the **GORILLA GUIDE**® to...

Kubernetes

Blueprints

Foundation Edition

DAN SULLIVAN

INSIDE THE GUIDE:

- Learn what blueprints are in Kubernetes
- Explore sample blueprint implementations
- How to extend Kasten K10 functionality with blueprints

KASTEN by Veeam

POWERED BY ActualTech MEDIA

Kubernetes Blueprints

By Dan Sullivan

TABLE OF CONTENTS

Please register at KubeCampus.io to get hands-on experience with Kubernetes.

Publisher's Acknowledgements

EDITORIAL DIRECTOR

Keith Ward

DIRECTOR OF CONTENT DELIVERY

Wendy Hernandez

CREATIVE DIRECTOR

Olivia Thomson

SENIOR DIRECTOR OF CONTENT

Katie Mohr

WITH SPECIAL CONTRIBUTIONS FROM KASTEN BY VEEAM®

Michael Courcy, Solutions Architect

Thomas Keenan, Sr. Product Marketing Manager

Mark Lavi, Principal, Cloud Product Manager

Tom Leyden, VP Product Marketing

ABOUT THE AUTHOR

Dan Sullivan is a principal engineer and architect specializing in cloud architecture, data engineering, and analytics. He has designed and implemented solutions for a wide range of industries and is the author.

Introduction

Kubernetes is increasingly the platform of choice for efficiently deploying applications and services and that's driving the adoption of practices such as using Infrastructure as Code (IaC) to manage Kubernetes environments. In this Gorilla Guide, you'll learn about Kubernetes blueprints, which are declarative specifications to maintain resources for Kubernetes clusters. As the name implies, blueprints are templates that can be customized as needed for specific requirements. One of the key advantages of blueprints is the existing set of examples available from open source repositories that allow you to quickly take advantage of IaC.

Practitioners welcome any tool that eases the operational burdens of running Kubernetes clusters.

This Gorilla Guide begins with an introduction to Kanister blueprints in Kubernetes and the functions they serve. Next, we investigate the challenges they address and consider example blueprints for PostgreSQL and SQL Server backups. With a solid understanding of blueprints, we shift our attention to Kanister, an open source framework for creating data management blueprints in a Kubernetes cluster. By the end of this guide, you'll be ready for hands-on practice with Kanister blueprints in Kubernetes with a KubeCampus lab.

What Are Blueprints in Kubernetes?

Kubernetes is known to be a challenge to learn, manage, and operate. Practitioners welcome any tool that eases the operational burdens of running Kubernetes clusters. For example, Helm is a widely used application for finding and deploying software in Kubernetes. Another complementary tool for application-level data management in Kubernetes is Kanister, an open source solution that uses blueprints to help standardize and consistently manage data management operations in Kubernetes.

Blueprints are templates for deploying Kubernetes resources and performing data management actions. For example, a Kubernetes cluster may have a MongoDB database deployed using a StatefulSet. While the StatefulSet specification describes characteristics of the deployment, such as the number of replicas to have available, it doesn't capture details of operations we may want to perform on that database. For example, pausing an application for a consistent backup may require flushing filesystem buffers and maintaining write locks. So, while Kubernetes database operators may have a database backup operation, the orchestration across upstream- and downstream-dependent applications and systems requires coordination that Kanister can invoke in addition to the operator function. This is where blueprints come in. A blueprint specifies actions that are applied to a resource, such as a MongoDB database. Actions include operations such as creating a backup or restoring data from a backup.

Of course, cluster and database administrators can manually run backups or create shell scripts scheduled as cron jobs to perform backups. Manually executing data protection operations like backups can be error-prone. Although scripts can help reduce the chance of errors, it's challenging to write scripts that correctly specify what steps to execute in the wide array of conditions that may arise. This is one of the reasons declarative specifications are so important. They define what should happen and the underlying system, such as Kubernetes, uses controllers to determine how to implement the desired state (see **Figure 1**).

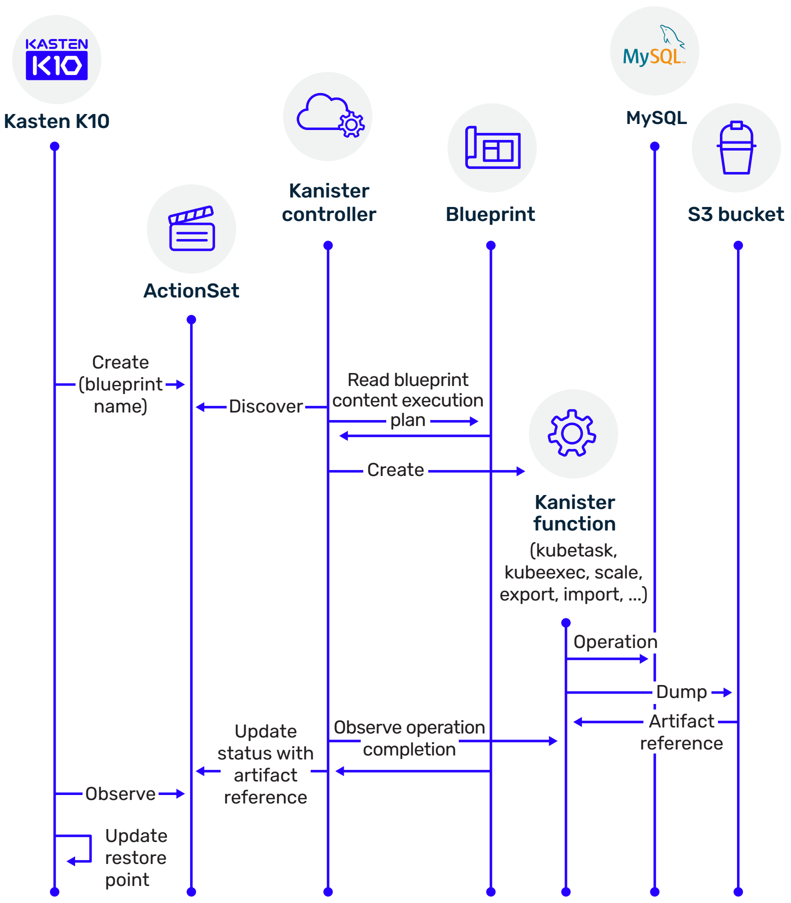
```
kubectl --namespace mysql apply -f \
https://raw.githubusercontent.com/kanisterio/
kanister/0.91.0/examples/mysql/blueprint-v2/
mysql-blueprint.yaml

kubectl --namespace mysql annotate statefulset/
mysql-release \
  kanister.kasten.io/blueprint=mysql-blueprint
```

Figure 1: By installing a Kanister Blueprint and adding an annotation to a MySQL deployment, data management operations are associated to this MySQL instance (Source: https://docs.kasten.io/latest/kanister/mysql/install.html)

Blueprints allow cluster administrators to centrally manage Kubernetes resources as well as automatically deploy infrastructure and carry out management operations, like backing up and restoring databases.

Blueprints provide functionality that allows system administrators to define cluster and application specifications in code. This extends the benefits of IaC to the application level (see **Figure 2**). Many of the benefits administrators have come to expect from using code to manage infrastructure now extend to managing applications and data operations on those applications. With the extension comes a reduced risk of human error as well as improving the scalability of

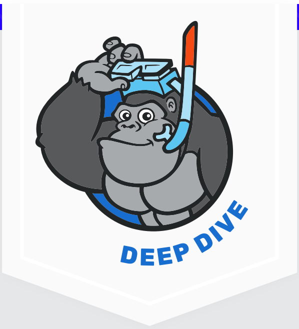


Kasten K10
ActionSet
Kanister controller
Blueprint
MySQL
S3 bucket
Create (blueprint name)
Discover
Read blueprint content execution plan
Create
Kanister function (kubetask, kubeexec, scale, export, import, ...)
Operation
Dump
Artifact reference
Observe operation completion
Update status with artifact reference
Observe
Update restore point

Figure 2: Steps in the blueprint lifecycle

DevOps staff who can now more effectively and efficiently manage large, complicated Kubernetes environments.

Blueprints help system administrators overcome several challenges. First, they provide consistent data protection for backup and recovery across environments, such as staging and production. This is especially important when agile software development methods are employed. In those environments, small and frequent changes to code are the norm.



Fostering the Path to GitOps

To maintain quality control over software, teams often use multiple environments to develop and then test software before putting it into production. It's important for testing and staging environments to be consistent with production environments. This is challenging because a Kubernetes environment can have many services running, each using a variety of libraries, applications, databases, and operating systems. Something as simple as running a different minor version of a database in staging and production environments can result in code operating correctly in staging and then failing to operate in production. Consistency across environments is essential for ensuring reliable system operations. In effect, this is the way to transition from manual bespoke operations for one environment to more automated cloud-native operations across all scenarios and foster the path to GitOps.

Now that we've reviewed what blueprints are and the value they bring to development and operations, it's time to investigate some implementation details.

Sample Blueprint Implementations

Blueprints are specified with YAML, which should be accessible to anyone who is used to working with Kubernetes and deploying resources. Like other Kubernetes resources, blueprint definitions begin with a kind and metadata specification such as shown in **Figure 3**.

```
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
  name: mysql-blueprint
```

Figure 3: Kind and metadata specification for a blueprint

Next, there's an actions section where operations such as backup, restore, and delete can be found. Each action can specify input and output artifacts, secrets, and phases in which functions are applied. For example, a backup operation may include a dump to object store function. The specification for the function includes an image to use and a command to run. An example of a command to back up a SQL Server database is shown in **Figure 4**.

```
  actions:
  backup:
    outputArtifacts:
      mysqlBackup:
        # 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.dumpToStore.
Output.kopiaOutput }}"
    phases:
    - func: KubeTask
      name: dumpToStore
      objects:
        mysqlSecret:
          kind: Secret
          name: '{{ index .Object.metadata.labels
"app.kubernetes.io/instance" }}'
          namespace: '{{ .StatefulSet.
Namespace }}'
      args:
        image: ghcr.io/kanisterio/
mysql-sidecar:0.91.0
        namespace: "{{ .StatefulSet.Namespace }}"
        command:
        - bash
        - -o
        - errexit
        - -o
        - pipefail
        - -c
        - |
          backup_file_path="dump.sql"
          root_password="{{ index .Phases.dumpTo-
Store.Secrets.mysqlSecret.Data "mysql-root-pass-
word" | toString }}"
          dump_cmd="mysqldump --column-statis-
tics=0 -u root --password=${root_password} -h {{
index .Object.metadata.labels "app.kubernetes.io/
instance" }} --single-transaction --all-databases"
          ${dump_cmd} | kando location push
--profile '{{ toJson .Profile }}' --path "${back-
up_file_path}" --output-name "kopiaOutput" -
```

Figure 4: A MySQL database backup function (Source: https://github.com/kanisterio/kanister/blob/master/examples/mysql/blueprint-v2/mysql-blueprint.yaml)

```
  restore:
    inputArtifactNames:
    # The kopia snapshot info created in backup 
phase can be used here
    # Use the `--kopia-snapshot` flag in kando to 
pass in `mysqlBackup.KopiaSnapshot`
    - mysqlBackup
    phases:
    - func: KubeTask
      name: restoreFromStore
      objects:
        mysqlSecret:
          kind: Secret
          name: '{{ index .Object.metadata.labels 
"app.kubernetes.io/instance" }}'
          namespace: '{{ .StatefulSet.
Namespace }}'
      args:
        image: ghcr.io/kanisterio/
mysql-sidecar:0.91.0
        namespace: "{{ .StatefulSet.Namespace }}"
        command:
        - bash
        - -o
        - errexit
        - -o
        - pipefail
        - -c
        - |
          backup_file_path=”dump.sql"
          kopia_snap='{{ .ArtifactsIn.mysqlBackup.
KopiaSnapshot }}'
          root_password=”{{ index .Phases.restore-
FromStore.Secrets.mysqlSecret.Data "mysql-root-
password” | toString }}"
          restore_cmd="mysql -u root --pass-
word=${root_password} -h {{ index .Object.metada-
ta.labels "app.kubernetes.io/instance" }}"
          kando location pull --profile '{{ toJson 
.Profile }}' --path "${backup_file_path}" --kop-
ia-snapshot "${kopia_snap}" – | ${restore_cmd}
```

Figure 5: A MySQL database restore function (Source: https://github.com/kanisterio/kanister/blob/master/examples/mysql/blueprint-v2/mysql-blueprint.yaml)

This function instructs the Kanister controller to create a sidecar container which executes a Bash command to back up a MySQL database and write the backup file to an external repository. Notice the template is parameterized so it can be applied to different databases, secrets (such as the root password), and repositories.

The structure of the restore operation in a MySQL blueprint is the inverse of the backup, as shown in **Figure 5**. Notice the action similarities: both include parameterized commands, access to secrets, and database specific tools for operations. The Kanister blueprint data management model can be customized and extended to any CLI and API to accommodate operations on any data service.

Having access to blueprints such as these can help you quickly get started. It also helps to use tools specifically designed to streamline operations with blueprints.

Extending Kasten K10 Functionality with Blueprints: What's Involved and How Do I Start?

To get started using blueprints with Kasten K10, we'll work with the open source framework Kanister. Kanister was designed with a few key goals in mind, including support for data management at the application level, APIs to support integration of Kanister with other tools and services, and extensibility to adapt to a wide range of use cases.

KANISTER ARCHITECTURE

Kanister includes several components that implement the blueprint management tool, including:

- Controller
- Blueprints
- ActionSets
- Profiles

The Controller is the component that is installed into a Kubernetes cluster and watches for new or updated executing blueprints. The Controller implements the operator pattern that's commonly used when implementing custom resources in Kubernetes. The operator pattern allows for extensions to Kubernetes functionality without requiring changes to the underlying Kubernetes code. In the case of Kanister, we're able to implement a controller for blueprints.

Blueprints, as described earlier, are specifications for sets of data management actions that will be applied within the Kubernetes cluster.

ActionSets are structures which cause the Controller to perform actions that are defined in the blueprint, such as backing up or restoring a database. ActionSets include supporting information and data needed to perform operations, including details on artifacts, as well as Kubernetes structures such as ConfigMaps and Secrets.

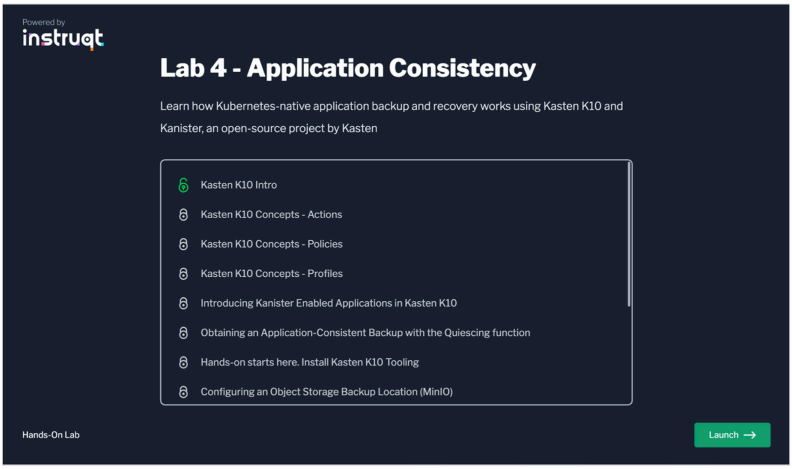
Profiles specify storage for data operation artifacts and related credentials.

GETTING STARTED WITH KASTEN K10 AND KANISTER

Kasten K10 can add and execute Kanister blueprints through its dashboard. This is a good place to start because the blueprints have been designed by experts and show helpful patterns to use when designing your own. For additional examples, see the Kanister Git repository for blueprints.

Hands-on Practice with Blueprints

Kasten by Veeam has created an application consistency lab to provide you with the opportunity to work with blueprints directly (see **Figure 6**).



Lab 4 - Application Consistency

Learn how Kubernetes-native application backup and recovery works using Kasten K10 and Kanister, an open-source project by Kasten

- Kasten K10 Intro
- Kasten K10 Concepts - Actions
- Kasten K10 Concepts - Policies
- Kasten K10 Concepts - Profiles
- Introducing Kanister Enabled Applications in Kasten K10
- Obtaining an Application-Consistent Backup with the Quiescing function
- Hands-on starts here. Install Kasten K10 Tooling
- Configuring an Object Storage Backup Location (MinIO)

Figure 6: Application Consistency Lab

The lab is configured with a K3s Kubernetes cluster with K10, minio for object storage, and MongoDB installed. During the lab, you'll be able to observe objects created during the backup process.

Blueprints are specified with YAML, which should be accessible to anyone who is used to working with Kubernetes and deploying resources.

As the process executes, you'll see VolumeSnapshots created along with PersistentVolumeClaims. Pods are created to export data and after the export completes, the PersistentVolumeClaims will be deleted, and the Pods terminated. MongoDB snapshots will be available in the object storage buckets at the end of the action.

In the lab you'll also explore running a logical backup, running a consistent backup, and running generic backups.

Now that you understand Kubernetes blueprints, it's time to take the next step, and go to KubeCampus.io to work through the lab and learn how to apply IaC-like techniques for managing data operations.

About Kasten by Veeam

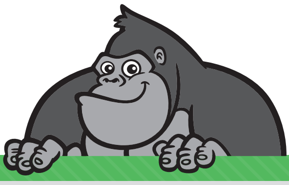
Kasten by Veeam® is the leader in Kubernetes backup. Kasten K10 is a Cloud Native data management platform for Day 2 operations. It provides enterprise DevOps teams with backup/restore, disaster recovery and application mobility for Kubernetes applications. Kasten K10 features operational simplicity and integrates with relational and NoSQL databases, all major Kubernetes distributions, and runs in any cloud to maximize freedom of choice. Our customers are confident that their Kubernetes applications and data are protected and always available with the most easy-to-use, reliable and powerful Cloud Native data management platform in the industry. For more information, visit www.kasten.io or follow [@kastenhq](https://twitter.com/kastenhq) on Twitter.

About ActualTech Media

ActualTech Media, a Future company, is a B2B tech marketing company that connects enterprise IT vendors with IT buyers through innovative lead generation programs and compelling custom content services.

ActualTech Media's team speaks to the enterprise IT audience because we've been the enterprise IT audience.

Our leadership team is stacked with former CIOs, IT managers, architects, subject matter experts and marketing professionals that help our clients spend less time explaining what their technology does and more time creating strategies that drive results.



If you're an IT marketer and you'd like your own custom Gorilla Guide® title for your company, please visit https://www.gorilla.guide/custom-solutions/