KASTEN by Veeam

Tutorial:
What went wrong with my persistent data?

Le Tran
Member of Technical Staff
Kasten by Veeam

Michael Cade
Field CTO
Kasten by Veeam

NFS

Local Out Of Tree

Persistent Volume Claims

Driver

Remote

ConfigMap

File

Flex

CSI

Persistent Volumes

Stateful

Secret

In-Tree

Projected

Ephemeral

Object

StatefulSet

Volume

Block

Plugin

KASTEN by Veeam

Kubernetes storage

KASTEN
by Veeam

Session Flow:

Introduction to Kubernetes storage concepts

- Ephemeral Volumes
- Projected Volumes
- Persistent Volumes (PV) and Persistent Volume Claims (PVC)
- StorageClasses
- Provisioners
- Volume Plugins
- StatefulSets and their relationship to storage

Troubleshooting Kubernetes storage issues.

Storage Considerations: Performance & Protection

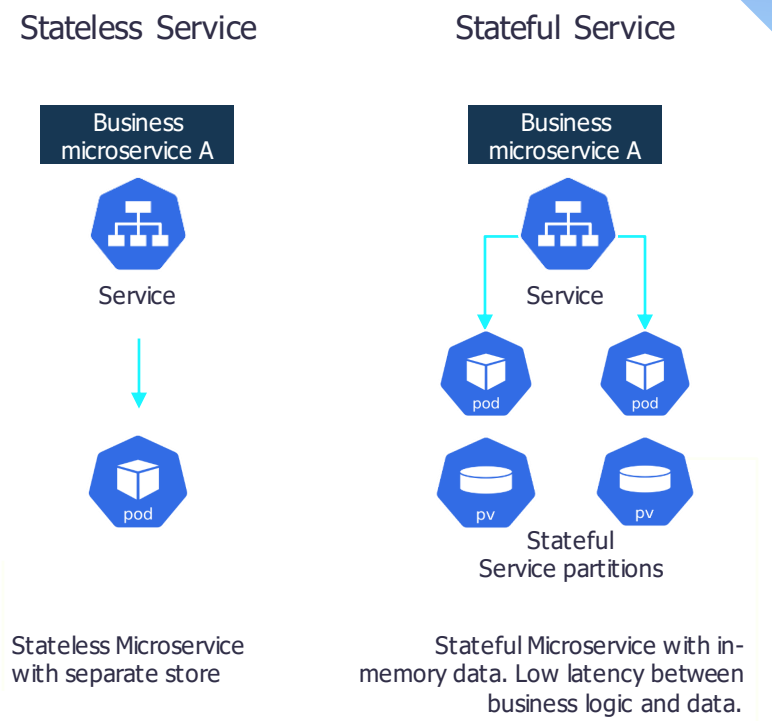
What is state?

Two types of processes

Some form of data that our application needs to function:

A container, pod, or application is just a process.

- Containers have their own virtual filesystem, but it is not persistent, when a container gets destroyed and recreated a new filesystem is recreated.
- Host processes have access to the underlying host filesystem.

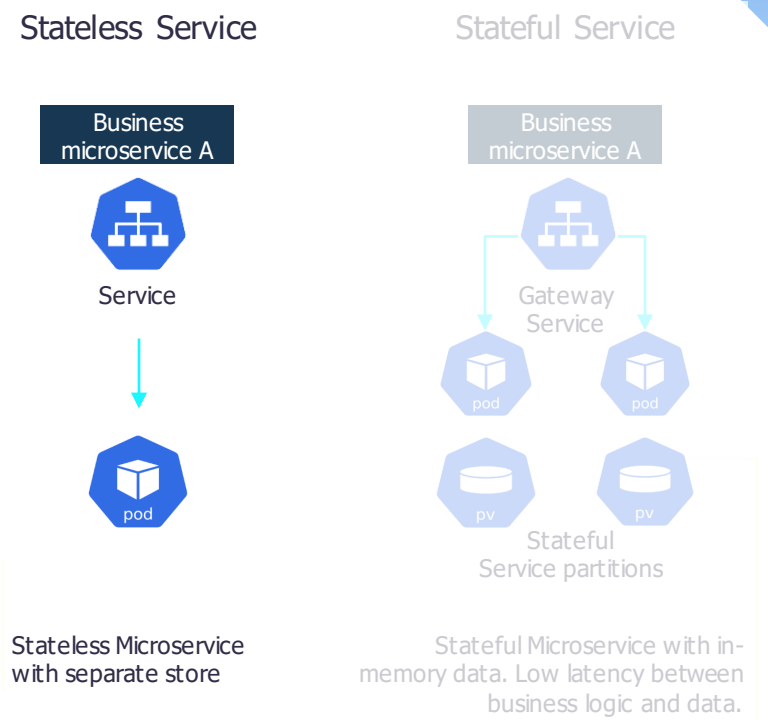


KASTEN by Veeam

What is state?

Stateless Processes

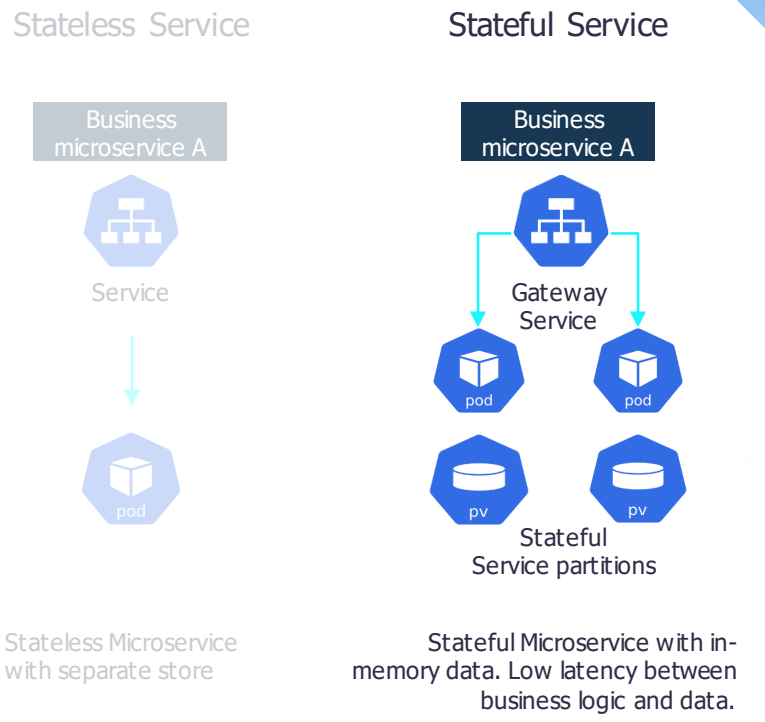
- Stateless processes do not rely on stored state or data to function.
- They do not store any state or data in memory or on the filesystem.
- Commonly used in microservices, they can be created and destroyed without impacting users.



What is state?

Stateful Processes

- Stateful processes rely on stored state or data to function.
- They store state and data in memory or on the filesystem.
- They require persistent storage and may need to synchronize state across multiple instances.
- Stateful processes can be harder to scale and maintain than stateless processes.
- However, they are necessary for applications requiring complex data processing or maintaining user session state.



KASTEN by Veeam

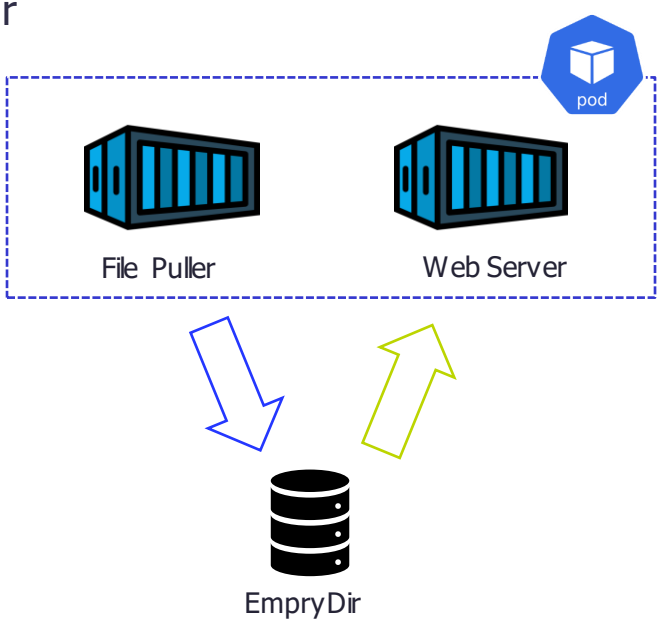
What if I was to create a database with no persistent storage?

..n2023_EU/Lab1

```
→ Lab1 (main) ✗
```

Ephemeral Volumes

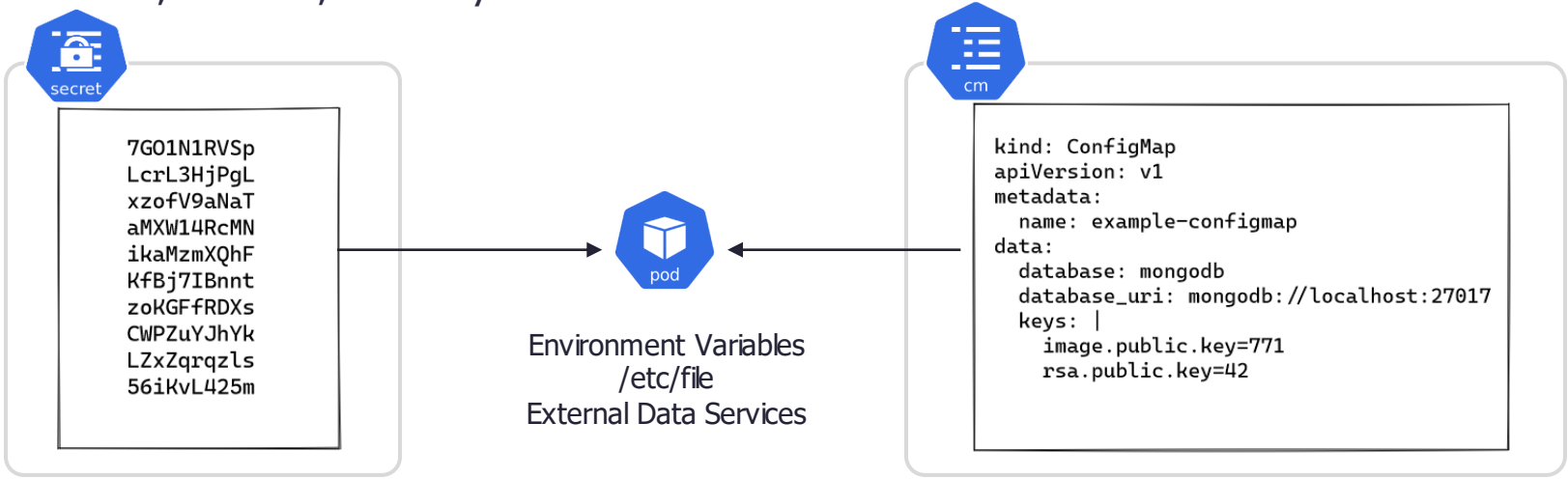
- Temp scratch file space from the host machine
- Data exists only for the lifecycle of the pod
- Can only be referenced “in-line” in pod definition, not via Persistent Volume | Claim
- Volume Plugin: EmptryDir



Projected Volumes

ConfigMaps - A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.

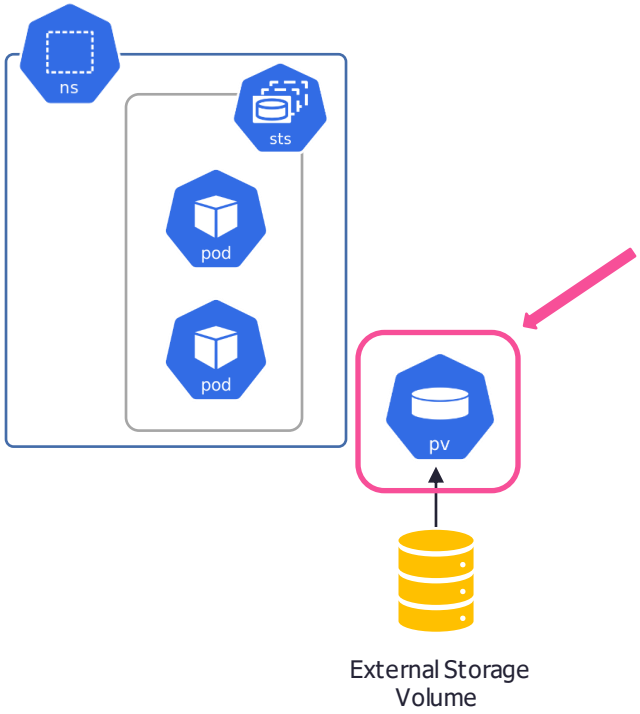
Secrets - A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.



KASTEN by Veeam

Persistent Volumes

- Kubernetes storage provides data persistence beyond the pod or container lifetime.
- Can be created dynamically or statically provisioned.
- Backed by local disk, NAS, or cloud storage.
- Allows data to persist across pod restarts and enables scaling without losing data.
- Critical component of many Kubernetes deployments, providing reliable and scalable storage for containerized applications.



KASTEN by Veeam

Persistent Volumes

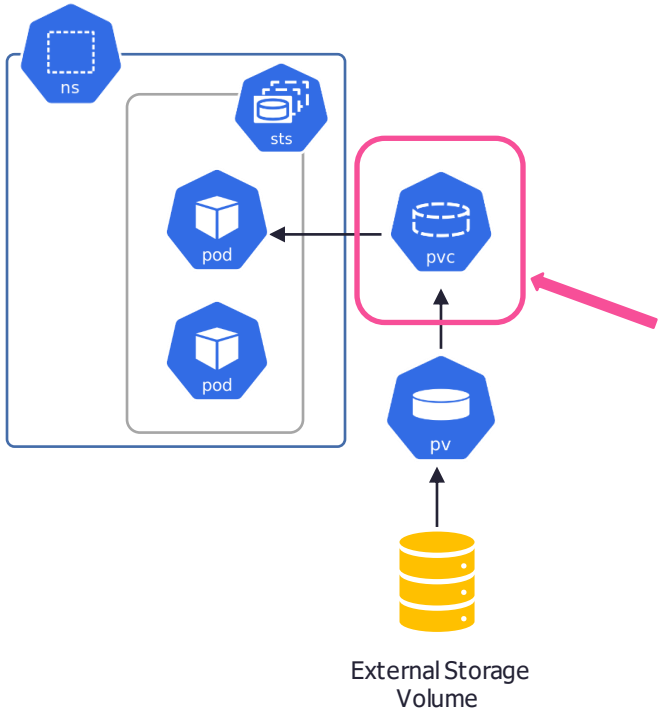
Types of Persistent Volumes

PersistentVolume types are implemented as plugins. Kubernetes currently supports the following plugins:

- `cephfs` - CephFS volume
- `csi` - Container Storage Interface (CSI)
- `fc` - Fibre Channel (FC) storage
- `hostPath` - HostPath volume (for single node testing only; WILL NOT WORK in a multi-node cluster; consider using `local` volume instead)
- `iscsi` - iSCSI (SCSI over IP) storage
- `local` - local storage devices mounted on nodes.
- `nfs` - Network File System (NFS) storage
- `rbd` - Rados Block Device (RBD) volume

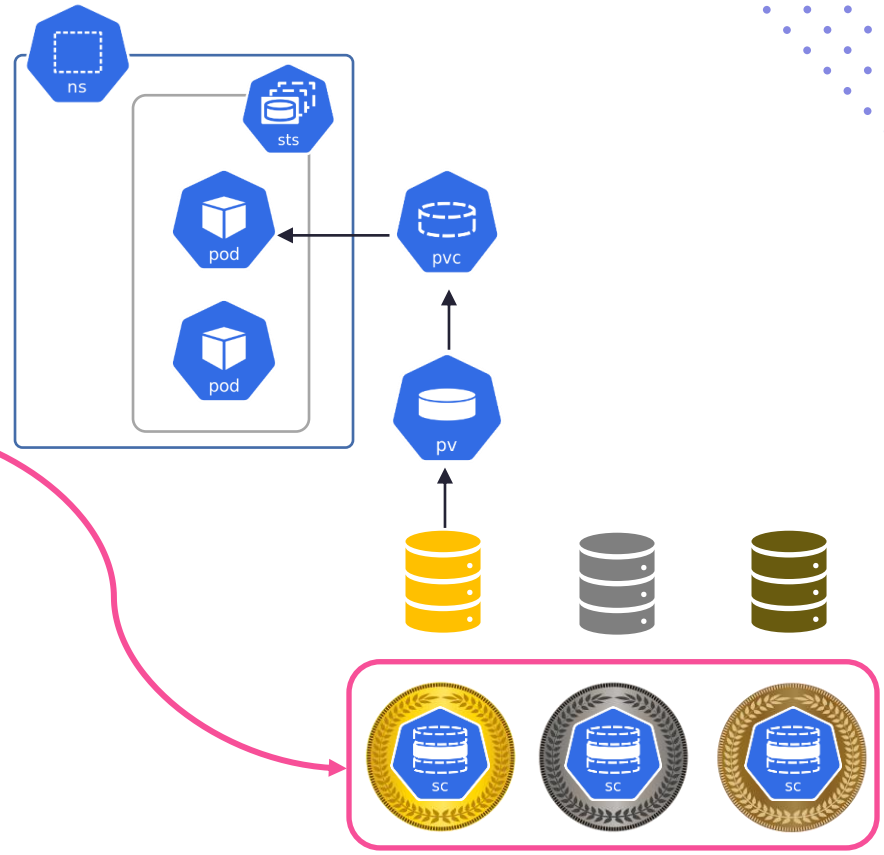
Persistent Volume Claims

- Kubernetes abstraction for requesting storage resources from available persistent volumes.
- Decouples storage requirements from the underlying implementation.
- Specifies requirements such as capacity, access mode, and storage class.
- Dynamically created and destroyed.
- Key component of Kubernetes storage management, providing a simple and flexible way to manage persistent storage resources for containerized applications.



Storage Classes

- Kubernetes object defining type and quality of storage volume.
- Used to dynamically provision persistent volumes.
- Can define performance levels, data protection policies, and access modes.
- Simplifies storage management and enables better resource utilization.
- Key feature of Kubernetes storage management, enabling dynamic and efficient provisioning of storage resources for containerized applications.



KASTEN by Veeam

Provisioners

Each StorageClass has a provisioner that determines what volume plugin is used for provisioning PVs. This field must be specified.

"internal" provisioners are shipped alongside Kubernetes (in-tree)

Volume Plugin	Internal Provisioner	Config Example
AWSElasticBlockStore	✓	AWS EBS
AzureFile	✓	Azure File
AzureDisk	✓	Azure Disk
CephFS	-	-
Cinder	✓	OpenStack Cinder
FC	-	-
FlexVolume	-	-
GCEPersistentDisk	✓	GCE PD
iSCSI	-	-
NFS	-	NFS
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
Local	-	Local

Volume Plugins

In-Tree

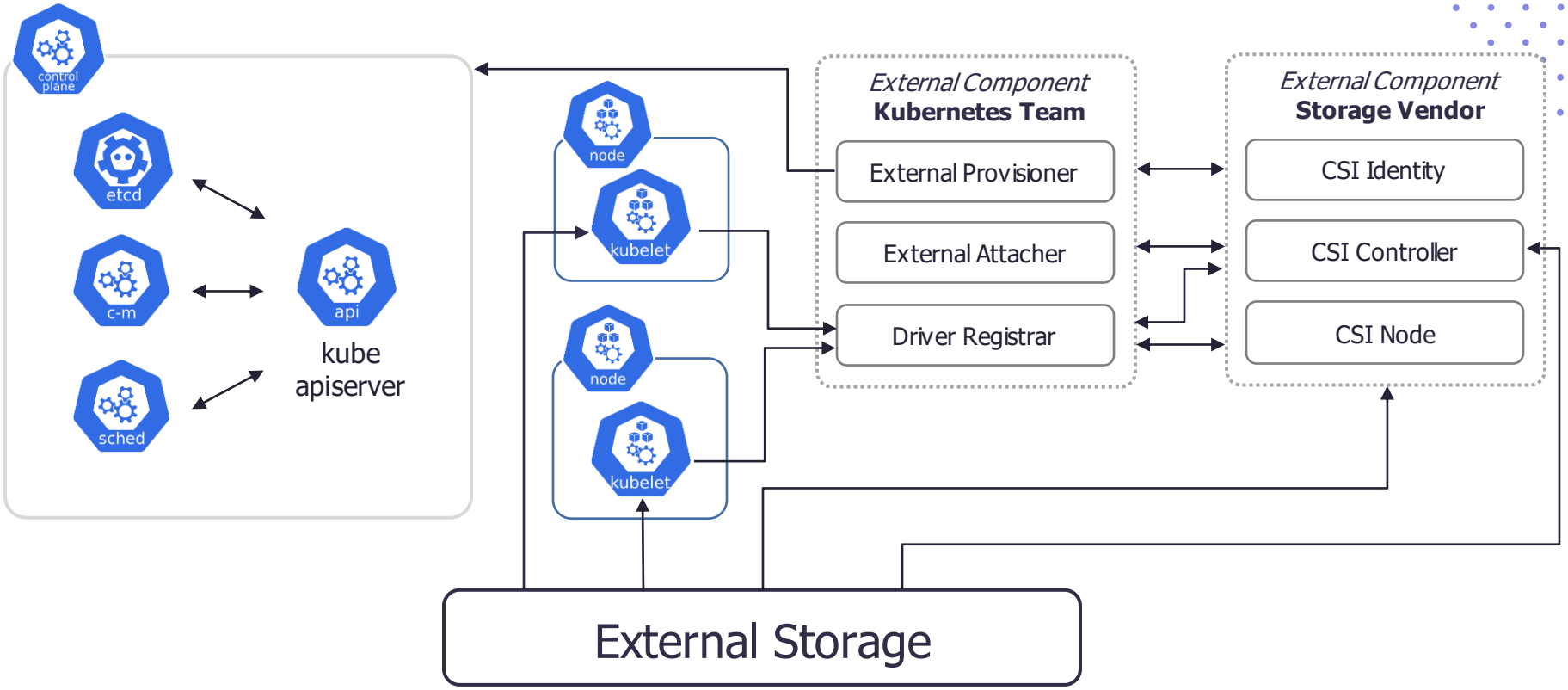
- volume plugins were “in-tree” meaning their code was part of the core Kubernetes code and shipped with the core Kubernetes binaries

Container Storage Interface (CSI)

- Containerised storage plugin deployed using standard Kubernetes primitives
- Built from driver and side cars.

KASTEN by Veeam

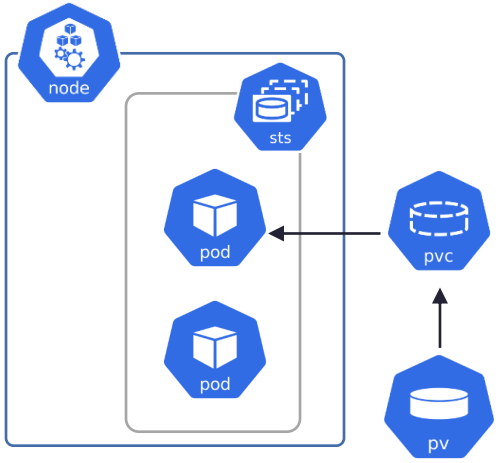
Container Storage Interface



KASTEN by Veeam

StatefulSets and their relationship to storage

- Stable network identifiers
- Ordered deployment and scaling
- Stable storage
- Self-healing
- Rolling updates



StatefulSets & Deployments

StatefulSet	Deployment
Used to deploy stateful applications	Used to deploy stateless applications
Pods created by StatefulSets have unique names which remain constant across application rescheduling	Pods created by Deployment have dynamic, random names and numbers that change across application rescheduling.
Pods are created in a sequential order and deleted in reverse	Pods are created and deleted randomly.
Pods are not interchangeable and maintain their identities after restarts.	Pods are interchangeable and do not maintain their identities after restarts.
It does not allow shared volume. Each pod replica has its own sticky volume and PVC.	It allows for shared via volume and PVC across all of the pod replicas

Lab 1 – Kubernetes Storage 

- Volumes
- StorageClasses

Start Lab 1

instruqt

- Recommended for **following along live**
- Provide your name & e-mail, then click **Start** to begin your hosted lab session

minikube

- Recommended for **self-paced learning at any time**
- Pre-reqs described in full on the GitHub README

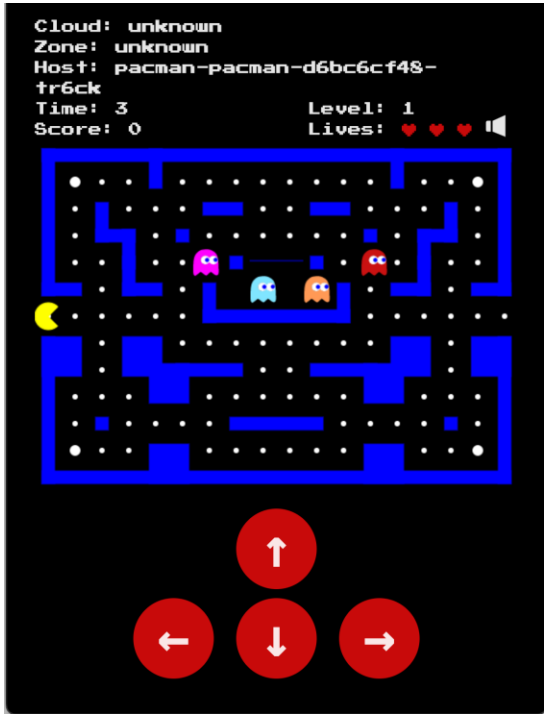
KASTEN by Veeam

Lab 2 - Storage Troubleshooting

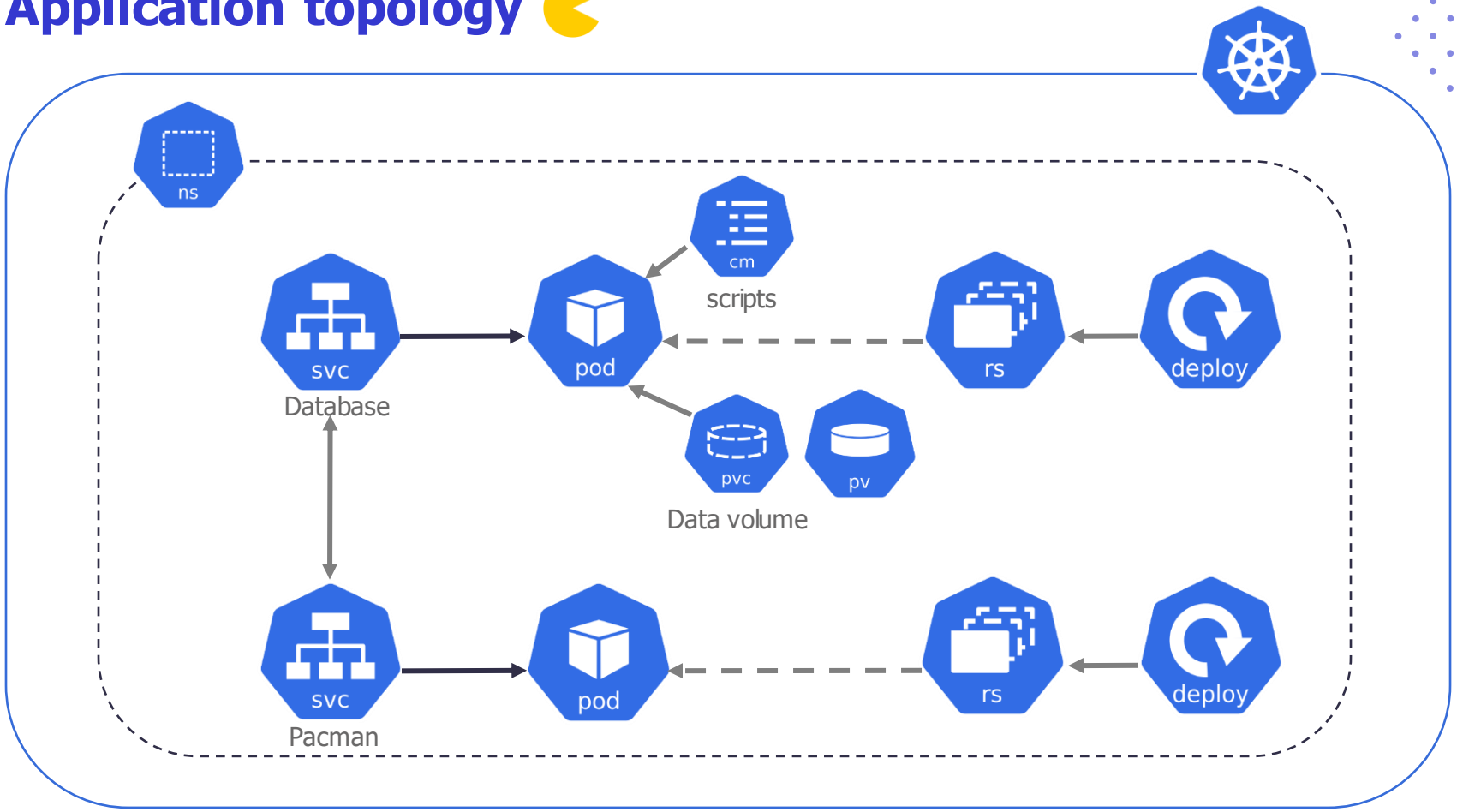
KASTEN by Veeam

Troubleshooting & Debugging

- Task: deploy a pacman application!



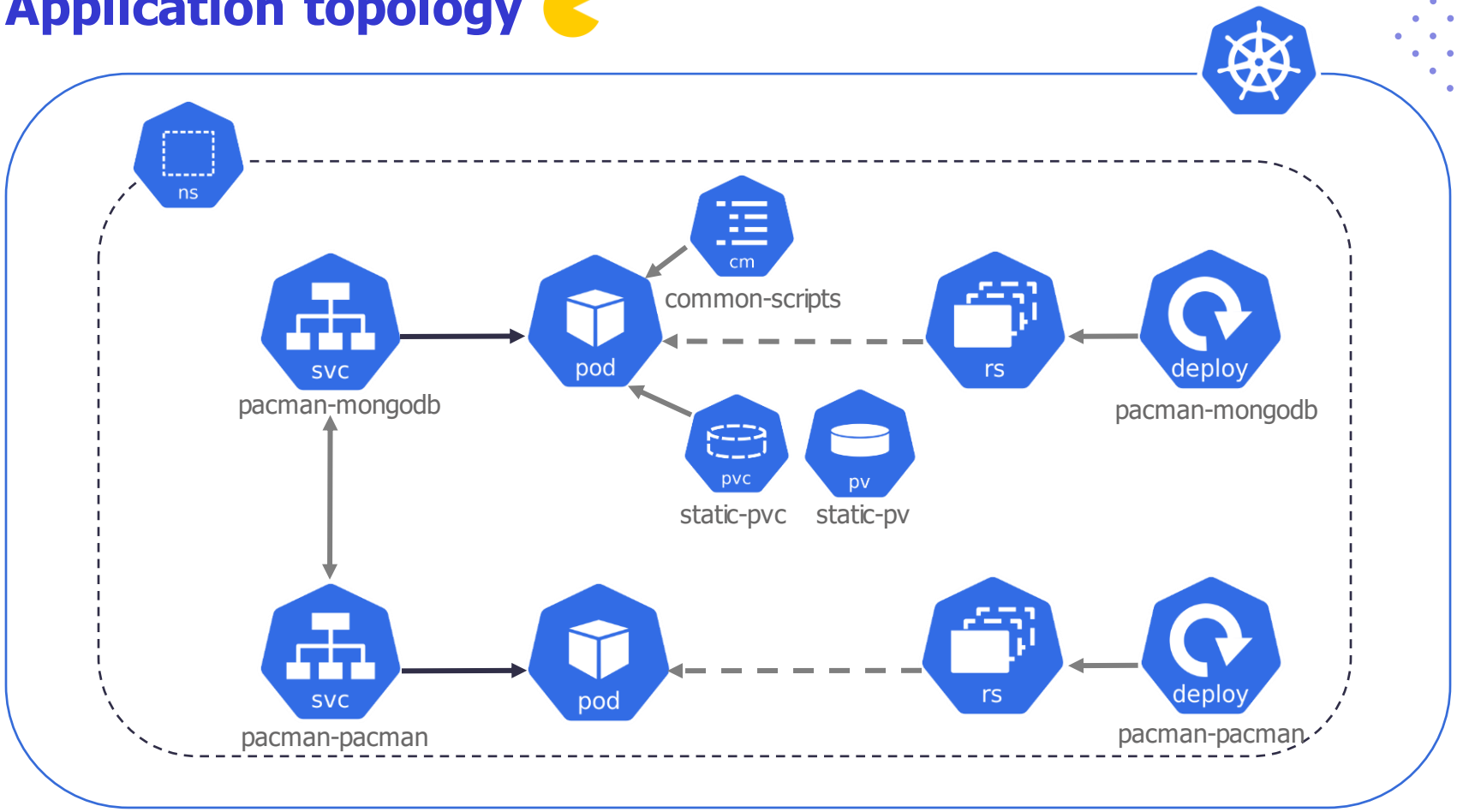
Application topology



Troubleshooting & Debugging

- Task: deploy a pacman application!
- Premise:
  - The resource manifests for the application are available in Lab2

Application topology



Troubleshooting & Debugging

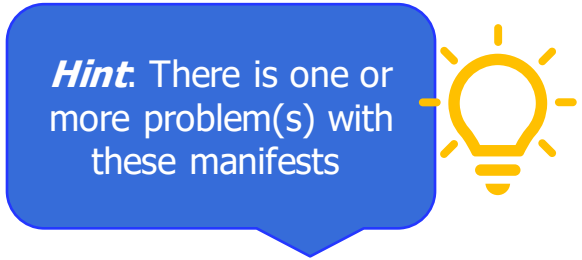
- Task: deploy a pacman application!
- Premise:
  - The resource manifests for the application are available in Lab2
  - Deploy them, then access the game with the following commands:

```
export POD_NAME=$(kubectl get pods --namespace $NS -l "app.kubernetes.io/name=pacman,app.kubernetes.io/instance=pacman" -o
jsonpath="{.items[0].metadata.name}")
export CONTAINER_PORT=$(kubectl get pod --namespace $NS $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
kubectl --namespace $NS port-forward $POD_NAME 8080:$CONTAINER_PORT
```

  - The game in your browser at localhost:8080 (http://127.0.0.1:8080)

Troubleshooting & Debugging

- Task: deploy a pacman application!



- Premise:
  - The resource manifests for the application are available in Lab2
  - Deploy them, then access the game with the following commands:

```
export POD_NAME=$(kubectl get pods --namespace $NS -l "app.kubernetes.io/name=pacman,app.kubernetes.io/instance=pacman" -o
jsonpath="{.items[0].metadata.name}")
export CONTAINER_PORT=$(kubectl get pod --namespace $NS $POD_NAME -o jsonpath="{.spec.containers[0].ports[0].containerPort}")
kubectl --namespace $NS port-forward $POD_NAME 8080:$CONTAINER_PORT
```

  - The game in your browser at localhost:8080 (http://127.0.0.1:8080)

KASTEN by Veeam

Useful kubectl commands

https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Purpose	Command
Set namespace for all subsequent commands	kubectl config set-context —-current-context --namespace=<ns>
Overview of pods, deployment, services, statefulsets, and replicasets	kubectl get all –n <ns>
List pods or get a specific pod	kubectl get pods -n <ns> kubectl get pod <my_pod> -n <ns> -owide | -oyaml | -ojson
View human-readable description of the object, including related objects and events	kubectl describe pod –n <ns>
See the logs for a running container	kubectl logs pod –n <ns> | -c <container_name> | -f
Create a new resource or modify an exisiting resource	kubectl apply –f <manifest_file(s)> | -n <ns>
Edit an existing resource from the default editor	kubectl edit <resource> <resource_name> -n <ns>
Update field(s) of a resource	kubectl patch <resource> <resource_name> -n <ns> -p '<field_to_be_updated>' | --type=<patch_type>
Delete resource	kubectl delete <resource> <resource_name> -n <ns> kubectl delete –f <manifest_file>
Interactive shell access to a running pod	kubectl exec <pod_name> -n <ns> -it – bash kubectl exec <pod_name> -n <ns> --stdin --tty -- /bin/sh

KASTEN by Veeam

Lab 2 – Storage Troubleshooting

KASTEN by Veeam

Start Lab 2

instruqt

- Recommended for **following along live**
- Provide your name & e-mail, then click **Start** to begin your hosted lab session

minikube

- Recommended for **self-paced learning at any time**
- Pre-reqs described in full on the GitHub README

KASTEN by Veeam

Lab 2 – recap

- What have we seen so far?

Static vs Dynamic Provision

Static Provision	Dynamic Provision
Volumes are pre-provisioned through Persistent Volumes manifests	Volumes are created on-demand via PVC and Storage Class
Supports CSI integration	Support CSI integration
Supports Volume Reclaim	Support Volume Reclaim
Fixed volume size	Flexible volume size
Complexity in DevOps integration	Easier, and provides automations for Infrastructure-as-Code

Sources:

1. https://bluexp.netapp.com/blog/cvo-blg-static-vs.-dynamic-storage-provisioning-a-look-under-the-hood
2. https://www.suse.com/c/rancher_blog/stupid-simple-kubernetes-persistent-volumes-explained-part-3/

Static vs Dynamic Provision

Static Provision	When to use ?
Volumes are pre-provisioned through Persistent Volumes manifests	• Predictable storage usage – when parameters such as sizing are known • Specific customization of storage – such as when using specific credentials • Usage of existing volumes, or re-use volumes • Usage of shared volumes
Supports CSI integration	
Supports Volume Reclaim	
Fixed volume size	
Complexity in DevOps integration	

Sources:

1. https://bluexp.netapp.com/blog/cvo-blg-static-vs.-dynamic-storage-provisioning-a-look-under-the-hood
2. https://www.suse.com/c/rancher_blog/stupid-simple-kubernetes-persistent-volumes-explained-part-3/

KASTEN by Veeam

Static vs Dynamic Provision

When to use ?	Dynamic Provision
• Flexible storage usage – when parameters such as sizing are unknown ahead of time • Infrastructure-as-Code • Optimization of cost – volumes are provisioned only on demand	Volumes are created on-demand via PVC and Storage Class
	Support CSI integration
	Support Volume Reclaim
	Flexible volume size
	Easier, and provides automations for Infrastructure-as-Code

Sources:

1. https://bluexp.netapp.com/blog/cvo-blg-static-vs.-dynamic-storage-provisioning-a-look-under-the-hood
2. https://www.suse.com/c/rancher_blog/stupid-simple-kubernetes-persistent-volumes-explained-part-3/

Storage Considerations

Performance & Protection

KASTEN by Veeam

Expand the PVC

Kubernetes 1.24: Volume Expansion Now A Stable Feature

Thursday, May 05, 2022

Not every volume type however is expandable by default.

- In-tree hostpath volumes are not expandable at all.
- CSI driver must have capability EXPAND_VOLUME in the controller or node service

Protect data and make it available under any circumstances

What level of protection is enough?

Data Protection workflows are complex

What do we have natively within Kubernetes to help us protect our data?

VolumeSnapshots

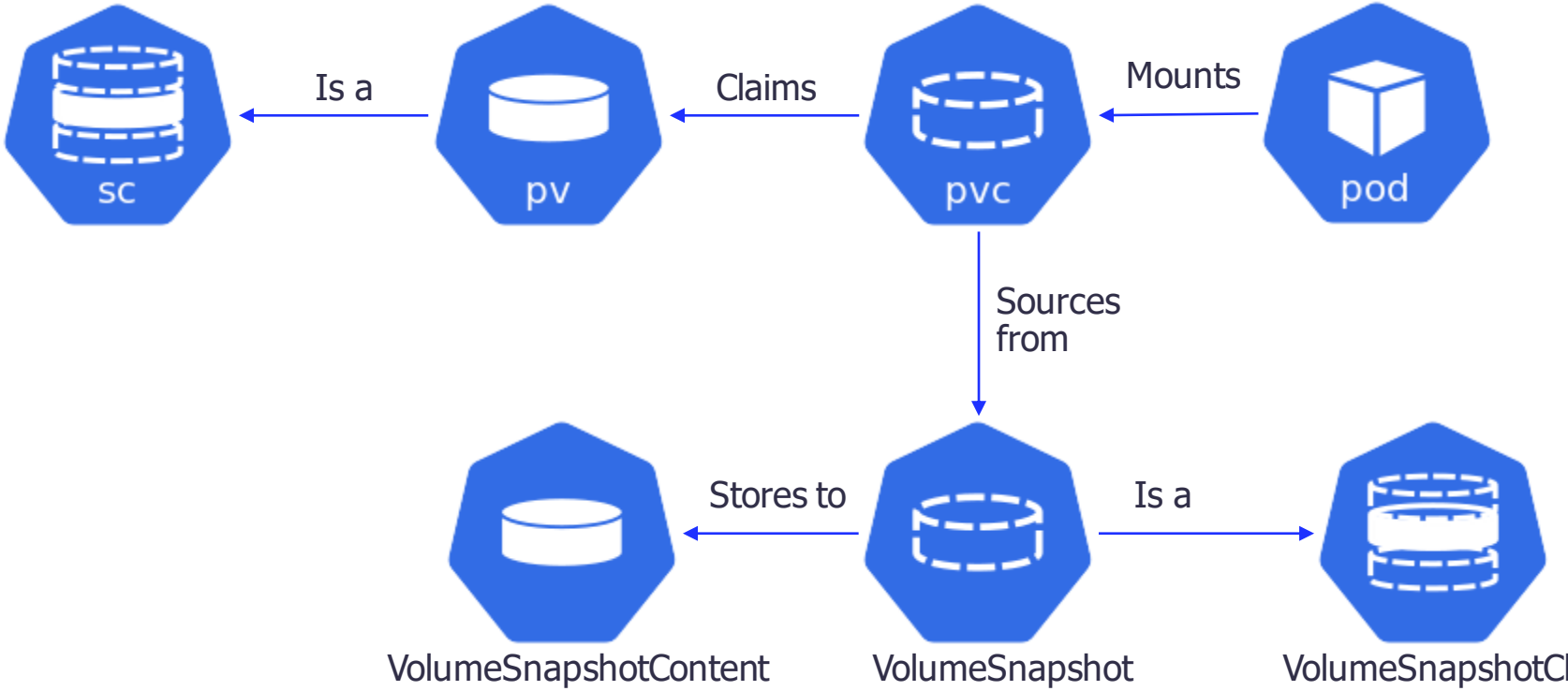
In Kubernetes, a VolumeSnapshot represents a snapshot of a volume on a storage system.

Similar to how API resources PersistentVolume and PersistentVolumeClaim are used to provision volumes for users and administrators, VolumeSnapshotContent and VolumeSnapshot API resources are provided to create volume snapshots for users and administrators.

A VolumeSnapshotContent is a snapshot taken from a volume in the cluster that has been provisioned by an administrator. It is a resource in the cluster just like a PersistentVolume is a cluster resource.

A VolumeSnapshot is a request for snapshot of a volume by a user. It is similar to a PersistentVolumeClaim.

VolumeSnapshots in Action



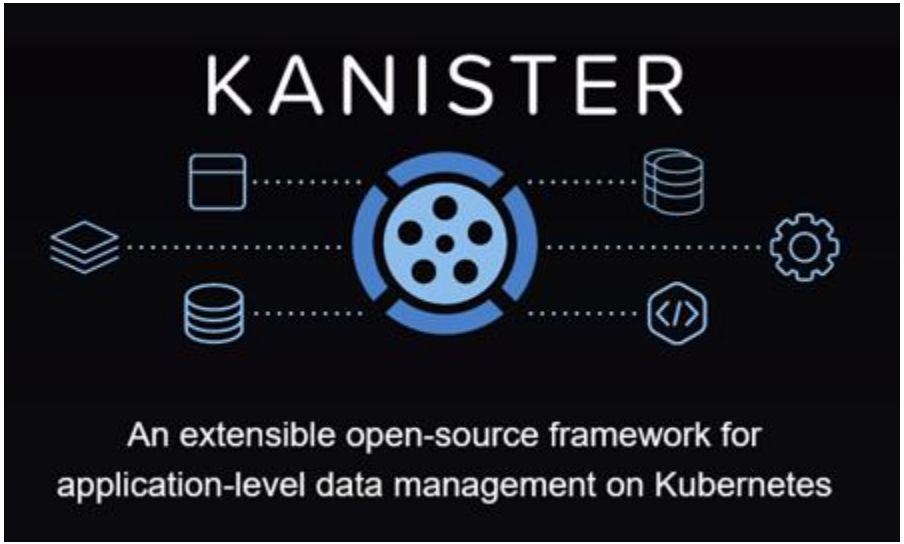
KASTEN by Veeam

Lab 3 – Storage Considerations

KASTEN by Veeam

Data Protection

KASTEN by Veeam



KANISTER allows domain experts to capture application-specific data management tasks in blueprints which can be easily shared and extended.

Kanister Framework for App-level Data Management

- Kanister Controller
  - Operator responsible for Kubernetes Custom Resources and state management
- Blueprints
  - Define workflows for backup, restore and delete operations
- ActionSets
  - Run action to backup, restore and delete
- Profiles
  - Define target destination for backups or sources for restores
- Tools
  - *kanctl*
  - *kando*

Blueprints

- Define how to backup, restore and delete
- Building blocks:

Actions:
Phases:
Kanister Functions:
Commands (Args)

Example Blueprint - Mongodb

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

Example Blueprint - Mongodb

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
  name: mongodb-blueprint
actions:
  backup:
    type: StatefulSet
```

Example Blueprint - Mongodb

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
  name: mongodb-blueprint
actions:
  backup:
    type: StatefulSet
    outputArtifacts:
      cloudObject:
        keyValue:
          path: '/mongodb-replicaset-backups/{{ .StatefulSet.Name }}/rs_backup.gz'
```

Example Blueprint - Mongodb

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
metadata:
  name: mongodb-blueprint
actions:
  backup:
    type: StatefulSet
    outputArtifacts:
      cloudObject:
        keyValue:
          path: '/mongodb-replicaset-backups/{{ .StatefulSet.Name }}/rs_backup.gz'
    phases:
    - func: KubeTask
      args:
        namespace: '{{ .StatefulSet.Namespace }}'
        image: kanisterio/mongodb:0.67.0
        command:
        - bash
        - -c
        - mongodump --oplog --gzip --archive --host ${host} -u root -p "${dbPassword}"
```

ActionSet

- Run an action from a Blueprint
- Select a Kubernetes resource to run the action on
- Select a Profile to use as the destination or source for the action
- Building blocks:

Actions:
    Name (Name of the action from the blueprint)
        Blueprint (Name of the blueprint to run the action from)
        Object (Kubernetes resource to run the action on)
        [Profile] (Destination or source for the action)

Example ActionSet

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  generateName: s3backup-
  namespace: kanister-controller
```

Example ActionSet

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  generateName: s3backup-
  namespace: kanister-controller
spec:
  actions:
  - name: backup
    blueprint: mongodb-blueprint
```

Example ActionSet

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  generateName: s3backup-
  namespace: kanister-controller
spec:
  actions:
  - name: backup
    blueprint: mongodb-blueprint
    object:
      kind: StatefulSet
      name: mongodb-replicaset
      namespace: mongodb
```

Example ActionSet

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  generateName: s3backup-
  namespace: kanister-controller
spec:
  actions:
  - name: backup
    blueprint: mongodb-blueprint
    object:
      kind: StatefulSet
      name: mongodb-replicaset
      namespace: mongodb
    profile:
      kind: profile
      name: example-profile
      namespace: kanister-controller
```

Example ActionSet

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
metadata:
  generateName: s3backup-
  namespace: kanister-controller
spec:
  actions:
  - name: backup
    blueprint: mongodb-blueprint
    object:
      kind: StatefulSet
      name: mongodb-replicaset
      namespace: mongodb
    profile:
      kind: profile
      name: example-profile
      namespace: kanister-controller
status:
  actions:
  - artifacts:
    cloudObject:
      keyValue:
        path: '/mongodb-replicaset-backups/mongodb-replicaset/rs_backup.gz'
```

Profile

- Define the destination for backups or source for restores
- Building blocks:

Location
- *Type*
- *Bucket*
- *Endpoint*

Credential
- *Type*
- *Secret (Reference to the Kubernetes Secret)*

Example Profile

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Profile
metadata:
  generateName: s3profile-
  namespace: kanister-controller
```

Example Profile

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Profile
metadata:
  generateName: s3profile-
  namespace: kanister-controller
location:
  type: s3Compliant
  bucket: kanister-backup
```

Example Profile

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Profile
metadata:
  generateName: s3profile-
  namespace: kanister-controller
location:
  type: s3Compliant
  bucket: kanister-backup
credential:
  type: keyPair
  keyPair:
    idField: example_key_id
    secretField: example_secret_access_key
    secret:
      apiVersion: v1
      kind: Secret
      name: example-secret
      namespace: example-namespace
```

Kanister CLI Tools

- kanctl
  - *CLI to create Kanister Profile CRs and ActionSets*
- kando
  - *CLI used within containers to push and pull backup data to and from an object store location*

Execution Walkthrough

Controller

Blueprint

Database
Workload

Execution Walkthrough

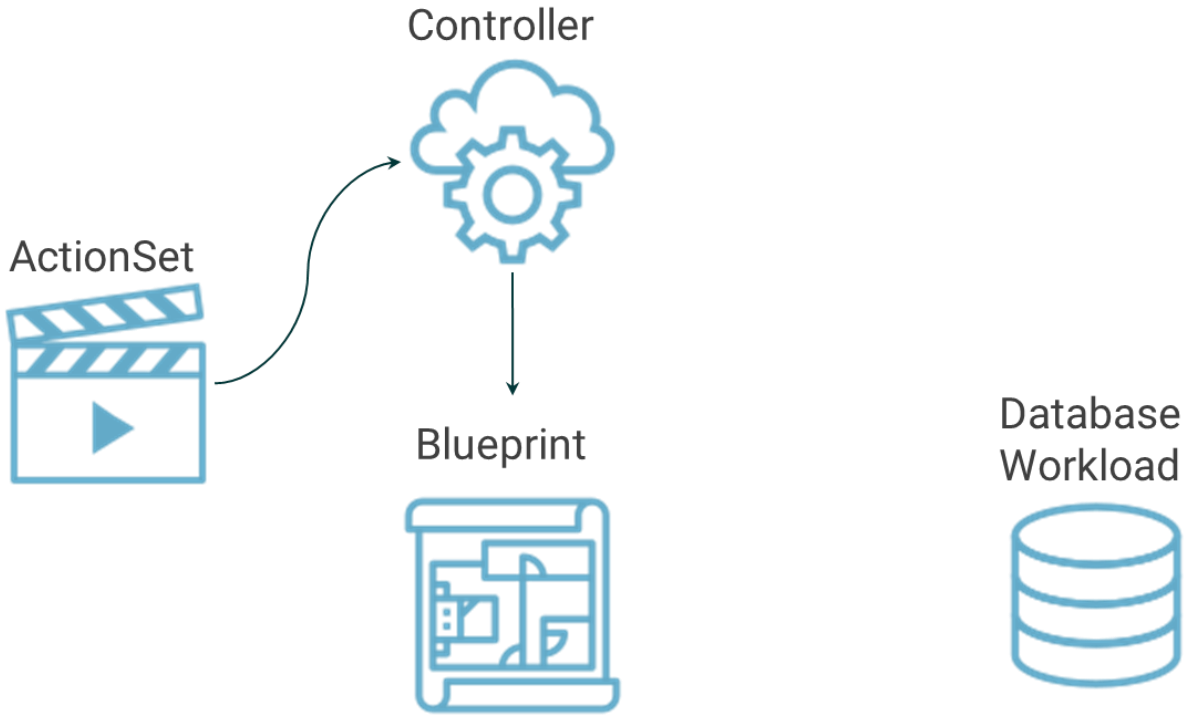
Controller

ActionSet

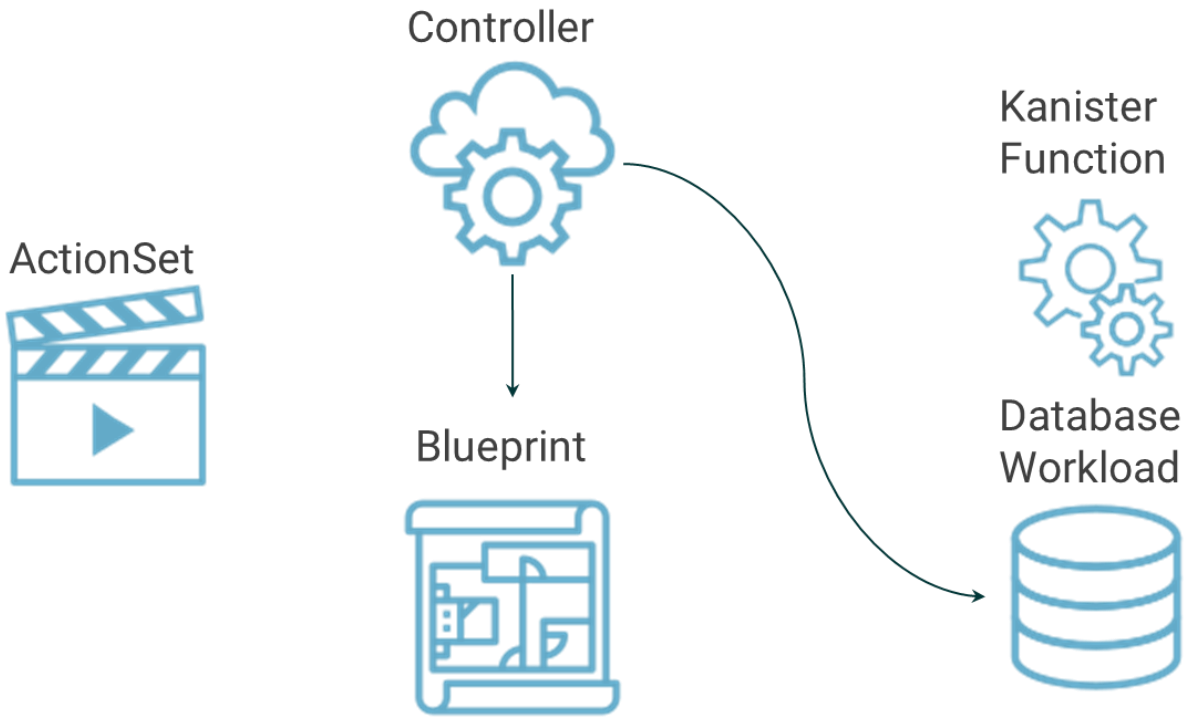
Blueprint

Database
Workload

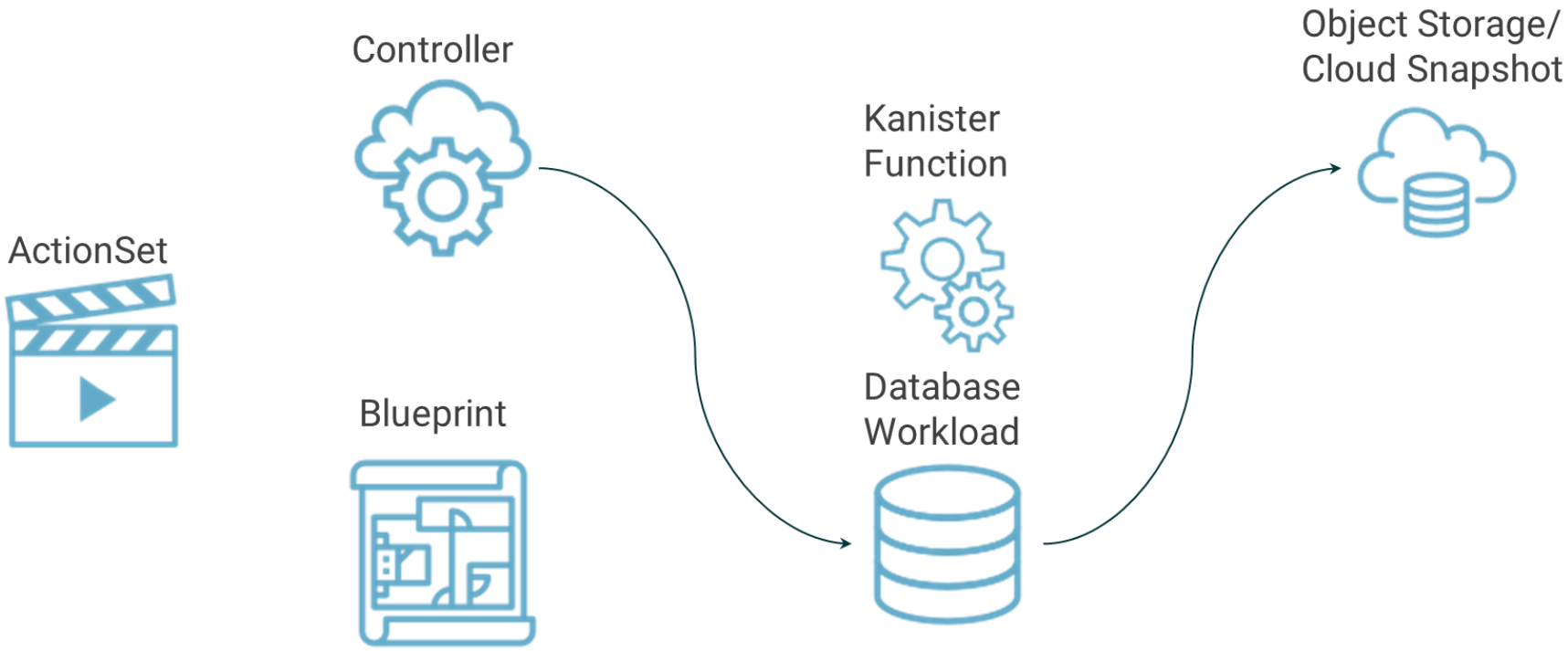
Execution Walkthrough



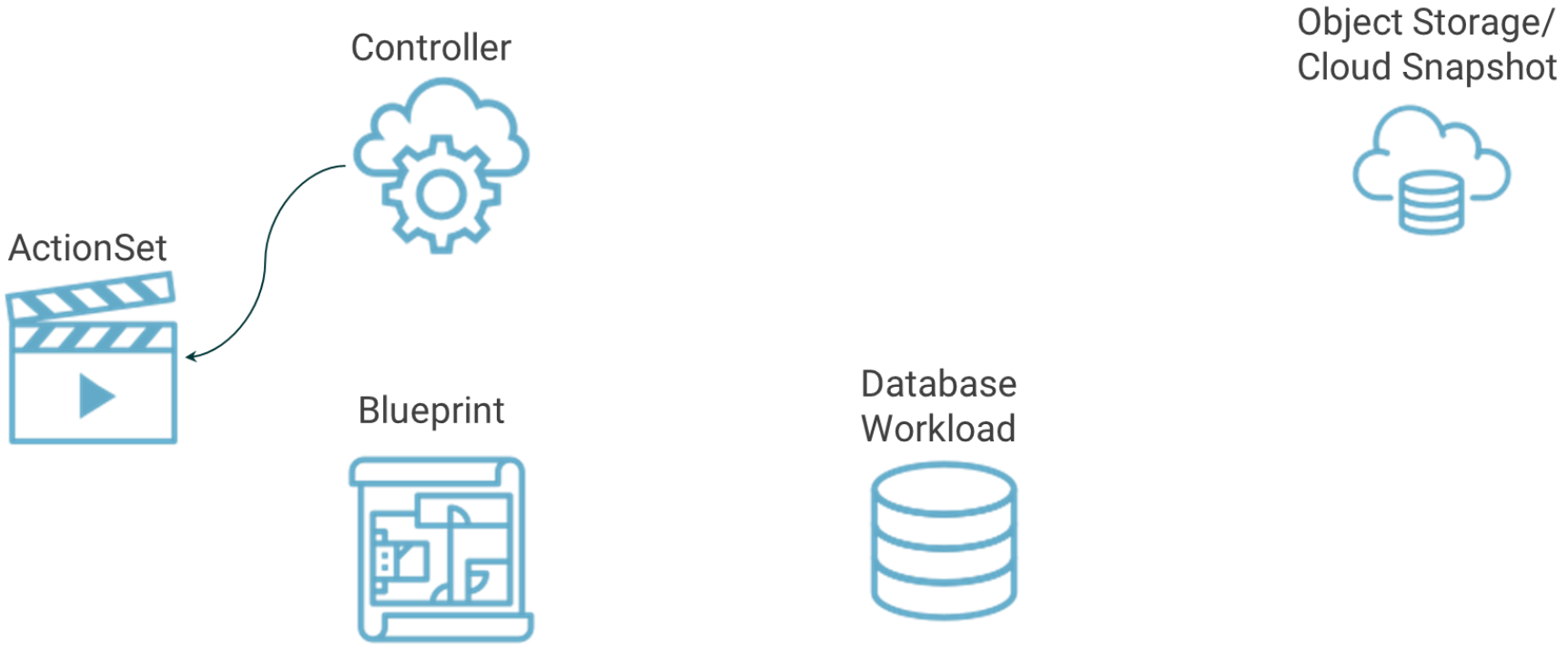
Execution Walkthrough



Execution Walkthrough



Execution Walkthrough



Functions & Providers

Kanister Functions

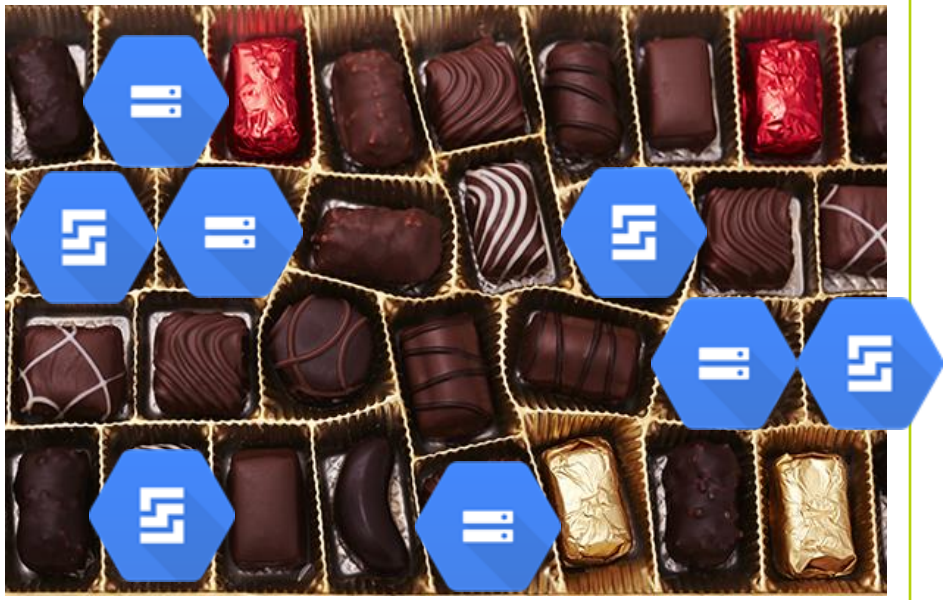
- Custom Logic
  - *KubeExec (All)*
  - *KubeTask*
- Resource Lifecycle
  - Scale up/down workload
  - *KubeTask* to run kubectl
- Handle PVC
  - *Backup/RestoreData*
  - *PrepareData*
- Volume Snapshots
  - *Create/Restore*
- RDS
  - *Create/Restore*
  - *CopyToRegion*

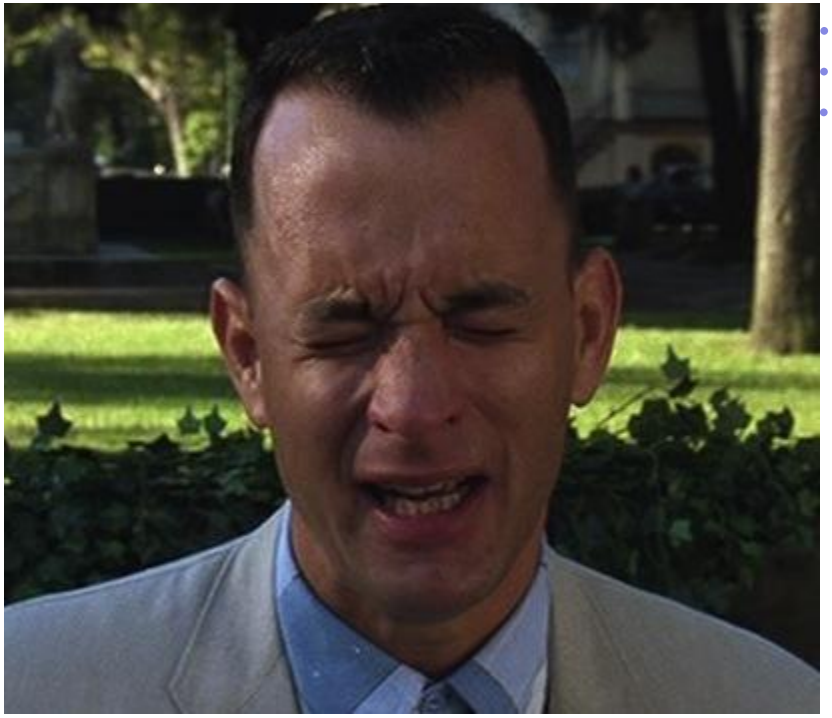
Providers Supported

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

Storage Performance

How do you know what you are getting?





Persistent Storage Options are not equal

- Choosing the right storage?
- Is the storage fast enough for my applications and workload?
- Are we using over-the-top storage systems for our workloads?
- Is your storage ready for data protection
- The ideal situation for businesses, you have access to the best and fastest storage.
- But financial and technical constraints!

Debugging storage issues in the field

- Is your storage setup correct?
- Is your storage ready for data protection?
- Is your storage appropriate for your workloads?
- How can you benchmark your storage?

Explore your Kubernetes Storage options

Identify

The various storage options present in the cluster

Validate

If the storage options are configured correctly

Evaluate

The storage using common benchmarking tools like FIO

Other Storage Considerations

Volume Health Monitoring (Alpha)

Currently in Alpha

CSI volume health monitoring allows CSI Drivers to detect abnormal volume conditions from the underlying storage systems and report them as events on PVCs or Pods

If a CSI Driver supports the Volume Health Monitoring feature from the controller side, an event will be reported on the related PersistentVolumeClaim (PVC) when an abnormal volume condition is detected on a CSI volume.

You need to enable the CSIVolumeHealth feature gate to use this feature from the node side.

Lab 4 – Storage Considerations

KASTEN by Veeam

Q&A

KASTEN
by Veeam