Kanister.io

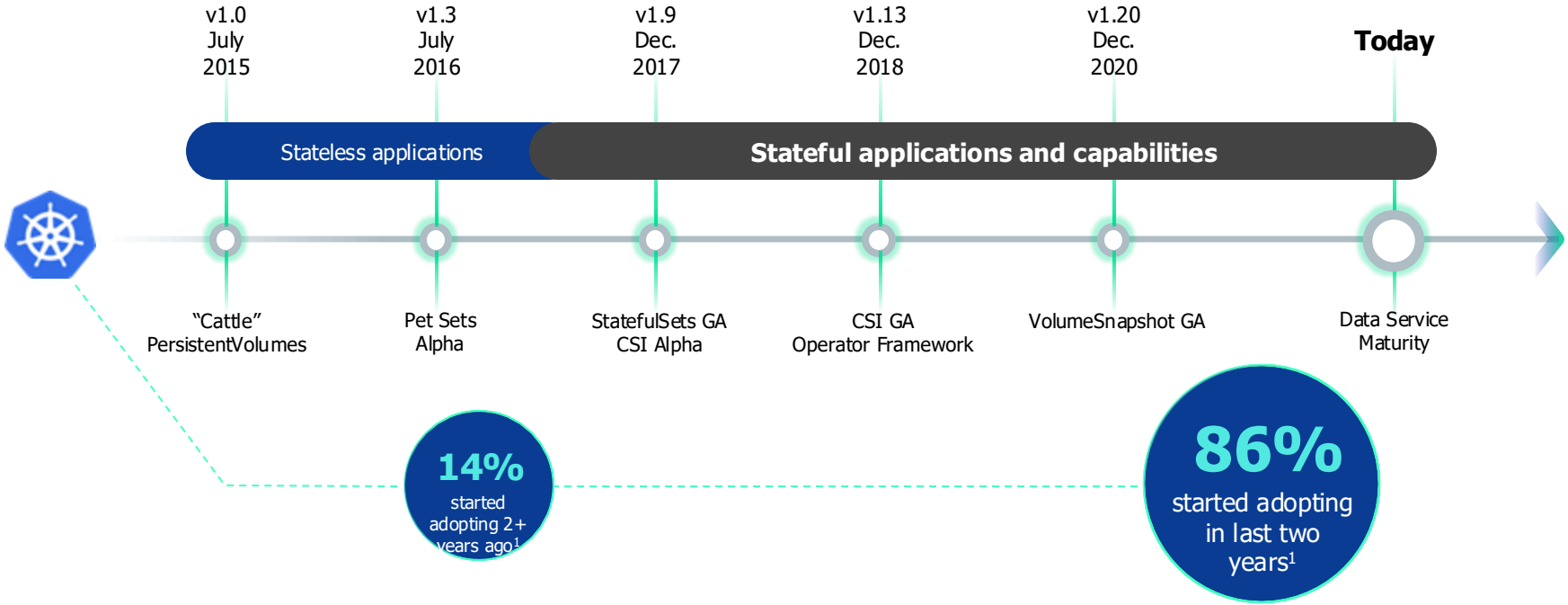
Application Consistent Backups on Kubernetes

Daniil Fedotov
Senior Open Source Engineer
@Veeam Kasten
daniil.fedotov@veeam.com

Motivation: Data on Kubernetes

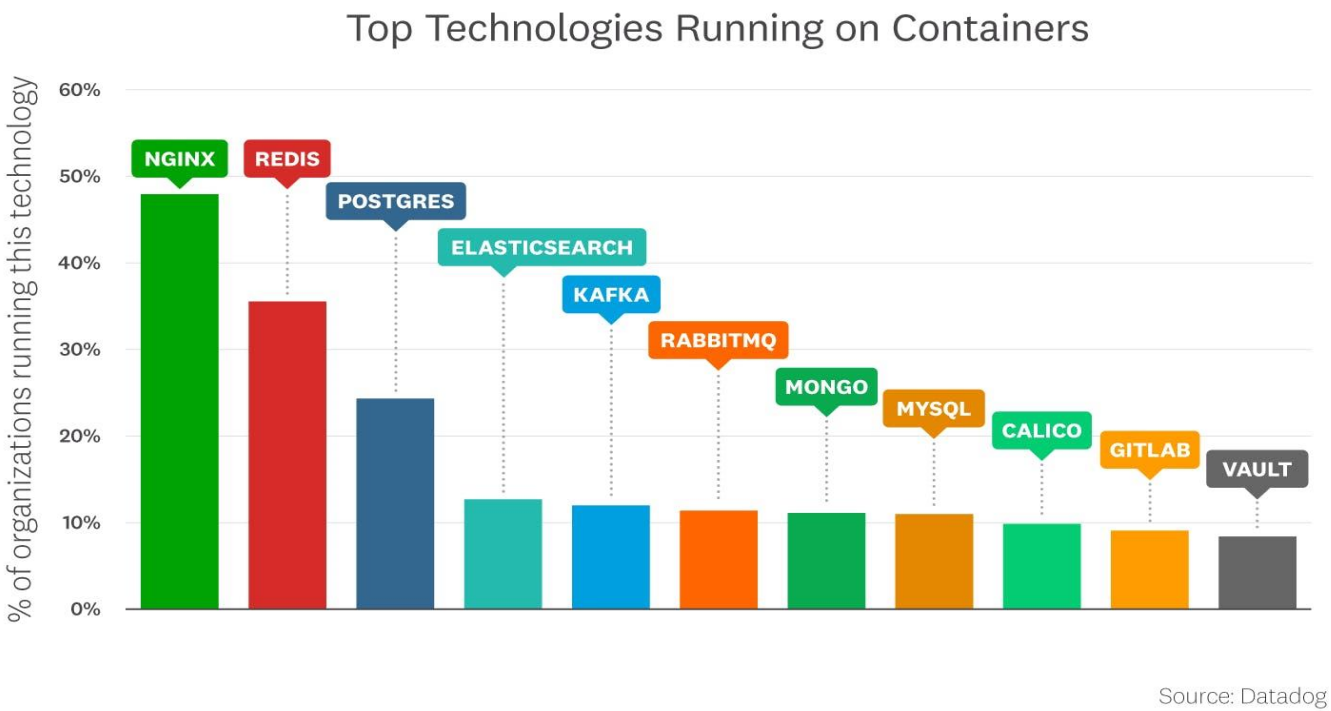
- Myth: everything is stateless on Kubernetes
  - Kubernetes cluster and application configuration has state
  - Secrets, RBAC, DNS, etc
  - CNCF Study 2020: 55% of respondents running stateful workloads
  - Audit requirements unsatisfied: proof of point in time cluster state configuration
- Myth: with GitOps I can recover a cluster with all applications
  - Stateful workloads still need to backup artifacts outside the cluster
  - Audit, forensics, point-in-time recovery concerns are still there
- Myth: with public cloud I am protected
  - Cloud providers recommend you set up disaster recovery
- Myth: etcd backups protect Kubernetes data
  - Etcd restore almost never gets applications into desired state

From Stateless to Stateful



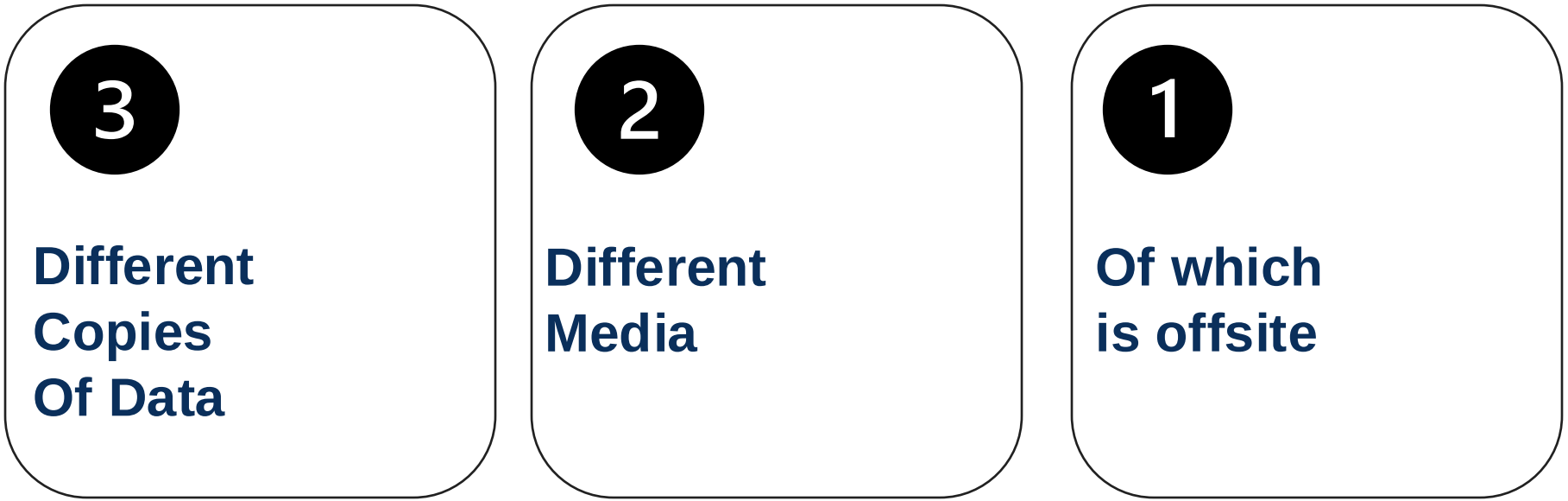
[1] 2022 Data on Kubernetes Report (N=500+)

Applications on Kubernetes



Datadog Real-World Container Use Report – Nov 2022

3-2-1 backup rule



Source: The DAM Book: Digital Asset Management for Photographers, Peter Krogh, 2005.

- https://community.veeam.com/blogs-and-podcasts-57/3-2-1-1-0-golden-backup-rule-569
- https://www.cisa.gov/uscert/sites/default/files/publications/data_backup_options.pdf

The Challenge: Complex Workflows

- One application includes many domains
  - *Difficult to separate concerns of different domain experts*

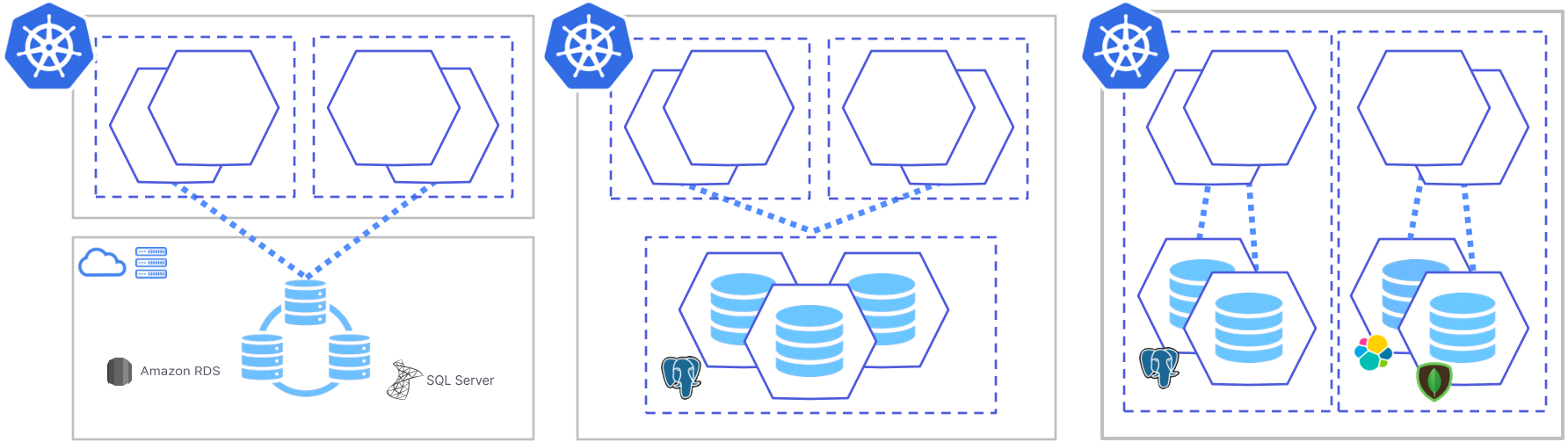
- Many moving parts
  - Different types of backups
    - *Logical backups*
    - *Volume snapshots*
    - *Provider specific API calls – Amazon RDS, data service operators*

  - Application Lifecycle
    - *Scale up/down workloads*
    - *Quiesce/Unquiesce*

  - Different types of targets
    - *Object storage*
    - *Vendor targets*

Cloud Native Deployment Patterns

Application using data services outside of Kubernetes	Data services in Kubernetes – separate from Application	Application includes data services – all in Kubernetes



Kanister.io

The Challenge: Flavours of Data Management

- Storage-centric snapshots
  - Provided by the underlying file or block storage
  - Crash-consistent
- Storage-centric with data service hooks
  - Freeze/unfreeze data service layer during snapshot process
- Data service-centric
  - Use database specific utilities
  - mysqldump, pg_dump, mongodump etc.
- Application-centric
  - Exercise all the above capabilities in a coordinated manner

Kanister.io: CNCF sandbox history

- 2017: Created & launched @ KubeCon NA
- 2023-06-20: Submitted to CNCF
- 2023-09-19: Vote passes for Acceptance https://github.com/cncf/sandbox/issues/46 Project Onboarding: https://github.com/cncf/toc/issues/1172
- 2023-11-07: Veeam Press Release, adopter and ISV support

Kanister Contribution to CNCF (Sandbox)

kanister.io

CLOUD NATIVE COMPUTING FOUNDATION

Kanister community blueprints:

Amazon RDS · Cassandra · PostgreSQL · MySQL · MongoDB · SQL Server · K8ssandra · Elasticsearch

Growing database diversity, e.g., Vector databases

Weaviate · Milvus · Pinecone

Data on Kubernetes Growth

Databases (SQL/NoSQL) lead the growth of stateful workloads on Kubernetes, database diversity and deployments continue to increase data protection requirements

Kanister.io

Data protection operations in a cloud native, extensible manner for application consistent backup and recovery. Kanister has been an open source project and community since 2017.

Rapid Release and Innovation

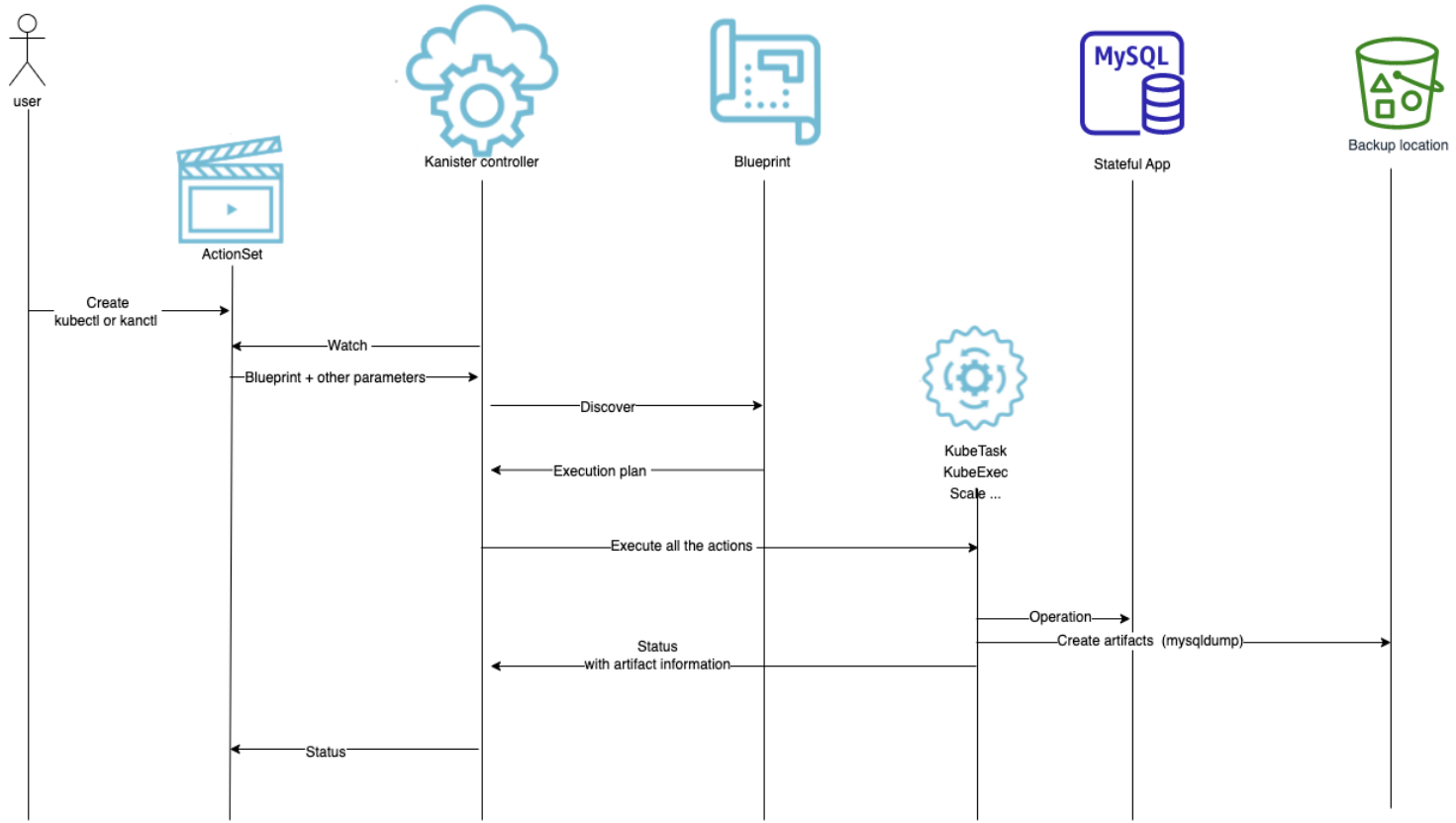
- 0.94 Prometheus metrics, OCP 4.13 support, Grype security scanner
- 0.92 MongoDB Atlas blueprint
- 0.91 OCP 4.12 support, AWS RDS Postgres blueprint
- 0.90 Kopia controller, Incremental Elastic blueprint

How Kanister.io Orchestrates Data Protection

A Kanister controller, installed by Helm chart, provides new Custom Resource Definitions:

- Blueprint
  - Defines workflows for backup, restore and delete operations
  - Part of your infrastructure setup
- ActionSet
  - Runs an action to backup, restore and delete
  - Created on backup and restore action, contains status of operation
- Profile
  - Defines target destination for backups or sources for restores, e.g. S3 bucket

Kanister: in Action



Breaking down the Blueprint

Blueprints are custom resources

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
  name: rds-postgres-snapshot-bp
```

```
$ kubectl get crds | grep kanister
actionsets.cr.kanister.io                2024-05-07T15:30:32Z
blueprints.cr.kanister.io                2024-05-07T15:30:33Z
profiles.cr.kanister.io                  2024-05-07T15:30:33Z
repositoryservers.cr.kanister.io         2024-05-07T15:30:33Z
```

https://github.com/kanisterio/kanister/blob/master/examples/aws-rds/postgresql/rds-postgres-snap-blueprint.yaml

Blueprints are templates for actions

- ‘backup’, ‘restore’, ‘delete’ are actions
- Actions have phases
- Each phase executes a Function

ActionSets execute actions

- Target blueprint and action name
- Each action provide template values
- All actions in actionset execute in parallel
- Phases run sequentially

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  name: rds-backup
  namespace: kasten-io
spec:
  actions:
  - name: backup
    blueprint: rds-postgres-snapshot-bp
    object:
      apiVersion: v1
      name: dbconfig
      namespace: pgtestrds
      resource: configmaps
    profile:
      name: s3-profile-sph7s
      namespace: pgtestrds
```

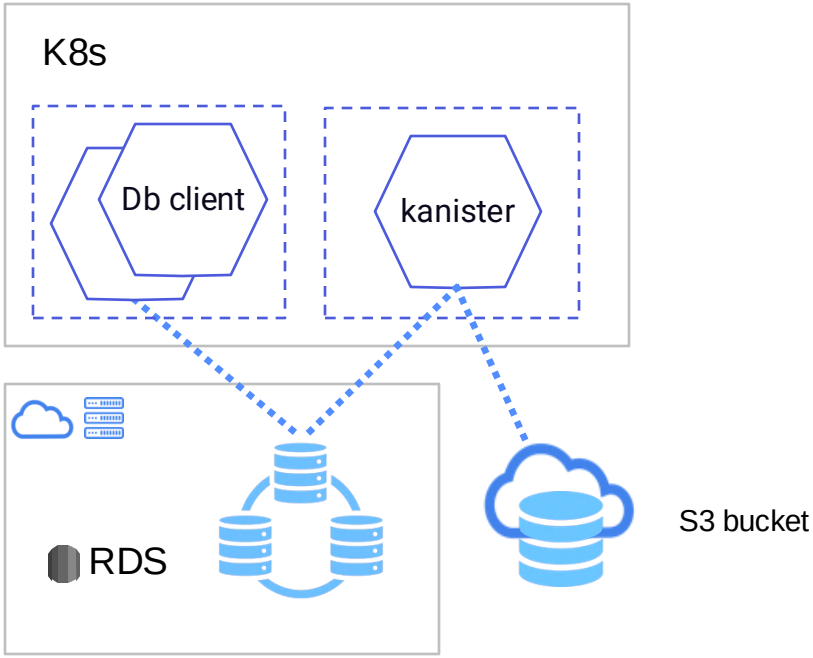
ActionSets track status

- Each phase has progress and state
- ActionSet produces artifacts

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
...
status:
  actions:
  - name: backup
    blueprint: rds-postgres-snapshot-bp
    artifacts:
      backupInfo:
        keyValue:
          backupID: backup-xd6c7jp6xl.tar.gz
          dbSubnetGroup: default
          instanceID: rds-demo-postgresql-instance
          securityGroupID: |
            - sg-xyz
          snapshotID: rds-demo-postgresql-instance-r6wffg56nf
    phases:
    - name: createSnapshot
      output:
        allocatedStorage: 20GiB
        dbSubnetGroup: default
        instanceID: rds-demo-postgresql-instance
        securityGroupID: |
          - sg-xyz
        snapshotID: rds-demo-postgresql-instance-r6wffg56nf
      progress:
        lastTransitionTime: "2024-05-07T16:08:42Z"
        progressPercent: "100"
      state: complete
```

Demo: RDS backup

Demo infra setup



Advantage of kanister approach

- We shipped backup off from RDS into object storage
- We ran all operations on K8s runners
- We can have blueprints and for all services in the same catalog in K8s
- We have backups metadata (actionsets) in K8s

Kanister integrations

Kanister functions

- Custom Logic
  - KubeExec
  - KubeTask
- Resource Lifecycle
  - Scale up/down workloads
  - KubeTask with kubectl command
- Handle PVC
  - Backup/Restore/DeleteData
  - PrepareData
- Volume Snapshots
  - Create/Restore/Delete
- Amazon RDS
  - Create/Restore/Delete
  - ExportSnapshotToRegion

Providers supported

- Object Storage
  - Amazon S3
  - S3 Compliant
  - Azure Blob
  - Google Cloud Storage
- Block/File Storage (in-tree)
  - Amazon EBS/EFS
  - Azure Disk
  - Google Persistent Disk
  - IBM Disk
  - CSI

Thank you!

Kanister resources:

- github.com/kanisterio/kanister
- @kanisterio
- #kanisterio
- tiny.cc/kanisterio

```
$ git clone git@github.com:kanisterio/kanister.git
# install Kanister operator controller
$ kubectl apply -f bundle.yaml
# install your application
$ kubectl apply -f examples/mongo-sidecar/mongo-cluster.yaml
# use an existing blueprint, tweak one, or create one yourself
$ kubectl apply -f examples/mongo-sidecar/mongo-blueprint.yaml
# perform operations (requires setting secrets and configmap)
$ kubectl create -f examples/mongo-sidecar/backup-actionset.yaml
```