

Supercharge Kubernetes to run Big Data and Databases

Challenges to overcome and Solutions

Partha Seetala

Chief Technology Officer, ROBIN.IO



Kubernetes is a great platform to run STATELESS workloads

Imagine if you could run STATEFUL workloads on it

Not just any STATEFUL workloads, but ones where money & effort is being spent to achieve deployment agility and management simplicity



Big Data

Hortonworks Spark
cloudera MAPR



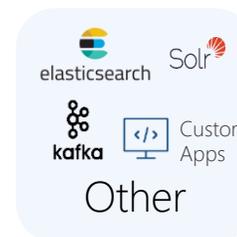
NoSQL

mongoDB
cassandra
couchDB DATASTAX



RDBMS

ORACLE MySQL
PostgreSQL MariaDB



Other

elasticsearch Solr
kafka Custom Apps

Isn't this a solved problem?

- > There are **27 Storage vendors** and **21 Network vendors** providing Storage & Networking solutions for containers and Kubernetes¹



- > CNCF²: **48%** say Storage is a big challenge, **44%** say Networking is a challenge in Kubernetes

Despite so many vendor solutions, why is it still a challenge for so many people?

¹ <https://github.com/cncf/landscape>

² <https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform>

Who am I?

Chief Technology Officer at [ROBIN.IO](https://robin.io)

before that Distinguished Engineer at Veritas/Symantec



✉ partha@robin.io

[in https://www.linkedin.com/in/parthaseetala](https://www.linkedin.com/in/parthaseetala)

[twitter @parthaseetala](https://twitter.com/parthaseetala)

ROBIN software allows you run complex Big Data and Databases on Kubernetes

(Storage + Networking + Application Workflow Management + Kubernetes)

DEPLOYMENT PROOF POINTS

11 billion security events ingested and analyzed a day

(Elasticsearch, Logstash, Kibana, Kafka)

6 petabytes under active management in a single ROBIN cluster

(Cloudera, Impala, Kafka, Druid)

400 Oracle RAC databases managed by a single ROBIN cluster

(Oracle, Oracle RAC)

CSI (Storage) challenges to overcome

Providing persistent storage to pods is a solved problem

Any storage vendor who tells you otherwise is lying 😊

4 Storage challenges to overcome:

1. Data management that is app-consistent
Snapshots, Clones, Backup, Restore, Migrate (with app-consistency)
2. Data placement that honors fault-domain constraints of an app
Data locality, affinity & anti-affinity for both pods and data
3. Handling apps that modify Root filesystem?
Configuration changes to /etc, /var and / directories
4. Guarantee performance SLAs when running on a consolidated platform
Noisy neighbor challenges when running transactional workloads

CNI (Networking) challenges to overcome

Providing connectivity, routing and networking security to pods is a largely solved problem

20+ CNI providers, 14+ Security providers

3 Networking challenges to overcome:

1. Handle IP address persistency on pod restarts/relocations
Ephemeral IP addresses (even with StatefulSets) is a challenge when apps embed IP addresses into their config/business logic
2. Multiple NICs to a pod
Needed to separate North/South client traffic from East/West management traffic
3. Connectivity to pods network from clients in a different L3 subnet
Breaks apps that embed POD IP addresses in their Web UI

CRI (Runtime) challenges to overcome

Docker containers are everywhere. Everyone understands how to build one.

4 Docker challenges to overcome:

1. Docker Shim (K8S default CRI) doesn't adequately virtualize cgroups
2. JVM sees entire host memory even if you cap the memory for container
Results in Out of Memory container kills when operating at high memory consumption
3. blkio cgroups setting is useless to avoid noisy neighbor problems
Breaks apps that embed POD IP addresses in their Web UI
4. Raw block devices access is incorrectly done
WWN management of devices is critical to avoid breaking correctness of a database

Kubernetes challenges to overcome

Don't StatefulSets and PersistentVolume/Claims address all challenges

3 Microservices challenges to overcome:

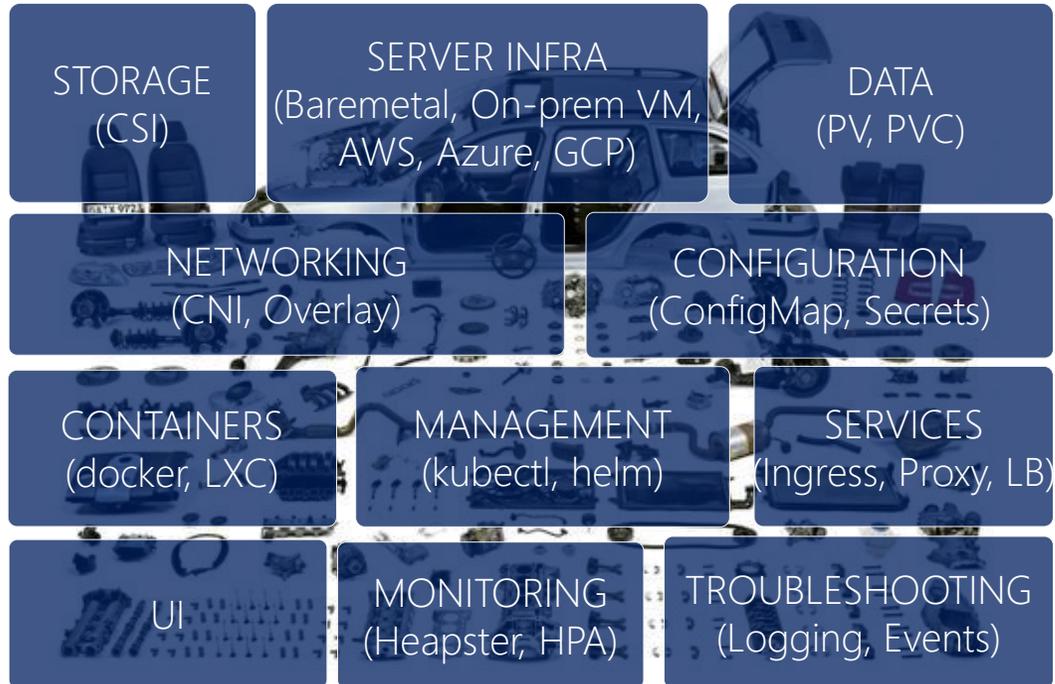
1. Most heavily used Big Data and Database platforms predate both Docker and Kubernetes
2. There are decades of built-in assumptions that don't easily fit into the microservices philosophy of K8S
3. Rewriting them to fit this model is not realistic

But aren't Operators solving this problem for us?

What about Operators?

- › Operators is custom logic for the provisioning and scaling complex workloads on Kubernetes.
CSI, CNI and CRI challenges called out earlier still need to be addressed outside an Operator
- › Big Data and Databases rely on lifecycle management which depends on external storage providers
Snapshots, Clones, Backups, Restores
- › How to handle multi-tier apps that span multiple Operators?
e.g., how to scale, snapshot, clone and backup a 3-tier app, when an Operator understands only one tier?

This is getting too complicated!!



Desired



Reality

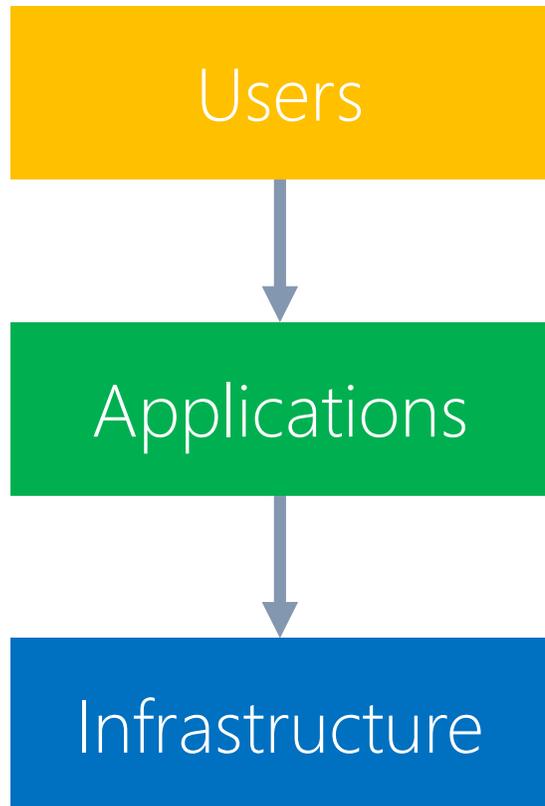


Time to reframe our thinking

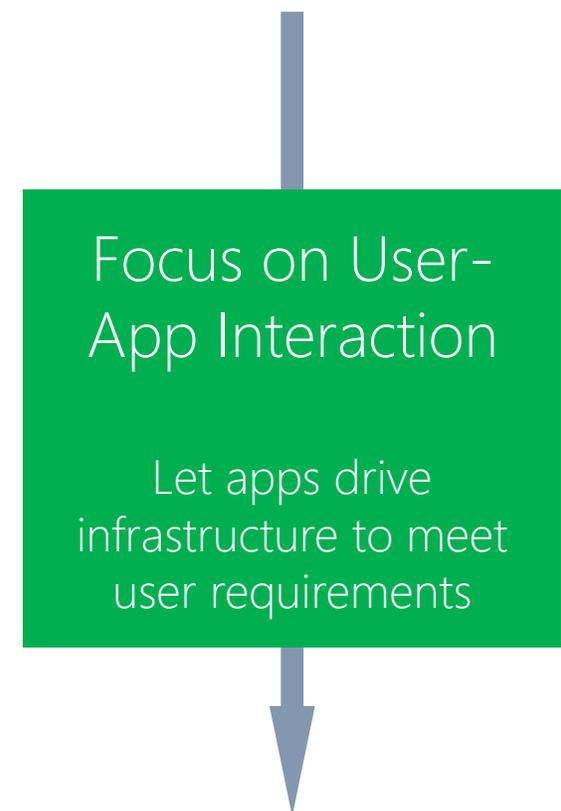
Let applications drive infrastructure to meet user requirements

(in this model application workflows configure Kubernetes, Networking and Storage)

What if we focus on Apps, not Infrastructure



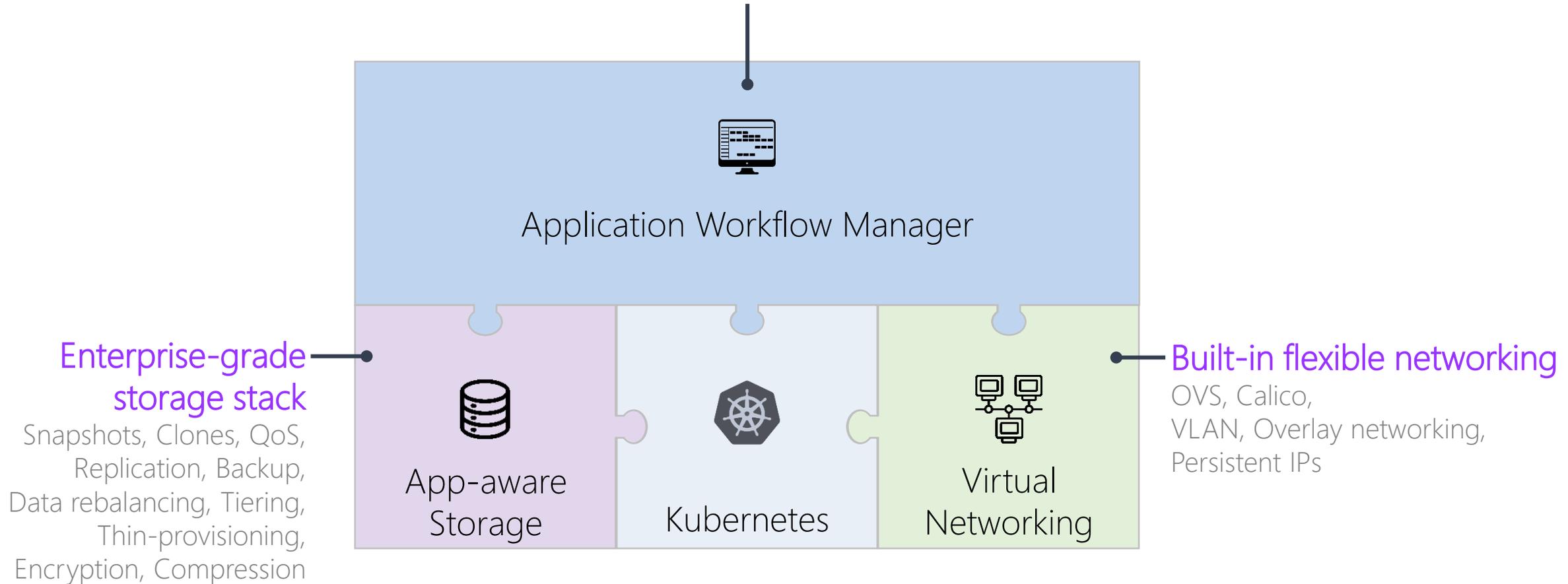
Whereas we should be looking at it in this direction



What would an ideal solution stack look like?

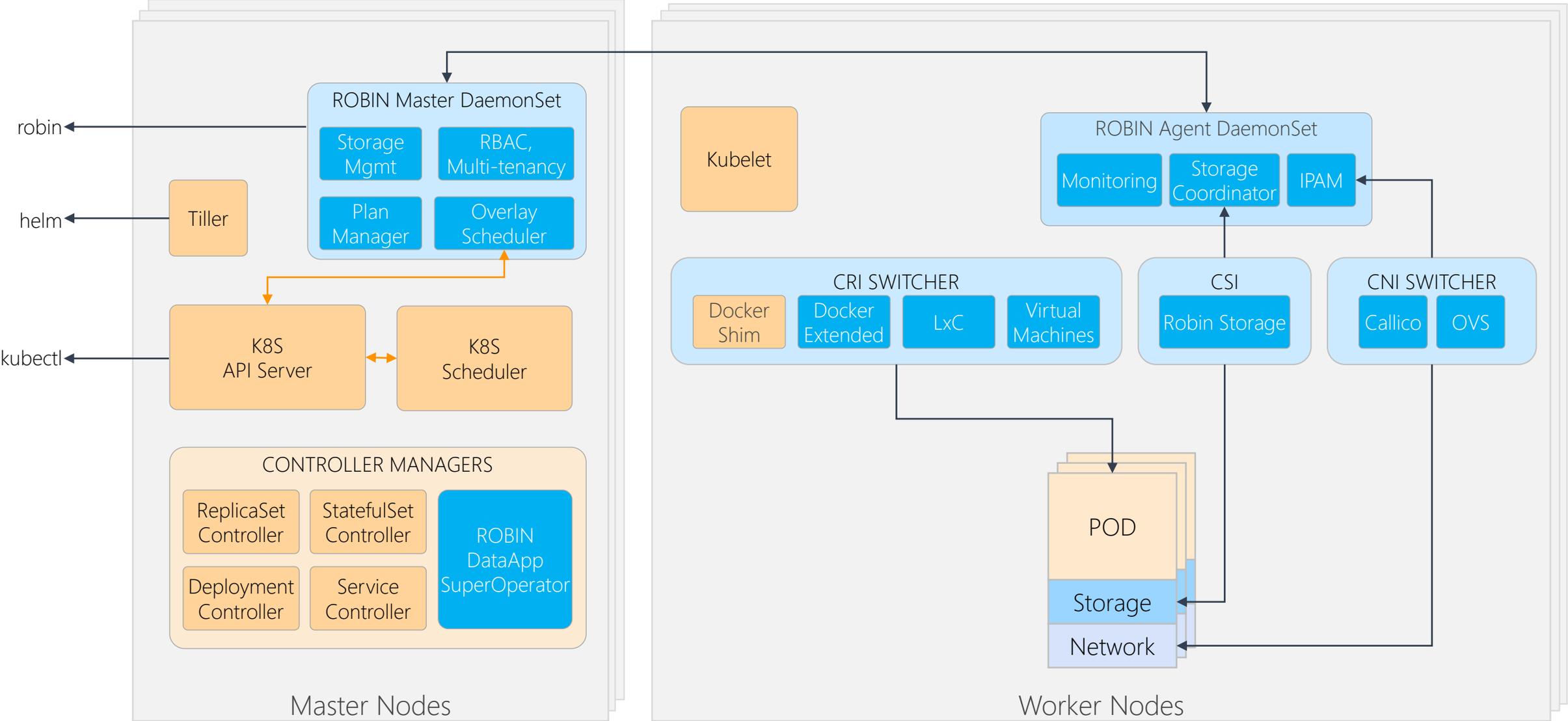
1-click application Deploy, Snapshot, Clone, Scale, Upgrade, Backup

Application workflows configure Kubernetes, Storage & Networking



Under the hood of an ideal solution

Orange is K8S components
Blue is ROBIN Components



1-Click Deployment of Big Data and Databases

The screenshot shows the 'Create Hortonworks (2.6) Application' interface. At the top, there are fields for 'APP NAME' (HDP_Cluster), 'RESOURCE POOL' (default), and 'IP SUBNET' (p1). Below this is a 'CONFIGURE COMPONENTS' section with a list of services: ambari_server, atlas, datanode, edgenode, hive, kafka, kdcserver, knox, mysql, namenode, nodemanager, oozie, ranger, resourcemanager, spark, and zookeeper. Several of these have toggle switches. A 'Specify Scale' annotation points to the 'INSTANCES' field for 'datanode' (set to 48). A 'Specify Compute' annotation points to the 'CPU CORES' (16) and 'MEMORY' (64 GB) fields. A 'Specify Storage' annotation points to the 'STORAGE' section, which shows 'log' volumes (500 GB, HDD, /var/log/) and 'data' volumes (2 TB, HDD, /hadoop). A 'Specify Data-locality, Anti/Affinity Constraints and Placement Hints' annotation points to the 'PLACEMENT RULES' section, which includes options like 'Round-robin placement of datanode containers across different nodes within different rack', 'Prevent placing more than one datanode container on the same node', 'Enforce Storage and Compute for datanode to be on same node', 'Do not place datanode on a node which is also running kafka, namenode, zookeeper', and 'Always place datanode on a node which is also running these services (click to select)'. At the bottom, there are buttons for 'Provision Application', 'Save as Template', and 'Cancel'. A 'Enable Service Components' annotation points to the 'hive' toggle switch.

Application is up and running in minutes, not weeks

- ✓ K8S components auto created (StatefulSets, PVC, Services, ...)
- ✓ Storage & Networking provisioned
- ✓ Data-locality, anti/affinity policies enforced
- ✓ Run workflow hooks to customize application

Common provisioning workflow for any Big Data, NoSQL, Database, AI/ML app

Agile DevOps for Big Data and Databases

- > **Time machine for applications**
Time travel across multiple application states
- > Clone and share entire applications for running reports, tests, and what-if analysis
- > Backup and restore entire application avoid fear of app+data loss
- > Safely upgrade application without fear of service disruption due to version incompatibilities
- > Migrate entire applications with data to cloud

The screenshot displays the ROBIN DevOps dashboard for a Cassandra cluster named 'mycassandra1'. The cluster is running since 10/24/2017 3:25:19 PM. It consists of 10 containers, 40 cores, and 80 GB of memory. The cluster is running on a 'Production' resource pool and was created from a 'cassandra 3.0' bundle. The dashboard shows the following details:

Containers	10	Resource Pool	Production	Created from bundle	cassandra 3.0
Cores	40	Root Volumes	100 GB	Created On	10/24/2017 3:25:19 PM
Memory	80 GB	Data Volumes	800 GB	Created by user	Stephen Durant

The dashboard also shows the following tabs: INFO, SNAPSHOTS, PERFORMANCE, and DEPLOYMENT INPUTS. The 'Running Containers' section is expanded to show the following details:

Container Name	Running since	Container hostname	Container IP address	CPUs	Memory	Storage	Image version	Physical hostname
mycassandra1.seed.01	6 days	vnode-81-89	10.9.81.89/16	4	8 GB	90 GB	3.0	eqx01-backend08
mycassandra1.seed.02	6 days	vnode-81-28	10.9.81.28/16	4	8 GB	90 GB	3.0	eqx01-backend09
mycassandra1.member.01	6 days	vnode-81-102	10.9.81.102/16	4	8 GB	90 GB	3.0	eqx03-flash04
mycassandra1.member.02	6 days	vnode-81-55	10.9.81.55/16	4	8 GB	90 GB	3.0	eqx03-flash09
mycassandra1.member.03	6 days	vnode-81-174	10.9.81.174/16	4	8 GB	90 GB	3.0	eqx03-flash09
mycassandra1.member.04								
mycassandra1.member.05								
mycassandra1.member.06								

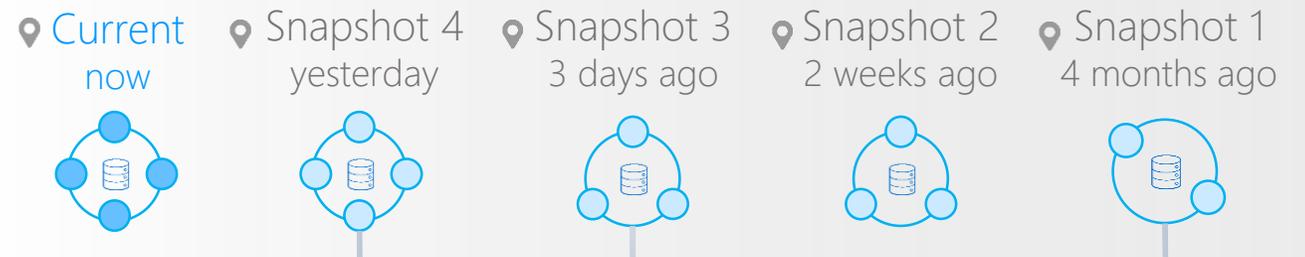
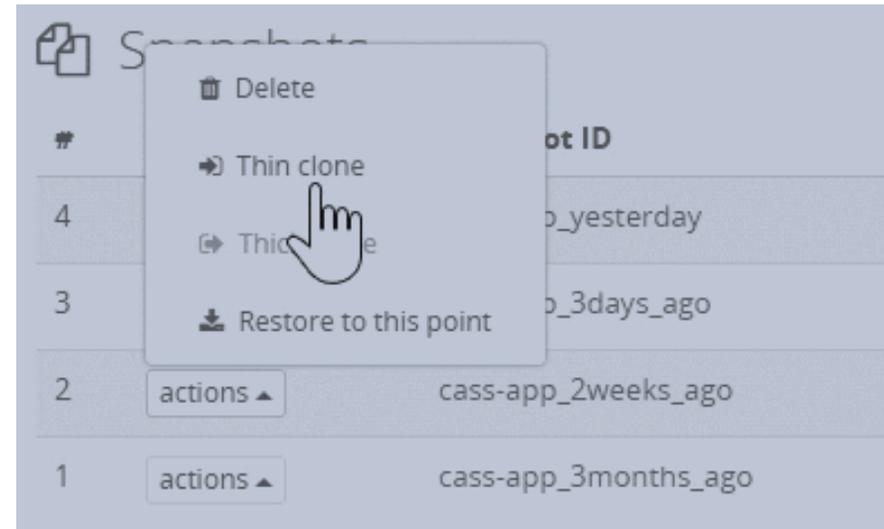
Agile DevOps for Big Data and Databases

- > **Time machine for applications**
Time travel across multiple application states
- > Clone and share entire applications for running reports, tests, and what-if analysis
- > Backup and restore entire application avoid fear of app+data loss
- > Safely upgrade application without fear of service disruption due to version incompatibilities
- > Migrate entire applications with data to cloud

The screenshot displays the ROBIN management interface for an application named 'mycassandra1'. The application is running since 10/24/2017 3:25:19 PM. Key resource details include 10 Containers, 40 Cores, 80 GB Memory, 100 GB Root Volumes, and 800 GB Data Volumes. A red box highlights the 'Snapshot' button in the 'actions' menu. A prominent blue banner states '1-click Application-consistent Snapshots'. Below this, a horizontal bar shows a sequence of snapshots: Snapshot 1, Snapshot 2, Snapshot 3, Snapshot 4, and Current, each represented by a database icon. The bottom of the screenshot shows details for individual containers, including 'mycassandra1.seed.01', 'mycassandra1.seed.02', and 'mycassandra1.member.04' through 'mycassandra1.member.06'.

Agile DevOps for Big Data and Databases

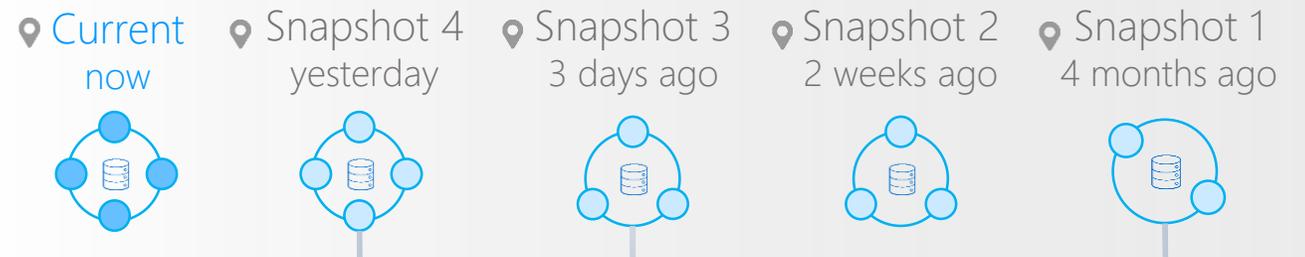
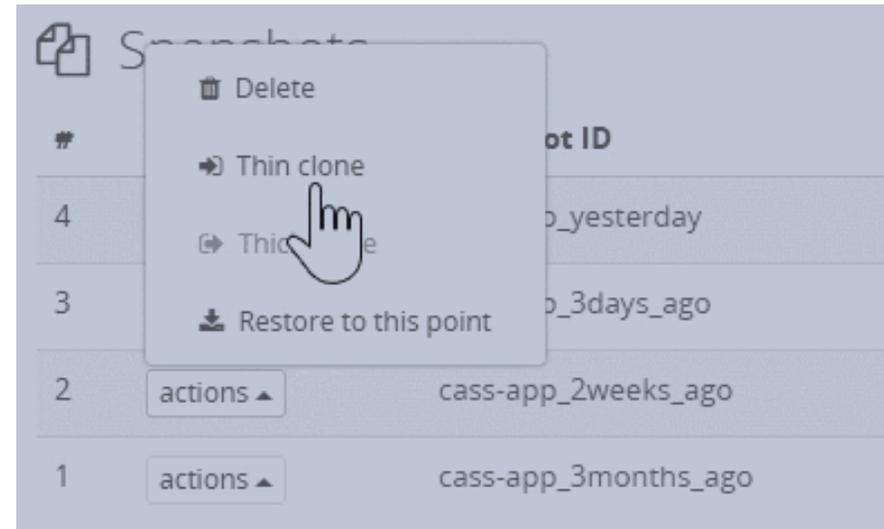
- > Time machine for applications
Time travel across multiple application states
- > **Clone and share entire applications**
for running reports, tests, and what-if analysis
- > Backup and restore entire application
avoid fear of app+data loss
- > Safely upgrade application
without fear of service disruption due to
version incompatibilities
- > Migrate entire applications with data to
cloud



1-click Ready-to-use Clones
RoW based Cloning (Ultra fast)
Clone gets different network identify

Agile DevOps for Big Data and Databases

- > Time machine for applications
Time travel across multiple application states
- > **Clone and share entire applications**
for running reports, tests, and what-if analysis
- > Backup and restore entire application
avoid fear of app+data loss
- > Safely upgrade application
without fear of service disruption due to
version incompatibilities
- > Migrate entire applications with data to
cloud



1-click Ready-to-use Clones
RoW based Cloning (Ultra fast)
Clone gets different network identify

Q&A



✉ partha@robin.io

[in https://www.linkedin.com/in/parthaseetala](https://www.linkedin.com/in/parthaseetala)

[twitter @parthaseetala](https://twitter.com/parthaseetala)

ROBIN.IO

Supercharge Kubernetes to Deliver **Big Data and Databases as-a-Service**

1-click Deploy, Scale, Snapshot, Clone, Upgrade, Backup, Migrate

