
        backup_path="dump.tgz"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=dump \
        --match='^[^.].*$' \
        --input=$ES_URL \
        --output=/tmp/es_backup
        tar -czvf dump.tgz /tmp/es_backup
        kando location push --profile '{{ toJson .Profile }}' --path $backup_path --
output-name "kopiaOutput" dump.tgz
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticrestore.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         ES_URL_CURL="https://{{ .Object.metadata.name }}-es-http:9200"
         echo "pull and untar the es artifact"
         kopia_snap='{{ .ArtifactsIn.cloudObjects.KopiaSnapshot }}'
         backup_path="dump.tgz"
```

```
        kando location pull --profile '{{ toJson .Profile }}' --path $backup_path --
kopia-snapshot "${kopia_snap}"  dump.tgz
        tar xvzf dump.tgz
        echo "Preparing es for restore"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/stop?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=true&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_stop?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/_data_stream/*?expand_wildcards=all&pretty"
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/*?expand_wildcards=all&pretty"
        echo "We are now ready to restore elasticsearch"
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=load \
        --match='^.*$' \
        --input=/tmp/es_backup \
```

```
        --output=$ES_URL
        echo "Restore successful, let's restart the service we stopped"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/start?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=false&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_start?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": null
         }
        }
        '
        sleep 4
 delete:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: deleteFromObjectStore
     args:
       namespace: "{{ .Namespace.Name }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
```

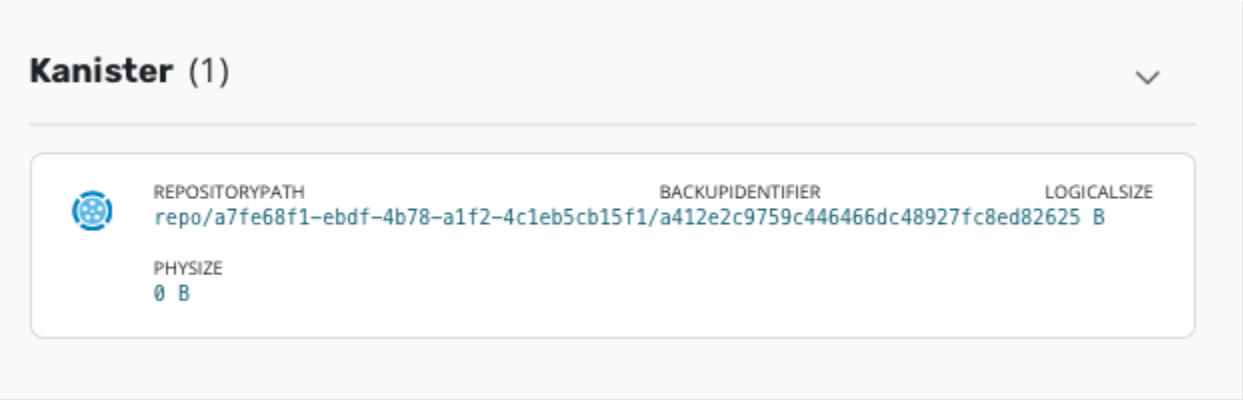
```
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        kopia_snap='{{ .ArtifactsIn.cloudObjects.KopiaSnapshot }}'
        backup_path="dump.tgz"
        kando location delete --profile '{{ toJson .Profile }}' --path
"${backup_path}"" --kopia-snapshot "${kopia_snap}"
```

Apply the annotation with the `overwrite` option:

```
k annotate -n test-es1 elasticsearch quickstart kanister.kasten.io/blueprint=elastic-
bp-v2 --overwrite
```

Note the simplified backup path, because Kasten K10 handles the path for the dump.tgz file inside the snapshot and the snapshot location: it is automatically placed with the standard URI file location of /k10/<cluster-uid>/migration/repo/<namespace-uid>.

If you check the restore point in Kasten K10, you can see the change in the Kanister artifact:




Artifacts are encrypted with a unique key, per namespace, that is solely managed by Kasten K10; you can’t read the content of dump.tgz by simply downloading the file.

## Should I include the Elasticsearch PVC in the restore point if I have a dump?

Now that you have an Elasticsearch cluster dump in the Kasten K10 snapshot, is it worth adding a copy of the Persistent Volume Claim: PVC used by Elasticsearch?

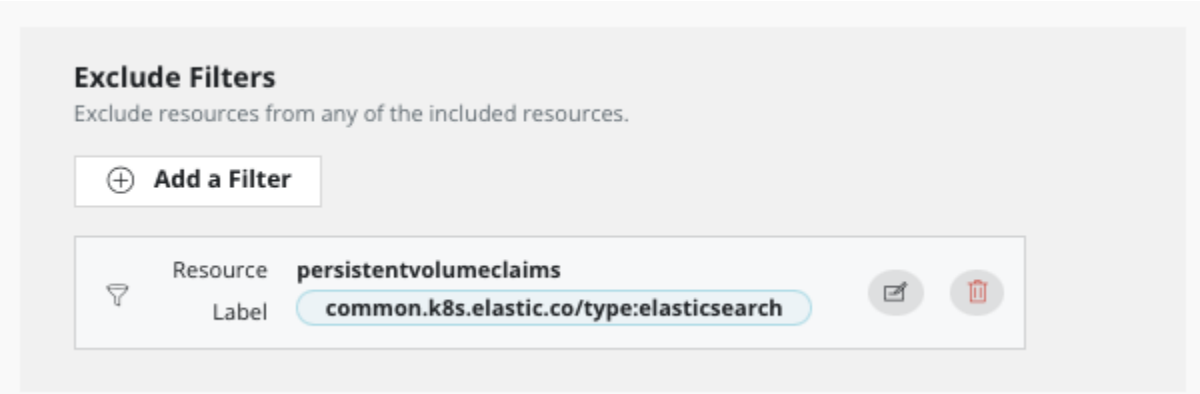
Recall Elasticsearch documentation stated a filesystem backup is not a valid backup.

From my experience, the answer would be that it's always better to have multiple copies of the data, even if some of them are not seen as 100% consistent. This is an aspect of the 3-2-1 rule. Robust backups leverage different formats (logical or storage snapshot) and different media (on the S3 or block mode). You never know where the disaster will strike, so the more locations and media you have, the safer your data protection.

There could be other considerations that would make you exclude the Elasticsearch PVCs:

- Your storage provider does not support snapshots: this is rarer today, but still exists.
    - For instance, the Netapp NAS Economy series and older, in tree vSphere drivers produce volumes that can't be snapshotted.
    - This is not an exhaustive list. Please check your storage provider's CSI driver documentation.
- You want to reduce backup operation time by saving time for the export of the PVC.
- You have very stringent storage capacity requirements or constraints on the backup location, so every byte you save counts.
    - This frequently happens when deploying workloads at edge locations.

When any of the answers above are negative, you can decide whether the Elasticsearch PVCs should not be included in the restore point with a label selector in your policy exclude filter.




If you wonder how to restore when the PVC definitions are not included with the restore point, the next section will cover this issue when dealing with an operator: it will take care of the volume re-creation, but then the restore needs to be done in a granular fashion.

# Restore

## The operator conflict

Backing up with Kasten K10 and dealing with operators is simple, but restoring with Kasten K10 is more complex than just restoring with Kanister.

Why is that?

With Kasten K10, a restore operation on a data service also rebuilds the infrastructure. For instance, when restoring in the same namespace, Kasten K10 will:

- Scale down the workloads (deployment and statefulset)
- Delete the PVCs
- Recreate the PVCs with their content at backup
- Scale up the workload

But the operator managing the quickstart Elasticsearch object will conflict with these operations:

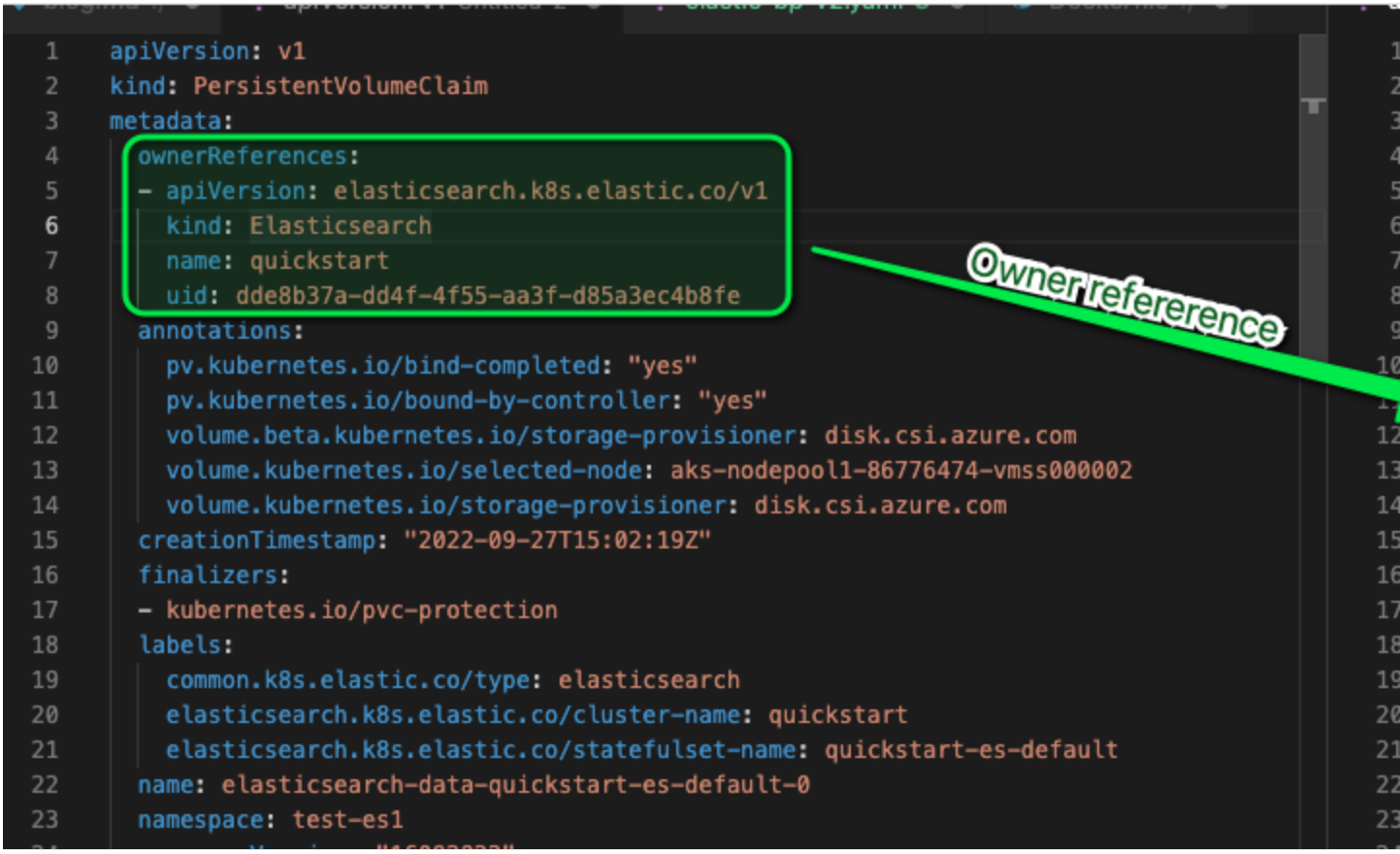
- Scale down: The elasticsearch object says two elastic instances must be up; the operator will ensure that the statefulset will always scale up to at least two instances, each with a PVC.
- Delete PVCs: For the same reason, the deletion of the two PVCs will conflict with the operator.

## Avoid the conflict by deleting before restoring

One solution could be to delete the elasticsearch object before launching the restore:

```
kubectldelete elasticsearch quickstart -n test-es1
```

Because of the ownership relation between quickstart and its dependencies (statefulsets, services, secrets, PVC…), all of these dependencies will be deleted when quickstart is deleted.




```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  ownerReferences:
  - apiVersion: elasticsearch.k8s.elastic.co/v1
    kind: Elasticsearch
    name: quickstart
    uid: dde8b37a-dd4f-4f55-aa3f-d85a3ec4b8fe
  annotations:
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: disk.csi.azure.com
    volume.kubernetes.io/selected-node: aks-nodepool1-86776474-vmss000002
    volume.kubernetes.io/storage-provisioner: disk.csi.azure.com
  creationTimestamp: "2022-09-27T15:02:19Z"
  finalizers:
  - kubernetes.io/pvc-protection
  labels:
    common.k8s.elastic.co/type: elasticsearch
    elasticsearch.k8s.elastic.co/cluster-name: quickstart
    elasticsearch.k8s.elastic.co/statefulset-name: quickstart-es-default
  name: elasticsearch-data-quickstart-es-default-0
  namespace: test-es1
```

Even if the operator controller is in the same namespace as the custom resource, Elasticsearch object deletion will work, because Kasten K10 first restores the workloads before restoring the custom resource. Hence, the operator controller will be up and running before the custom resource will be recreated.

## But if there is a blueprint, we must wait!

If there is a blueprint attached to the custom resource, we may have to wait before launching the restore, because the blueprint will be executed just after the recreation of the Elasticsearch cluster resource with no guarantee the Elasticsearch cluster is ready. So we have to add a Wait condition to insure the cluster is ready.

The operator is the source of truth to determine if the cluster is ready, so a new version of our blueprint will accommodate it:

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp-v2
 namespace: kasten-io
actions:
```

```
 backup:
   outputArtifacts:
     cloudObjects:
       # Capture the kopia snapshot information for subsequent actions
       # The information includes the kopia snapshot ID which is essential for restore
and delete to succeed
       # `kopiaOutput` is the name provided to kando using `--output-name` flag
       kopiaSnapshot: "{{ .Phases.multielasticdump.Output.kopiaOutput }}"
   phases:
   - func: KubeTask
     name: multielasticdump
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticdump.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
         mkdir /tmp/es_backup
         # we don't want to capture indices starting with a dot "." like ".monitoring"
         # --match='^[^.].+'
         backup_path="dump.tgz"
         NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
         --direction=dump \
         --match='^[^.].*$' \
         --input=$ES_URL \
         --output=/tmp/es_backup
         tar -czvf dump.tgz /tmp/es_backup
         kando location push --profile '{{ toJson .Profile }}' --path $backup_path --
```

```
output-name "kopiaOutput" dump.tgz
        sleep 4
 restore:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: Wait
     name: waitElasticGreen
     args:
       timeout: 360s
       conditions:
         anyOf:
           - condition: '{{ if  eq "{ $.status.health }" "green" }}true{{ else
}}false{{ end }}'
             objectReference:
               apiVersion: v1
               group: elasticsearch.k8s.elastic.co
               resource: elasticsearches
               name: "{{ .Object.metadata.name }}"
               namespace: "{{ .Object.metadata.namespace}}"
   - func: KubeTask
     name: multielasticrestore
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       namespace: "{{ .Object.metadata.namespace }}"
       image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         PASSWORD="{{ index .Phases.multielasticrestore.Secrets.elasticSecret.Data
"elastic" | toString }}"
         ES_URL="https://elastic:$PASSWORD@{{ .Object.metadata.name }}-es-http:9200"
```

```
        ES_URL_CURL="https://{{ .Object.metadata.name }}-es-http:9200"
        echo "pull and untar the es artifact"
        kopia_snap='{{ .ArtifactsIn.cloudObjects.KopiaSnapshot }}'
        backup_path="dump.tgz"
        kando location pull --profile '{{ toJson .Profile }}' --kopia-snapshot
"${kopia_snap}" --path $backup_path dump.tgz
        tar xvzf dump.tgz
        echo "Preparing es for restore"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/stop?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=true&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_stop?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": false
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/_data_stream/*?expand_wildcards=all&pretty"
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL_CURL}/*?expand_wildcards=all&pretty"
        echo "We are now ready to restore Elasticsearch"
```

```
        NODE_TLS_REJECT_UNAUTHORIZED=0 multielasticdump \
        --direction=load \
        --match='^.*$' \
        --input=/tmp/es_backup \
        --output=$ES_URL
        echo "Restore successful, let's restart the service we stopped"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "ingest.geoip.downloader.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL_CURL}/_ilm/start?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_ml/set_upgrade_mode?enabled=false&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "xpack.monitoring.collection.enabled": true
         }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL_CURL}/_watcher/_start?pretty"
        curl  -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL_CURL}/_cluster/settings?pretty" -H 'Content-Type: application/json' -d'
        {
         "persistent": {
           "action.destructive_requires_name": null
         }
        }
        '
        sleep 4
 delete:
   inputArtifactNames:
   - cloudObjects
   phases:
   - func: KubeTask
     name: deleteFromObjectStore
```

```
    args:
      namespace: "{{ .Namespace.Name }}"
      image: michaelcourcy/elasticdump-kanister:v6.88.0-0.81.0
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        kopia_snap='{{ .ArtifactsIn.cloudObjects.KopiaSnapshot }}'
        backup_path="dump.tgz"
        kando location delete --profile '{{ toJson .Profile }}' --path
"${backup_path}"" --kopia-snapshot "${kopia_snap}"
```

The configuration of the wait function is straightforward. Please see the [Kanister documentation] for more details.

Now we are ready to test the restore. First, delete the Elasticsearch object:

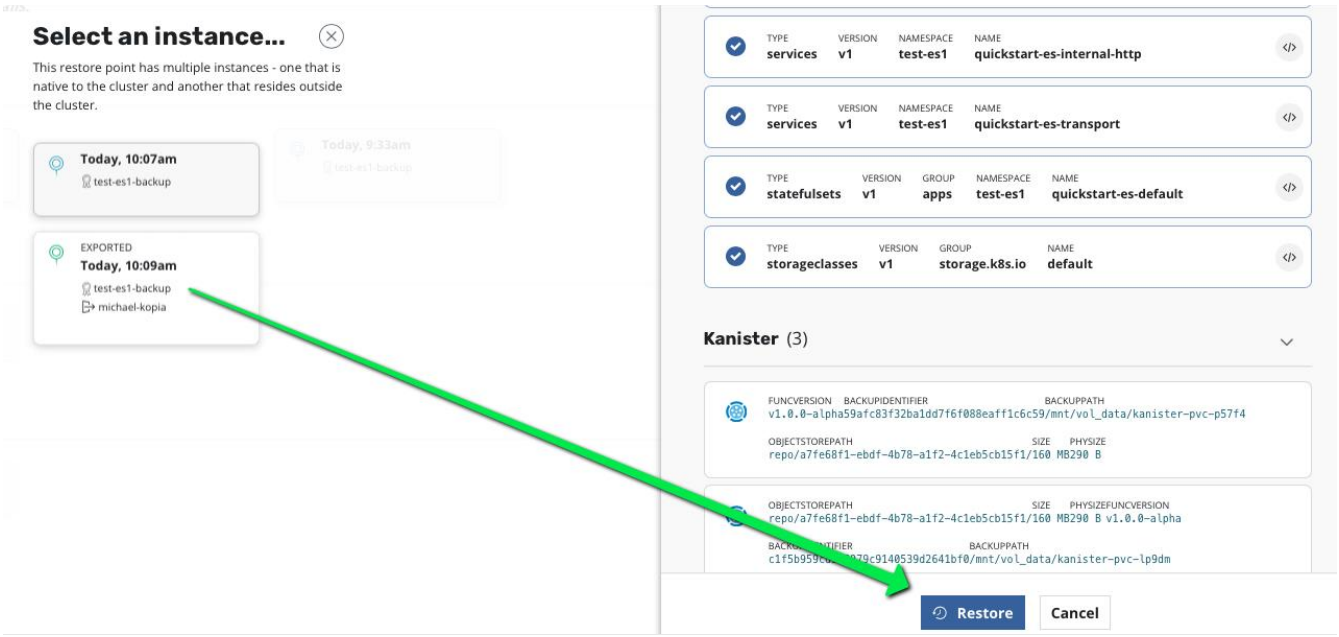
```
kubectldelete elasticsearch quickstart -n test-es1
```

Check that all the resources depending on this object were removed:

```
kubectlget pvc -n test-es1
No resources found in test-es1 namespace.
k get sts -n test-es1
No resources found in test-es1 namespace.
kubectlget svc -n test-es1
No resources found in test-es1 namespace.
kubectlget secret -n test-es1
NAME                  TYPE                                  DATA   AGE
default-token-tlnqs   kubernetes.io/service-account-token   3      8d
s3-secret-c2lcgj      Opaque                                3      6d21h
```

Great, now we can restore!

Find your last restore point and click restore:




Check the execution of the restore pod by running a while loop, similar to when we previously were troubleshooting Kanister:

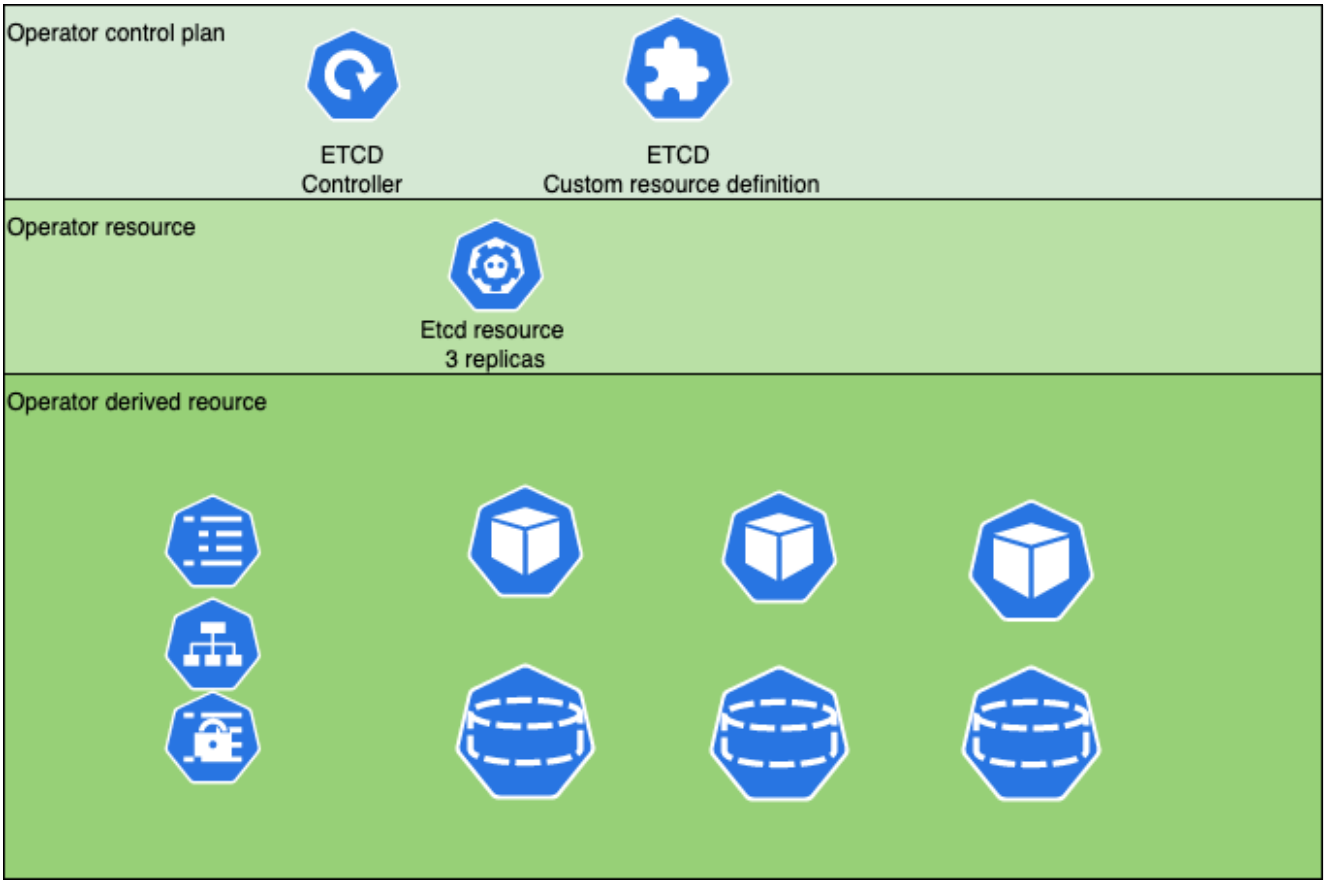
```
while true; do kubectl  -n test-es1 logs -f -l createdBy=kanister; sleep 2; done
```

## Blueprint is sometime the only solution for restore

With Kubernetes operators, sometimes a blueprint on the custom resource is the only solution for restoring. In our Elasticsearch example, the operator is robust, so when the operator starts to reconcile the custom resource with the object restored by Kasten K10, it deals properly with the existing object instead of trying to recreate them.

What do I mean by a “robust operator?” It is an operator that can reconcile with existing resources. This is not the case for the etcd operator, which at the time of publishing this article, always creates resources that it did not create.

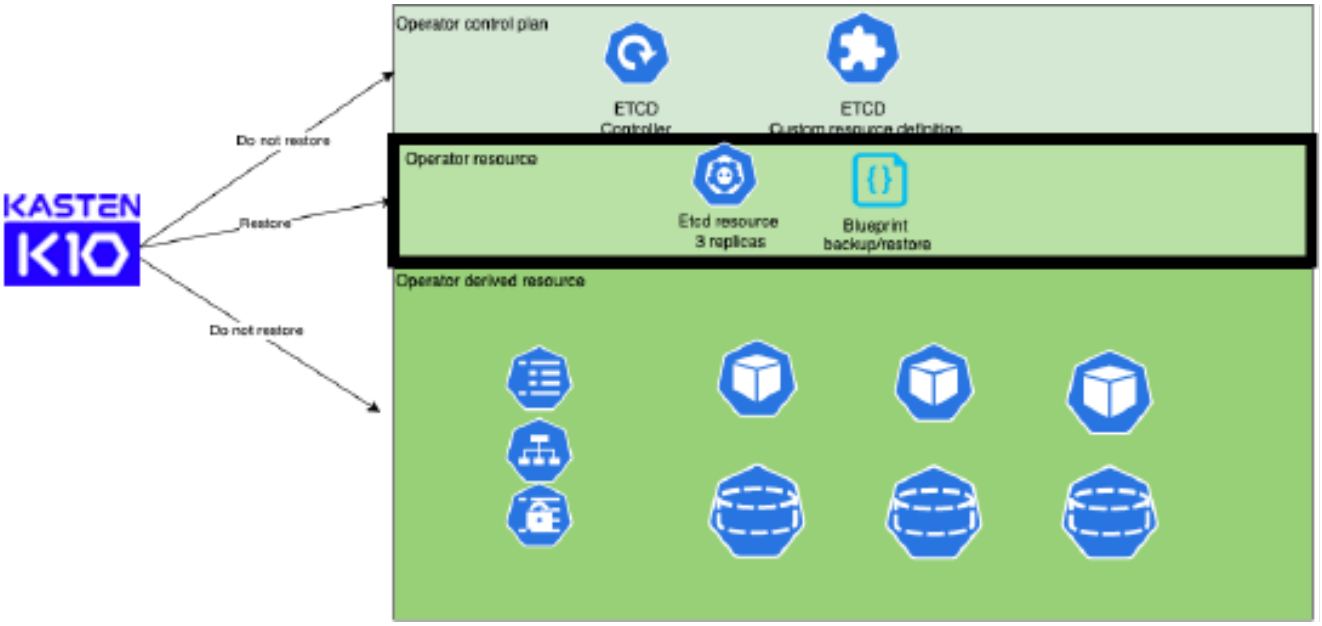
To understand that in more detail, here is a diagram that describes the etcd operator in a cluster:




- First layer – the operator control plane: This is the API server that will handle the creation of Etcd Custom Resource based on the ETCD custom resource definition.
- Second Layer – the Operator resource: This is the custom resource to use for defining the ETCD cluster specificity (for instance 3 replicas).
- Third layer – the Operator derived resource: This is the resource created by the operator to satisfy the desired state defined in layer 2.

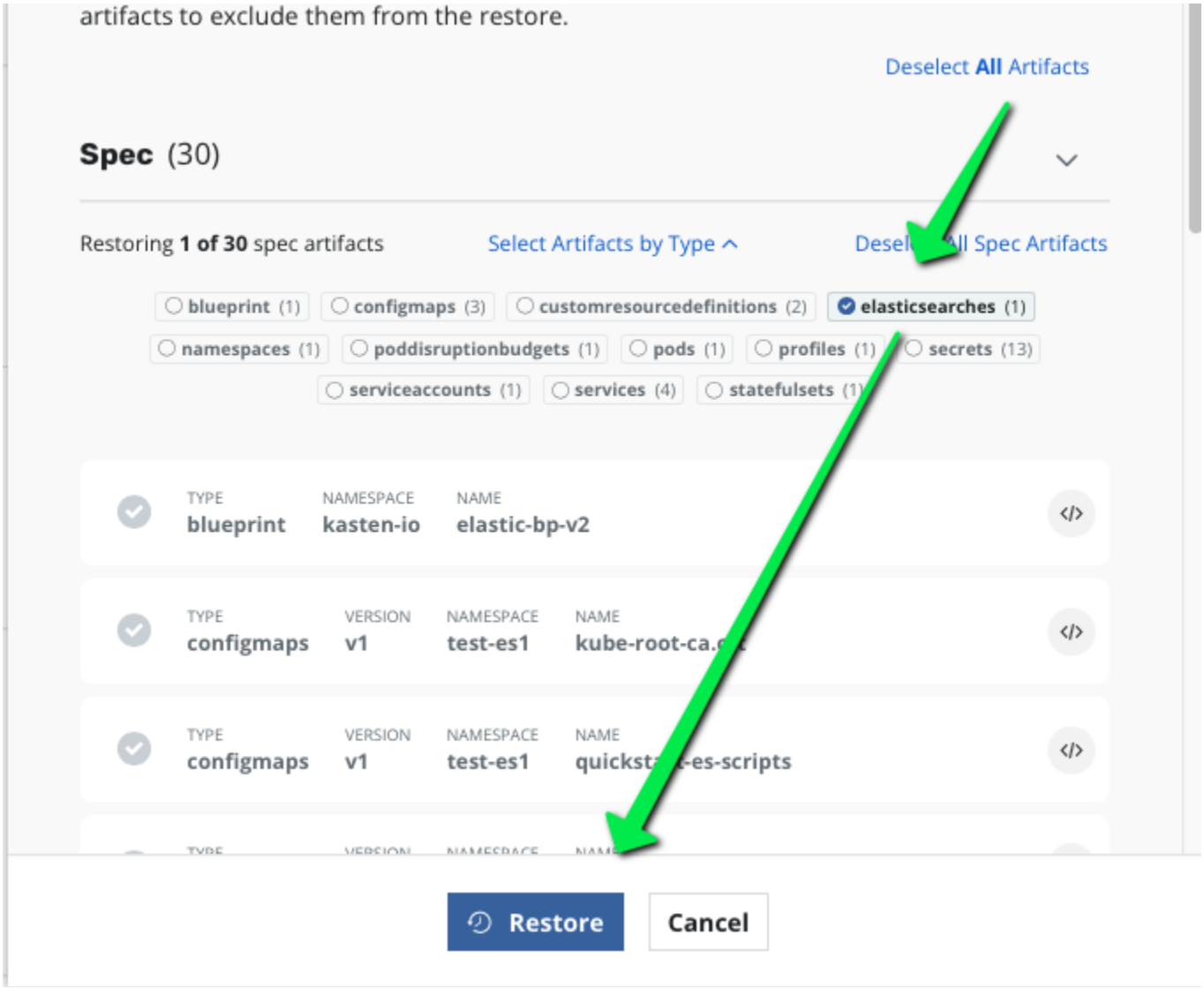
If Kasten K10 restores the etcd operator derived resource, the etcd operator will see that it did not create them. It will delete them, and recreate them, and you'll lose all your data (Note that this behavior is specific to the etcd operator it is not true for the elasticsearch cluster).

In this case, your only option is to restore the operator resource and filter out the derived resources. Thanks to a blueprint that implements etcddump and etcdrestore, you'll also get back all your data.




That’s why having a blueprint in conjunction with granular restore is the only way to fully restore.

We can do that also for Elasticsearch. In this screenshot, only the Elasticsearch resource is restored.




It's also worth to mention that when restoring to another namespace, some derived resource may hard code the namespace name in the configmap, the secrets, or even the data service data. In this case restoring in another namespace won't work.

But if you restore only the Elasticsearch object, then the operator controller will recreate everything properly, and when the cluster is ready, the Kanister blueprint can restore all the data.

# Creating a predictable RPO

## The solution may not work for all use cases

This approach is reliable for creating a predictable Recovery Point Objective (RPO) as long as the global size of the Elasticsearch object is not growing and follows the best practice with

periodic curation. In this case, the time for executing multielasticdump and sending the dump over the network is fairly predictable and constant.

However, Elasticsearch supports horizontal scalability, even if the size of the cluster eventually becomes stable, it may be constant and huge at the same time. I have encountered clusters with more than 20 PVCs!

If you're interested in backing up only some of the indices, then the blueprint example is still relevant. However you must adapt the blueprint to choose the indices with the proper regexp. If you don't know what is useful (and what to exclude), you will want to backup everything. The example blueprint solution is not relevant anymore, because multielasticdump will overflow your network due to the huge amount of data, and cause an unacceptable RPO.

## Becoming incremental

Having an incremental backup will minimize the backup duration. If backing up the PVCs was sufficient with Elasticsearch, then backups automatically become incremental, but this is not supported for Elasticsearch.

Elasticsearch internally works with segments, and each segment is immutable. This exercise does not go further into Elasticsearch internals, but Elasticsearch has a Snapshot API that sends only the new segments to a storage location. Hence, the Elasticsearch Snapshot API is incremental by design.

## Why not use the Elasticsearch Snapshot API from the beginning?

You may wonder, why did we not use the Elasticsearch snapshot API instead of the `elasticdump` approach? The reason is because we leverage a data mover for portability and security. Furthermore, a Kanister blueprint represents an external artifact for your organization's operations, which can be reused by multiple teams and roles, and can foster separation of duties based on governance needs. Finally, in the case of disaster recovery or a heavily loaded or impaired Elasticsearch cluster, you should restore independently of the existing cluster.

When using Elasticsearch Snapshot API, Elasticsearch is in charge of moving the data from the cluster to the backup location. That has serious consequences on blueprint portability and creates security concerns:

- The backup profile must now be declared in Elasticsearch, losing portability by making what was an external concern internal to Elasticsearch.
- The backup location in Elasticsearch can be declared programmatically from the blueprint, but the code will vary depending on the profile type, creating complexity while reducing portability.

- Declaring the backup location in the Elasticsearch deployment and calling the snapshot API from the blueprint, but changing the profile in the policy will have no effect. Also, it requires  that we expose the backup location credential in the namespace where the cluster is deployed.
- Kasten K10 does not control the encryption process. Elasticsearch snapshot API does not support encryption. Data is sent in nearly plain text to the backup location.
- Snapshot lifecycle management: The deletion of a snapshot requires your Elasticsearch cluster to be up and running by calling its API.

## But we may not have the choice and need to adapt

If the cluster is large and you cannot take the risk of the inconsistencies with PVC snapshots, you must accept some limitations. By adapting and implementing https encryption in transportation for the backup location (a standard feature of all S3 solutions), you will not sacrifice transport security, creating a security burden for TLS certificate and S3 bucket key encryption management.

## An example blueprint that should be improved

This example Elasticsearch Kanister blueprint could be improved to support more profile types. This blueprint only supports S3 profile type portability and the security burdens, explained above.

However, this blueprint has unbeatable performance on large Elasticsearch clusters with a very short RPO!

This exercise shows you the power and flexibility of Kanister blueprint extensibility. Please note that this blueprint is not supported by Kasten. In other words, use it at your own risk. I recommend that you implement performance and consistency tests to improve your blueprint usage confidence.

To understand Elasticsearch tool that operations leveraged in the blueprint further, consult the Elasticsearch documentation:

- https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshot-restore.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshots-register-repository.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/repository-s3.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/repo-analysis-api.html
- https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshots-restore-snapshot.html#restore-different-cluster

- https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshots-restore-snapshot.html#restore-different-cluster

Congratulations, you have the skills to understand how a Kanister blueprint works! The complete blueprint follows:

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
 name: elastic-bp-s3
 namespace: kasten-io
actions:
 backup:
   outputArtifacts:
     s3Snap:
       keyValue:
         repoPath: '{{ .Phases.setupPhase.Output.repoPath }}'
         snapshotName: '{{ .Phases.setupPhase.Output.snapshotName }}'
         esClusterName: '{{ .Phases.setupPhase.Output.esClusterName }}'
         esClusterNamespace: '{{ .Object.metadata.namespace }}'
   phases:
   - func: KubeTask
     name: setupPhase
     objects:
       elasticSecret:
         kind: Secret
         name: '{{ .Object.metadata.name }}-es-elastic-user'
         namespace: '{{ .Object.metadata.namespace }}'
     args:
       image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
       command:
       - bash
       - -x
       - -o
       - errexit
       - -o
       - pipefail
       - -c
       - |
         debug=0
         function dbg
         {
```

```
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
            fi
          }
          function bp_echo
          {
            # use
            # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
            # to grab the logs in kanisters
            echo "Elasticbps3echobegin backup-action: $* Elasticbps3echoend"
          }
          PROFILE_TYPE="{{ .Profile.Location.Type }}"
          if [[ $PROFILE_TYPE != "s3Compliant" ]]
          then
            dbg "Only s3Compliant profile are supported, exiting"
            exit 1
          fi
          CLUSTER_UID=$(kubectl get ns default -o jsonpath='{.metadata.uid}')
          NS_UID=$(kubectl get ns {{ .Object.metadata.namespace }} -o
jsonpath='{.metadata.uid}')
          ES_CLUSTER_UID={{ .Object.metadata.uid }}
          REPO_PATH=k10/$CLUSTER_UID/elastic-search/$NS_UID/$ES_CLUSTER_UID
          SNAPSHOT_NAME="snap_{{ toDate "2006-01-02T15:04:05.999999999Z07:00" .Time |
date "2006-01-02t15:04:05z07:00" }}"
          bp_echo "output the snapshot info $SNAPSHOT_NAME"
          kando output repoPath $REPO_PATH
          kando output snapshotName $SNAPSHOT_NAME
          kando output esClusterName {{ .Object.metadata.name }}

          # setup the credentials on the keystore setting of each nodes
          {{- if .Profile.Credential.KeyPair }}
          AWS_SECRET_KEY="{{ .Profile.Credential.KeyPair.Secret }}"
          AWS_ACCESS_KEY="{{ .Profile.Credential.KeyPair.ID }}"
          {{- else }}
          AWS_SECRET_KEY="{{ .Profile.Credential.Secret.Data.aws_secret_access_key |
toString }}"
          AWS_ACCESS_KEY="{{ .Profile.Credential.Secret.Data.aws_access_key_id |
toString }}"
          {{- end }}
          # No Support for more than one node set for the moment
```

```
        NODE_SET_COUNT={{ (index .Object.spec.nodeSets 0).count }}
        NODE_SET_COUNT=$((NODE_SET_COUNT-1))
        NODE_SET_NAME={{ (index .Object.spec.nodeSets 0).name }}

        for i in $(seq 0 $NODE_SET_COUNT)
        do
            pod="{{ .Object.metadata.name }}-es-$NODE_SET_NAME-$i"
            bp_echo "updating elasticsearch-keystore of $pod"
            kubectl exec -it -n {{ .Object.metadata.namespace }} $pod -c
elasticsearch -- bash -c "echo ${AWS_ACCESS_KEY} |
/usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.access_key"
            kubectl exec -it -n {{ .Object.metadata.namespace }} $pod -c
elasticsearch -- bash -c "echo ${AWS_SECRET_KEY} |
/usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.secret_key"
        done
        # make sure all output are grabbed by kanister pod
        sleep 4
  - func: KubeTask
    name: snapshotElastic
    args:
      namespace: "{{ .Object.metadata.namespace }}"
      image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        debug=0
        function dbg
        {
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
            fi
        }
        function bp_echo
```

```
        {
          # use
          # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
          # to grab the logs in kanisters
          echo "Elasticbps3echobegin backup-action: $* Elasticbps3echoend"
        }
        ES_URL="https://{{ .Object.metadata.name }}-es-http:9200"
        PASSWORD="{{ index .Phases.setupPhase.Secrets.elasticSecret.Data "elastic" |
toString }}"
        REGION="{{ .Profile.Location.Region }}"
        BUCKET="{{ .Profile.Location.Bucket }}"
        ENDPOINT="{{ .Profile.Location.Endpoint }}"
        if [[ -z $ENDPOINT ]]
        then
           ENDPOINT="s3.amazonaws.com"
        fi
        REPO_PATH="{{ .Phases.setupPhase.Output.repoPath }}"
        SNAPSHOT_NAME="{{ .Phases.setupPhase.Output.snapshotName }}"
        dbg "REGION=$REGION"
        dbg "BUCKET=$BUCKET"
        dbg "ENDPOINT=$ENDPOINT"
        dbg "PASSWORD=$PASSWORD"
        dbg "PROFILE_TYPE=$PROFILE_TYPE"
        dbg "REPO_PATH=$REPO_PATH"
        dbg "SNAPSHOT_NAME=$SNAPSHOT_NAME"
        # reload the secure settings to access the S3 profile
        curl -k -u "elastic:$PASSWORD" -X POST
"${ES_URL}/_nodes/reload_secure_settings?pretty" -H 'Content-Type: application/json' -
d'
        {}
        '
        bp_echo "Creating the repo"
        curl -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_snapshot/k10_repo?pretty" -
H 'Content-Type: application/json' -d'
        {
          "type": "s3",
          "settings": {
            "bucket": "'$BUCKET'",
            "endpoint": "'$ENDPOINT'",
            "region": "'$REGION'",
            "base_path": "'$REPO_PATH'"
```

```
          }
        }
        '
        bp_echo "creating the snap $SNAPSHOT_NAME"
        curl -k -u "elastic:$PASSWORD" -X PUT
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME?pretty"

        while curl -k -u "elastic:$PASSWORD" -X GET
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME/_status?pretty" | grep -P
"(IN_PROGRESS|STARTED)"
        do
          bp_echo "snapshot $SNAPSHOT_NAME still in progress"
          size_in_bytes=$(curl -k -u "elastic:$PASSWORD" -X GET
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME/_status?pretty" |grep size_in_bytes)
          bp_echo $size_in_bytes
          sleep 4
        done

        if  curl -k -u "elastic:$PASSWORD" -X GET
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME/_status?pretty" | grep SUCCESS
        then
           bp_echo "snapshot $SNAPSHOT_NAME was successful"
        else
           bp_echo "snapshot $SNAPSHOT_NAME was not successful"
           reason=$(curl -k -u "elastic:$PASSWORD" -X GET
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME/_status?pretty")
           bp_echo $reason
           # make sure all output are grabbed by kanister pod
           sleep 4
           exit 1
        fi
        # make sure all output are grabbed by kanister pod
        sleep 4
 restore:
   inputArtifactNames:
   - s3Snap  # use it with  for instance {{ .ArtifactsIn.s3Snap.KeyValue.repoPath }}
or {{ .ArtifactsIn.s3Snap.KeyValue.snapshotName }}
   phases:
   - func: Wait
     name: waitElasticGreen
     args:
       timeout: 360s
```

```
      conditions:
        anyOf:
          - condition: '{{ if  or (eq "{ $.status.health }" "green") (eq "{
$.status.health }" "yellow")}}true{{ else }}false{{ end }}'
            objectReference:
              apiVersion: v1
              group: elasticsearch.k8s.elastic.co
              resource: elasticsearches
              name: "{{ .Object.metadata.name }}"
              namespace: "{{ .Object.metadata.namespace}}"
  - func: ScaleWorkload
    name: ensureSTSIsScaled
    args:
      namespace: "{{ .Object.metadata.namespace}}"
      kind: statefulset
      name: "{{ .Object.metadata.name }}-es-{{ (index .Object.spec.nodeSets 0).name
}}"
      replicas: "{{ (index .Object.spec.nodeSets 0).count }}"
  - func: KubeTask
    name: setupPhase
    objects:
      elasticSecret:
        kind: Secret
        name: '{{ .Object.metadata.name }}-es-elastic-user'
        namespace: '{{ .Object.metadata.namespace }}'
    args:
      image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        debug=0
        function dbg
        {
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
```

```
            fi
        }
        function bp_echo
        {
          # use
          # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
          # to grab the logs in kanisters
          echo "Elasticbps3echobegin restore-action: $* Elasticbps3echoend"
        }
        PROFILE_TYPE="{{ .Profile.Location.Type }}"
        if [[ $PROFILE_TYPE != "s3Compliant" ]]
        then
          dbg "Only s3Compliant profile are supported, exiting"
          exit 1
        fi
        # setup the credentials on the keystore setting of each nodes
        {{- if .Profile.Credential.KeyPair }}
        AWS_SECRET_KEY="{{ .Profile.Credential.KeyPair.Secret }}"
        AWS_ACCESS_KEY="{{ .Profile.Credential.KeyPair.ID }}"
        {{- else }}
        AWS_SECRET_KEY="{{ .Profile.Credential.Secret.Data.aws_secret_access_key |
toString }}"
        AWS_ACCESS_KEY="{{ .Profile.Credential.Secret.Data.aws_access_key_id |
toString }}"
        {{- end }}
        # Support for more than one node set is not supported for the moment
        NODE_SET_COUNT={{ (index .Object.spec.nodeSets 0).count }}
        NODE_SET_COUNT=$((NODE_SET_COUNT-1))
        NODE_SET_NAME={{ (index .Object.spec.nodeSets 0).name }}
        for i in $(seq 0 $NODE_SET_COUNT)
        do
            pod="{{ .Object.metadata.name }}-es-$NODE_SET_NAME-$i"
            bp_echo "updating elasticsearch-keystore of $pod"
            kubectl exec -it -n {{ .Object.metadata.namespace }} $pod -c
elasticsearch -- bash -c "echo ${AWS_ACCESS_KEY} |
/usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.access_key"
            kubectl exec -it -n {{ .Object.metadata.namespace }} $pod -c
elasticsearch -- bash -c "echo ${AWS_SECRET_KEY} |
/usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.secret_key"
```

```
        done
        # make sure all output are grabbed by kanister pod
        sleep 4
  - func: KubeTask
    name: restoreElastic
    args:
      namespace: "{{ .Object.metadata.namespace }}"
      image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        debug=0
        function dbg
        {
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
            fi
        }
        function bp_echo
        {
          # use
          # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
          # to grab the logs in kanisters
          echo "Elasticbps3echobegin restore-action: $* Elasticbps3echoend"
        }
        ES_URL="https://{{ .Object.metadata.name }}-es-http:9200"
        PASSWORD="{{ index .Phases.setupPhase.Secrets.elasticSecret.Data "elastic" |
toString }}"
        REGION="{{ .Profile.Location.Region }}"
        BUCKET="{{ .Profile.Location.Bucket }}"
        ENDPOINT="{{ .Profile.Location.Endpoint }}"
        if [[ -z $ENDPOINT ]]
        then
           ENDPOINT="s3.amazonaws.com"
```

```
        fi
        REPO_PATH="{{ .ArtifactsIn.s3Snap.KeyValue.repoPath }}"
        SNAPSHOT_NAME="{{ .ArtifactsIn.s3Snap.KeyValue.snapshotName }}"
        dbg "REGION=$REGION"
        dbg "BUCKET=$BUCKET"
        dbg "ENDPOINT=$ENDPOINT"
        dbg "PASSWORD=$PASSWORD"
        dbg "PROFILE_TYPE=$PROFILE_TYPE"
        dbg "REPO_PATH=$REPO_PATH"
        dbg "SNAPSHOT_NAME=$SNAPSHOT_NAME"

        # reload the secure settings to access the S3 profile
        curl -k -u "elastic:$PASSWORD" -X POST
"${ES_URL}/_nodes/reload_secure_settings?pretty" -H 'Content-Type: application/json' -
d'
        {}
        '

        # create the repo
        curl -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_snapshot/k10_repo?pretty" -
H 'Content-Type: application/json' -d'
        {
          "type": "s3",
          "settings": {
            "bucket": "'$BUCKET'",
            "endpoint": "'$ENDPOINT'",
            "region": "'$REGION'",
            "base_path": "'$REPO_PATH'"
          }
        }
        '

        bp_echo "stop service geoip, machine learning, monitoring and watcher"
        curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty"
-H 'Content-Type: application/json' -d'
        {
          "persistent": {
            "ingest.geoip.downloader.enabled": false
          }
        }
        '
```

```
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_ilm/stop?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL}/_ml/set_upgrade_mode?enabled=true&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty"
-H 'Content-Type: application/json' -d'
        {
          "persistent": {
            "xpack.monitoring.collection.enabled": false
          }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_watcher/_stop?pretty"

        bp_echo "delete all datastreams and indices"
        curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_cluster/settings?pretty" -
H 'Content-Type: application/json' -d'
        {
          "persistent": {
            "action.destructive_requires_name": false
          }
        }
        '
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL}/_data_stream/*?expand_wildcards=all&pretty"
        curl  -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL}/*?expand_wildcards=all&pretty"

        bp_echo "restore all"
        # TODO manage the monitoring of the restoration process
        curl  -k -u "elastic:$PASSWORD" -X POST
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME/_restore?pretty" -H 'Content-Type:
application/json' -d'
        {
          "indices": "*",
          "include_global_state": true
        }
        '

        bp_echo "Restart service geoip, machine learning, monitoring and watcher"
        curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty"
-H 'Content-Type: application/json' -d'
        {
```

```
          "persistent": {
            "ingest.geoip.downloader.enabled": true
          }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_ilm/start?pretty"
        curl  -k -u "elastic:$PASSWORD"  -X POST
"${ES_URL}/_ml/set_upgrade_mode?enabled=false&pretty"
        curl  -k -u "elastic:$PASSWORD"  -X PUT "${ES_URL}/_cluster/settings?pretty"
-H 'Content-Type: application/json' -d'
        {
          "persistent": {
            "xpack.monitoring.collection.enabled": true
          }
        }
        '
        curl  -k -u "elastic:$PASSWORD"  -X POST "${ES_URL}/_watcher/_start?pretty"

        bp_echo "Reenable destructive_requires_name"
        curl  -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_cluster/settings?pretty" -
H 'Content-Type: application/json' -d'
        {
          "persistent": {
            "action.destructive_requires_name": null
          }
        }
        '
        # make sure all output are grabbed by kanister pod
        sleep 4
 # limitation : deletion need a cluster with the same name in the same namespace be up
and running
 # if you don't have that then just restore it with kasten or create it manually and
proceed the deletions of the restorepoint
 delete:
   inputArtifactNames:
   # use it with  for instance {{ .ArtifactsIn.s3Snap.KeyValue.repoPath }} {{
.ArtifactsIn.s3Snap.KeyValue.snapshotName }}
   #                          {{ .ArtifactsIn.s3Snap.KeyValue.esClusterName }} {{
.ArtifactsIn.s3Snap.KeyValue.esClusterNamespace  }}
   - s3Snap
   phases:
   - func: KubeTask
```

```
    name: setupPhase
    objects:
      elasticSecret:
        kind: Secret
        name: '{{ .ArtifactsIn.s3Snap.KeyValue.esClusterName }}-es-elastic-user'
        namespace: '{{ .ArtifactsIn.s3Snap.KeyValue.esClusterNamespace }}'
    args:
      image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        debug=0
        function dbg
        {
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
            fi
        }
        function bp_echo
        {
          # use
          # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
          # to grab the logs in kanisters
          echo "Elasticbps3echobegin delete-action: $* Elasticbps3echoend"
        }
        PROFILE_TYPE="{{ .Profile.Location.Type }}"
        if [[ $PROFILE_TYPE != "s3Compliant" ]]
        then
          dbg "Only s3Compliant profile are supported, exiting"
          exit 1
        fi
        # setup the credentials on the keystore setting of each nodes
        {{- if .Profile.Credential.KeyPair }}
        AWS_SECRET_KEY="{{ .Profile.Credential.KeyPair.Secret }}"
```

```
        AWS_ACCESS_KEY="{{ .Profile.Credential.KeyPair.ID }}"
        {{- else }}
        AWS_SECRET_KEY="{{ .Profile.Credential.Secret.Data.aws_secret_access_key |
toString }}"
        AWS_ACCESS_KEY="{{ .Profile.Credential.Secret.Data.aws_access_key_id |
toString }}"
        {{- end }}
        NAMESPACE="{{ .ArtifactsIn.s3Snap.KeyValue.esClusterNamespace }}"
        ES_CLUSTER_NAME="{{ .ArtifactsIn.s3Snap.KeyValue.esClusterName }}"
        # exit with an error if cluster does not exist
        if kubectl get elasticsearch -n $NAMESPACE $ES_CLUSTER_NAME
        then
          bp_echo "we have an es cluster $ES_CLUSTER_NAME in ns $NAMESPACE"
        else
          bp_echo "we don't have an es cluster $ES_CLUSTER_NAME in ns $NAMESPACE"
          sleep 4
          exit 1
        fi

        NODE_SET_NAME=$(kubectl get elasticsearch $ES_CLUSTER_NAME -n $NAMESPACE -o
jsonpath='{.spec.nodeSets[0].name}')
        NODE_SET_COUNT=$(kubectl get elasticsearch $ES_CLUSTER_NAME -n $NAMESPACE -o
jsonpath='{.spec.nodeSets[0].count}')
        NODE_SET_COUNT=$((NODE_SET_COUNT-1))
        for i in $(seq 0 $NODE_SET_COUNT)
        do
            pod="$ES_CLUSTER_NAME-es-$NODE_SET_NAME-$i"
            bp_echo "updating elasticsearch-keystore of $pod"
            kubectl exec -it -n $NAMESPACE $pod -c elasticsearch -- bash -c "echo
${AWS_ACCESS_KEY} | /usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.access_key"
            kubectl exec -it -n $NAMESPACE $pod -c elasticsearch -- bash -c "echo
${AWS_SECRET_KEY} | /usr/share/elasticsearch/bin/elasticsearch-keystore add --stdin -f
s3.client.default.secret_key"
        done
        # make sure all output are grabbed by kanister pod
        sleep 4
    - func: KubeTask
      name: deleteSnapshot
      args:
        namespace: '{{ .ArtifactsIn.s3Snap.KeyValue.esClusterNamespace }}'
        image: ghcr.io/kanisterio/kanister-kubectl-1.18:0.81.0
```

```
      command:
      - bash
      - -x
      - -o
      - errexit
      - -o
      - pipefail
      - -c
      - |
        debug=0
        function dbg
        {
            if [[ $debug -eq 1 ]]
            then
              echo "$(date "+%Y-%m-%d %H:%M:%S.%3N") - $0 - $*" >> /tmp/k10_debug.log
            fi
        }
        function bp_echo
        {
          # use
          # kubectl logs -n kasten-io -l component=kanister --tail=10000 -f | ggrep -
o -P '(?<=Elasticbps3echobegin).*(?=Elasticbps3echoend)'
          # to grab the logs in kanisters
          echo "Elasticbps3echobegin delete-action: $* Elasticbps3echoend"
        }
        ES_URL="https://{{ .ArtifactsIn.s3Snap.KeyValue.esClusterName }}-es-
http:9200"
        PASSWORD="{{ index .Phases.setupPhase.Secrets.elasticSecret.Data "elastic" |
toString }}"
        REGION="{{ .Profile.Location.Region }}"
        BUCKET="{{ .Profile.Location.Bucket }}"
        ENDPOINT="{{ .Profile.Location.Endpoint }}"
        if [[ -z $ENDPOINT ]]
        then
           ENDPOINT="s3.amazonaws.com"
        fi
        REPO_PATH="{{ .ArtifactsIn.s3Snap.KeyValue.repoPath }}"
        SNAPSHOT_NAME="{{ .ArtifactsIn.s3Snap.KeyValue.snapshotName }}"
        dbg "REGION=$REGION"
        dbg "BUCKET=$BUCKET"
        dbg "ENDPOINT=$ENDPOINT"
        dbg "PASSWORD=$PASSWORD"
```

```
        dbg "PROFILE_TYPE=$PROFILE_TYPE"
        dbg "REPO_PATH=$REPO_PATH"
        dbg "SNAPSHOT_NAME=$SNAPSHOT_NAME"

        # reload the secure settings to access the S3 profile
        curl -k -u "elastic:$PASSWORD" -X POST
"${ES_URL}/_nodes/reload_secure_settings?pretty" -H 'Content-Type: application/json' -
d'
        {}
        '

        bp_echo "create the repo"
        curl -k -u "elastic:$PASSWORD" -X PUT "${ES_URL}/_snapshot/k10_repo?pretty" -
H 'Content-Type: application/json' -d'
        {
          "type": "s3",
          "settings": {
            "bucket": "'$BUCKET'",
            "endpoint": "'$ENDPOINT'",
            "region": "'$REGION'",
            "base_path": "'$REPO_PATH'"
          }
        }
        '

        bp_echo "delete the snap $SNAPSHOT_NAME"
        curl -k -u "elastic:$PASSWORD" -X DELETE
"${ES_URL}/_snapshot/k10_repo/$SNAPSHOT_NAME?pretty"
        bp_echo "Deletion of $SNAPSHOT_NAME was successful"
        # make sure all output are grabbed by kanister pod
        sleep 4
```