STATEFUL APPLICATIONS IN KUBERNETES: READY FOR PRODUCTION!

Niraj Tolia, Co-Founder
@nirajtolia / ntolia@kasten.io

Julio Lopez, Member of Technical Staff
@julio5524 / julio@kasten.io
Kubernetes

Container Orchestration: Automated Deployment, Scaling, & Management
Kubernetes, the greatest thing since sliced bread?
Developer and Application Focused
Puts the needs of the application and developer first and optimizes for agility

Enforces Good DevOps Hygiene
Immutability, config as code, automation makes it easy to repave all infrastructure

Declarative Approach
A robust systems approach where the state of the world is reconciled with the expectation
Key Kubernetes Features

- **Self-Healing**: Auto restart of unhealthy containers to match service levels
- **Resource Utilization**: Better bin packing for higher resource utilization
- **Deployment Options**: Variety of upgrade deployment strategies w/ rollback options
- **Auto Scaling**: Scale applications up and down in response to load
- **Portability**: Isolates developers and applications from infrastructure
- **Service Discovery**: Familiar IP and DNS-based service discovery and load balancing
The Power of Community!
(selected) kubernetes concepts – cluster + nodes
(selected) kubernetes concepts – deployments
(selected) kubernetes concepts – deployed app
Storage Options for Kubernetes
dynamic storage provisioning
for persistent storage

01 Self Service
Allow high developer velocity, no admin in the loop

02 Portable
No references to underlying storage provider. Allows application portability

03 On-Demand
Provisioned at time of use. Lifecycle can be tied to the application.
A Persistent Volume (PV) represents provisioned storage in the cluster (e.g., NFS, iSCSI, other block, etc.). A PV’s lifecycle is independent of the container/pod that uses it.
kind: PersistentVolumeClaim
apiversion: v1
metadata:
  name: my-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: ssd
**dynamic storage provisioning**
**persistent volume claim (pvc)**

```
kind: Deployment
apiVersion: v1
metadata:
  name: my-app
spec:
template:
  spec:
    containers:
      - name: app-container
        image: alpine:3.7
        command: ["my-app.sh"]
        args: ["--datadir", "/data/my-app"]
    volumeMounts:
      - name: data-volume
        mountPath: /data
    volumes:
      - name: data-volume
        persistentVolumeClaim:
          claimName: my-claim
```
Dynamic storage provisioning

Putting it all together

- Application Definition
- PersistentVolumeClaim (PVC)
- StorageClass (SC)
- PersistentVolume (PV)
- Node
- App
- Volume

Volume mounted on node where Pod is scheduled (based on Pod -> PVC -> PV mapping)
container storage interface
the path forward

Out of Tree
Independent Development and Release Cycles, Easier to Maintain

Standard Deployment
Common deployment interface using native Kubernetes primitives

File & Block
Standardized implementation APIs for using file and block

Cross-Orchestrator
Vendor friendly. Kubernetes, Mesos, CloudFoundry,

See Managing Disk Volumes in Kubernetes SDC 2018 talk by Saad and Nikhil for more info!
other operational concerns
scheduling, backup, restore, migration

**state is meaningful**
- Instances are unique and are not interchangeable
- Access to persistent data is needed across restarts

**resiliency is complex**
- High-availability depends on instance coordination
- Frequent restarts/pre-empts destabilize service

**data is important**
- How does backup, recovery, and migrate work? See Kasten’s K10 as an example!
- Resource contention concerns
Developer and Operator Support
StatefulSets

support for stateful applications

- Stable Identifiers
  Stable network identifiers for applications that depend on this

- Stable Persistence
  Includes persistent mapping across pod restarts and reschedules

- Ordered Operations
  Ordered and graceful deployment, scaling, termination

- Update Operations
  Rolling updates with restrictions
the operator design pattern
to deploy and manage apps

human ops knowledge → software

Observe

Analyze

Act

Support Complex Ops
Backups, Recovery, Scaling, Upgrades

Active Reconciliation
Reconcile desired vs. actual state

SDK-based
Easy to get started with multiple SDKs.
Still a few sharp edges though.

Extensible
Developer-extensible via
CustomResourceDefinitions
kanister: A framework for application-level data management

- Supports complex distributed applications
- Separates mechanism from policy/orchestration
- Allows for unified schedulers and monitoring
- Clean API allows for developer extensions

[https://github.com/kanisterio](https://github.com/kanisterio)
operator

high-level overview

Application

Action Request
(Custom Resource)

Controller
kanister operator example
postgresql backup

1. Object Creation
2. Base + WAL Setup
3. Base + WAL Shipping
4. Status Update

Object Storage
Backup Request Object (Custom Resource)
Kanister Controller
Kanister Blueprints

PostgreSQL + WAL-E
KubeExec
apiVersion: cr.kanister.io/v1alpha1
kind: ActionSet
spec:
  actions:
    - name: backup
      blueprint: postgresql
      object:
        kind: StatefulSet
        name: postgresql-cluster
        namespace: default
      configMaps: ...

```yaml
apiVersion: cr.kanister.io/v1alpha1
kind: Blueprint
actions:
  backup:
    type: StatefulSet
    phases:
      - func: KubeExec
        args:
          - '{{ .StatefulSet.Namespace }}'
          - '{{ index .StatefulSet.Pods 0 }}'
          - postgresql-tools-sidecar
          - bash
          - -c
          - wal-e ...
          - func: ...
  restore:
    ...
```
other awesome stateful operators

Look at the extensive list at
https://github.com/operator-framework/awesome-operators

and more...
packaging your applications
helm: the kubernetes package manager

off-the-shelf stateful “charts”
Multiple community charts available for databases, NoSQL systems, and more.

organize settings
Easy-to-use mechanisms and a single place to codify your application’s configuration options.

supports composability
Enhance or restrict based on your goals. Compose stateful services within your apps.

$ helm install stable/postgresql
   --set persistence.size=40Gi
   --set persistence.storageClass=ssd

<your-app>/requirements.yaml
dependencies:
- name: postgresql
Upcoming Developments
cloud-native databases
cockroachdb, vitess, yugabyte, and more…

scalable
Auto-scaling built to respond to load and deliver predictable performance

resilient
Fault-tolerance built in to support transparent self-healing infra

self-managing
Reduces ops overhead by automatically handling system management tasks
Local Persistent Volumes (beta)
Local Disks “done right”

- **Leverage Local Disks**
  For systems (Ceph, Cassandra, etc.) that work best on local storage

- **Common Primitives**
  Uses well-known PersistentVolume, PersistentVolumeClaim, StorageClass

- **Smarter Scheduling**
  Smarter pod scheduling and volume binding compared to hostPath

- **Expose as Block**
  Not just file system access anymore
kubernetes and state

wrapping up

Stateful is Ready for Production!

01 Platform Support
Equivalent features and concepts that made stateless successful

02 Storage Vendor Choices
Large number of storage provider choices, CSI, Portability Abstractions

03 Relational / NoSQL Systems
Support from traditional relational and NoSQL systems. First-class operators. Cloud-Native DBs.

04 Increased Production Usage
50%+ users using stateful applications - SIG-APPS Survey, Apr’18
thank you