

# The Highs and Lows of Stateful Containers

Presented by Alex Robinson / Member of the Technical Staff  
@alexwritescode




# GitHub recovering after a widespread outage caused by networking and database issues

BY **TOM KRAZIT** on October 22, 2018 at 9:47 am

## Microsoft reveals train of mistakes that killed Azure in the South Central US 'incident'

Thunderbolt and lightning, Azure outage frightening

By **Richard Speed** 17 Sep 2018 at 18:39

48  SHARE ▼

Feb 10, 2017 - GitLab 

# Postmortem of database outage of January 31

Almost all real applications rely on state

When storage systems go down, so do  
the applications that use them

Containers are new and different

Change is risky

Great care is warranted when moving  
stateful applications into containers

To succeed, you must:

To succeed, you must:

1. Understand your stateful application

To succeed, you must:

1. Understand your stateful application
2. Understand your orchestration system



To succeed, you must:

1. Understand your stateful application
2. Understand your orchestration system
3. Plan for the worst

# Let's talk about stateful containers

- Why would you even want to run stateful applications in containers?
- What do stateful systems need to run reliably?
- What should you know about your orchestration system?
- What's likely to go wrong and what can you do about it?

# My experience with stateful containers

- Worked directly on Kubernetes and GKE from 2014-2016
  - Part of the original team that launched GKE
- Lead all container-related efforts for CockroachDB
  - Configurations for Kubernetes, DC/OS, Docker Swarm, even Cloud Foundry
  - AWS, GCP, Azure, On-Prem
  - From single availability zone deployments to multi-region
  - Help users deploy and troubleshoot their custom setups



**kubernetes**



**Cockroach DB**

Why even bother?

We've been running stateful services for decades

# Traditional management of stateful services

1. Provision one or more beefy machines with large/fast disks
2. Copy binaries and configuration onto machines
3. Run binaries with provided configuration
4. Never change anything unless absolutely necessary



# Traditional management of stateful services

- Pros
  - Stable, predictable, understandable
- Cons
  - Most management is manual, especially to scale or recover from hardware failures
    - And that manual intervention may not be very well practiced

# Moving to containers

- Can you do the same thing with containers?
  - Sure!
  - ...But that's not what you'll get by default if you're using any of the common orchestration systems

# So why move state into orchestrated containers?

- The same reasons you'd move stateless applications to containers
  - Automated deployment, placement, security, scalability, availability, failure recovery, rolling upgrades
    - Less manual toil, less room for operator error
  - Resource isolation
- Avoid separate workflows for stateless vs stateful applications



# Challenges of managing state

“Understand your stateful application”

What do stateful systems need?

# What do stateful systems need?

- Process management
- Persistent storage

# What do stateful systems need?

- Process management
- Persistent storage
- If distributed, also:
  - Network connectivity
  - Consistent name/address
  - Peer discovery

# What do stateful systems need?

- Process management
- Persistent storage
- If distributed, also:
  - Network connectivity
  - Consistent name/address
  - Peer discovery

# What do stateful systems need?

- Process management
- Persistent storage
- If distributed, also:
  - Network connectivity
  - Consistent name/address
  - Peer discovery

# Managing state in plain Docker containers

*"Understand your orchestration system"*

# Stateful applications in Docker

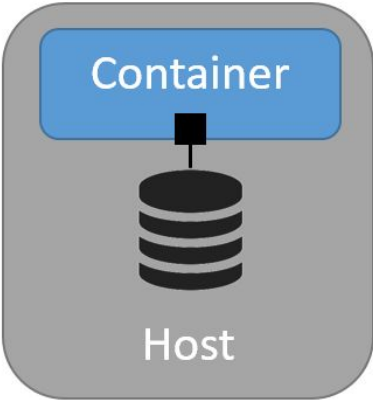
- Not much to worry about here other than storage
  - Never store important data to a container's filesystem



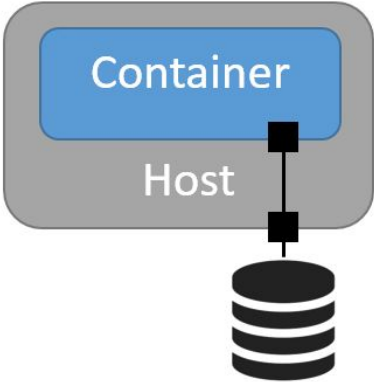
# Stateful applications in Docker



~~1. Data in container~~



2. Data on host filesystem



3. Data in network storage

# Stateful applications in Docker

- Don't:
  - `docker run cockroachdb/cockroach start`
- Do:
  - `docker run -v /mnt/data1:/data cockroachdb/cockroach start --store=/data`

# Stateful applications in Docker

- Don't:
  - `docker run cockroachdb/cockroach start`
- Do:
  - `docker run -v /mnt/data1:/data cockroachdb/cockroach start --store=/data`
- And in most cases, you'll actually want:
  - `docker run -p 26257:26257 -p 8080:8080 -v /mnt/data1:/data cockroachdb/cockroach start --store=/data`

# Stateful applications in Docker

- Hardly any different from running things the traditional way
- Automated - binary packaging/distribution, resource isolation
- Manual - everything else

# Managing State on Kubernetes

*"Understand your orchestration system"*

# Let's skip over the basics

- Unless you want to manually pin pods to nodes (see previous section), you should use either:
  - StatefulSet:
    - decouples replicas from nodes
    - persistent address for each replica, DNS-based peer discovery
    - network-attached storage instance associated with each replica
  - DaemonSet:
    - pin one replica to each node
    - use node's disk(s)

Where do things go wrong?

```
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: cockroachdb
spec:
  serviceName: "cockroachdb"
  replicas: 3
  template:
    metadata:
      labels:
        app: cockroachdb
    spec:
      containers:
        - name: cockroachdb
          image: cockroachdb/cockroach:v2.1.0
          ports:
            - containerPort: 26257
              name: grpc
            - containerPort: 8080
              name: http
          command: ["cockroach", "start", "--insecure", "--join=cockroachdb"]
```



# Don't trust the defaults!

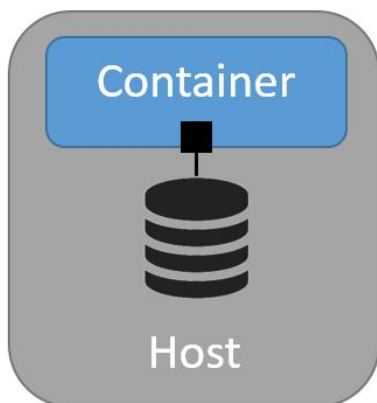
- If you don't specifically ask for persistent storage, you won't get any
  - **Always** think about and specify where your data will live

# Don't trust the defaults!

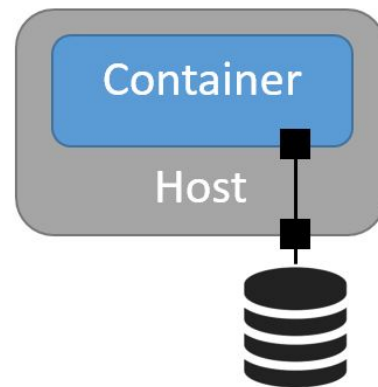
- If you don't specifically ask for persistent storage, you won't get any
  - **Always** think about and specify where your data will live



~~1. Data in container~~



2. Data on host filesystem



3. Data in network storage

# Ask for a dynamically provisioned PersistentVolume

```
    volumeMounts:
      - name: datadir
        mountPath: /cockroach/cockroach-data
    volumes:
      - name: datadir
        persistentVolumeClaim:
          claimName: datadir
    volumeClaimTemplates:
      - metadata:
          name: datadir
        spec:
          accessModes:
            - "ReadWriteOnce"
          resources:
            requests:
              storage: 100Gi
```

# Don't trust the defaults!

- Now your data is persistent
- But how's performance?



# Don't trust the defaults!

- If you don't create and request your own *StorageClass*, you're probably getting slow disks
  - Default on GCE is non-SSD (pd-standard)
  - Default on Azure is non-SSD (non-managed blob storage)
  - Default on AWS is gp2, which are backed by SSDs but with fewer IOPs than io2
- This really affects database performance

## Use a custom StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
```

```
volumeClaimTemplates:
- metadata:
  name: datadir
  spec:
    accessModes:
    - "ReadWriteOnce"
    resources:
      requests:
        storage: 100Gi
    storageClassName: fast
```

# Performance problems

- There are a lot of other things you have to do to get performance equivalent to what you'd get outside of Kubernetes
- For more detail, see

<https://cockroachlabs.com/docs/kubernetes-performance.html>

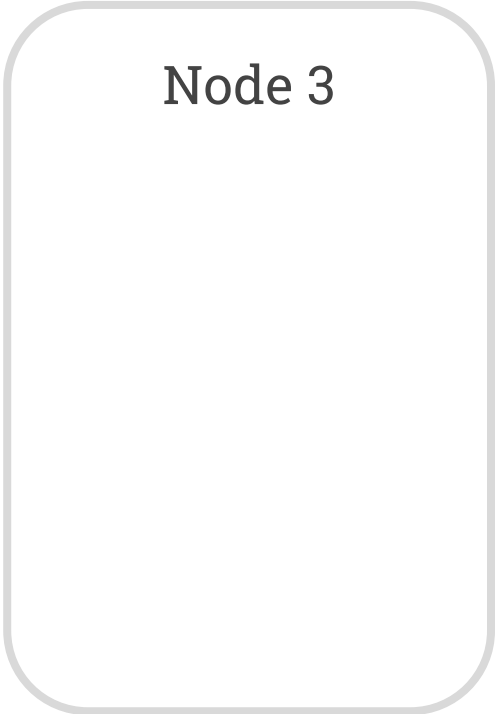
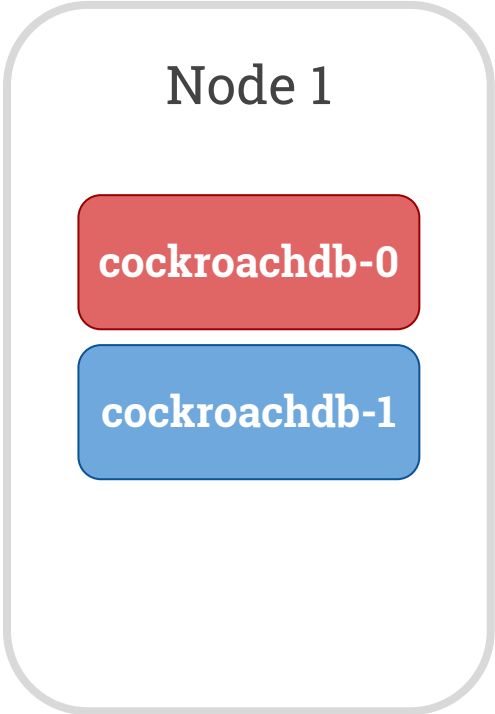
What other defaults are bad?



# What other defaults are bad?

- If you:
  - Create a Kubernetes cluster with 3 nodes
  - Create a 3-replica StatefulSet running CockroachDB
- What happens if one of the nodes fails?

# Don't trust the defaults!



# Don't trust the defaults!

- If you don't specifically ask for your StatefulSet replicas to be scheduled on different nodes, they may not be (k8s issue #41130)
  - If the node with 2 replicas dies, Cockroach will be unavailable until they come back
- This is terrible for fault tolerance
  - What's the point of running 2 database replicas on the same machine?

## Configure pod anti-affinity

```
affinity:  
  podAntiAffinity:  
    preferredDuringSchedulingIgnoredDuringExecution:  
      - weight: 100  
        podAffinityTerm:  
          labelSelector:  
            matchExpressions:  
              - key: app  
                operator: In  
                values:  
                  - cockroachdb  
          topologyKey: kubernetes.io/hostname
```

What can go wrong other than bad defaults?

## What else can go wrong?

- In early tests, Cockroach pods would fail to get re-created if all of them were brought down at once
- Kubernetes would create the first pod, but not any others

```
~ 0 $ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
cockroachdb-0	0/1	Running	0	1h

# What else can go wrong?

```
$ kubectl describe pod cockroachdb-0
```

Events:

Type	Reason	Age	From	Message
----	-----	----	----	-----
Warning	Unhealthy	4m (x420 over 1h)	kubelet, minikube	Readiness probe failed: HTTP probe failed with statuscode: 500

# Know your app and your orchestration system

- StatefulSets (by default) only create one pod at a time
- They also wait for the current pod to pass readiness probes before creating the next



# Know your app and your orchestration system

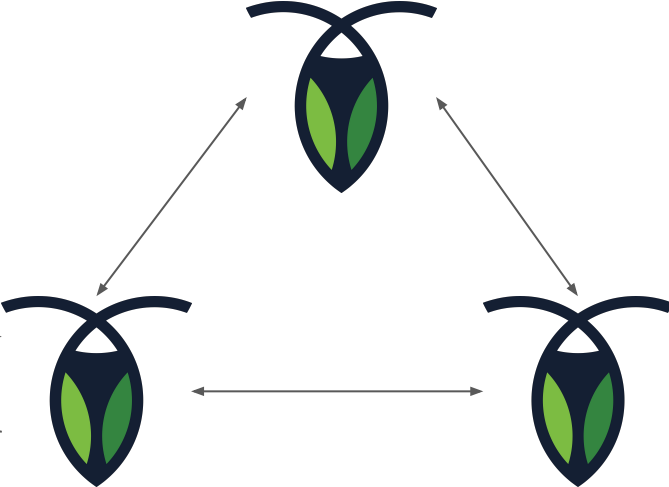
- StatefulSets (by default) only create one pod at a time
- They also wait for the current pod to pass readiness probes before creating the next
- The Cockroach health check used at the time only returned healthy if the node was connected to a majority partition of the cluster

# Before the restart

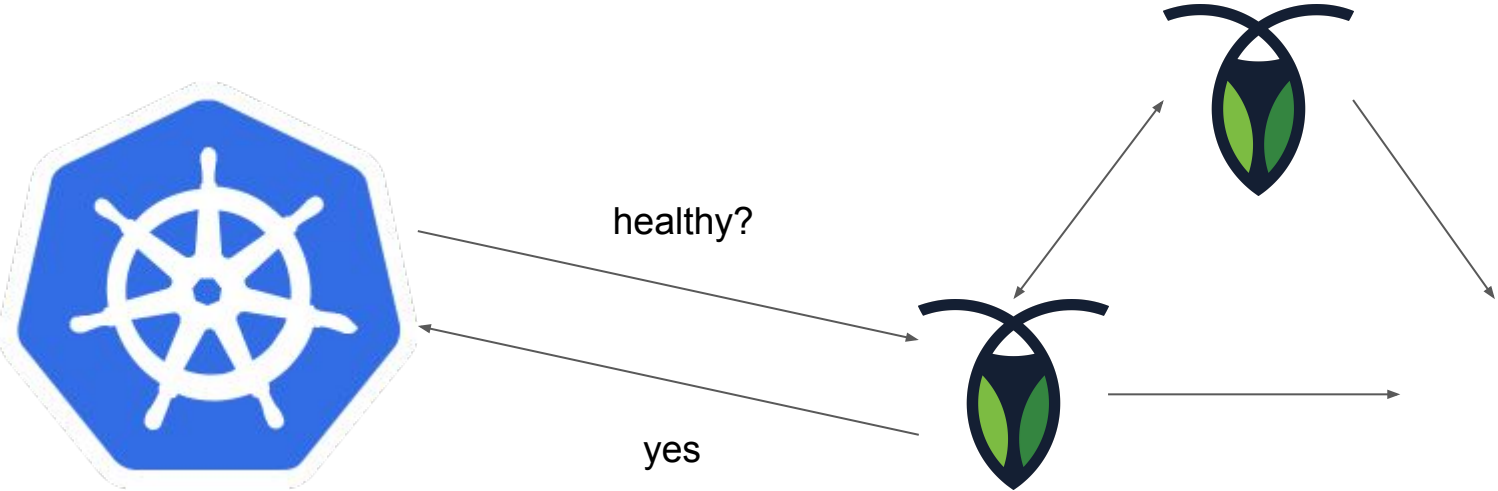


healthy?

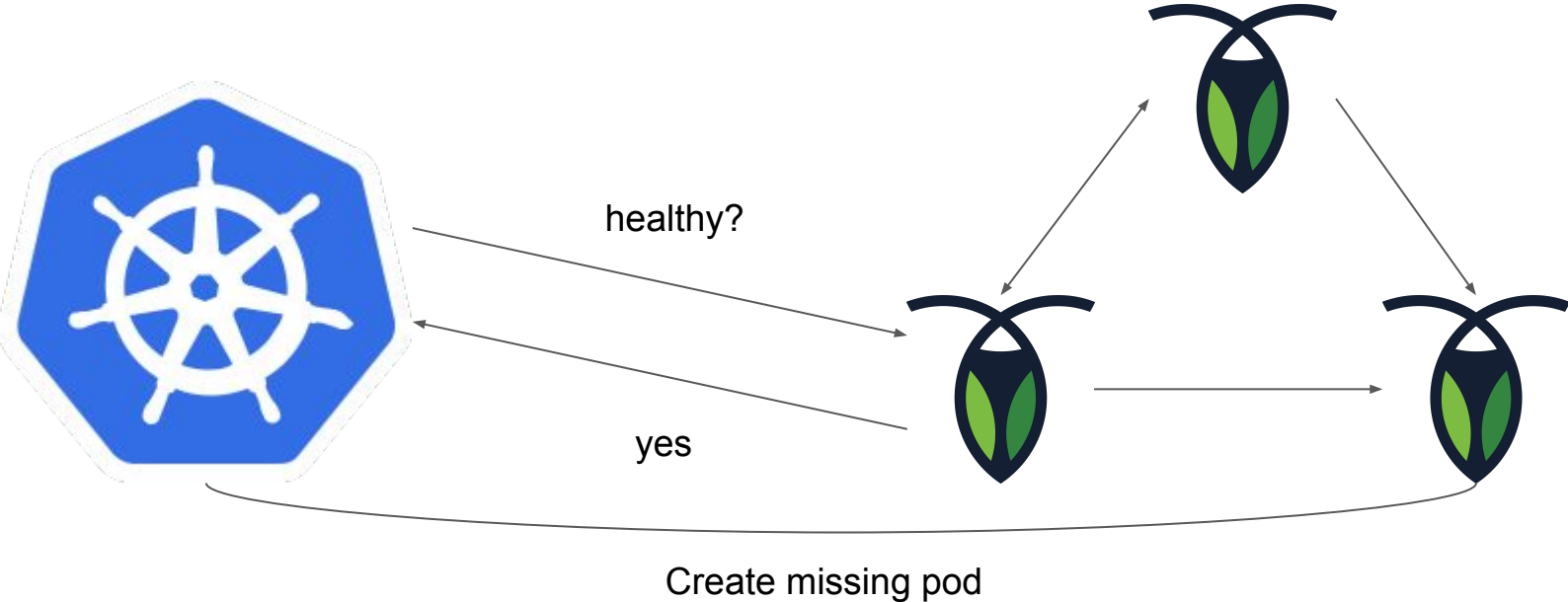
yes



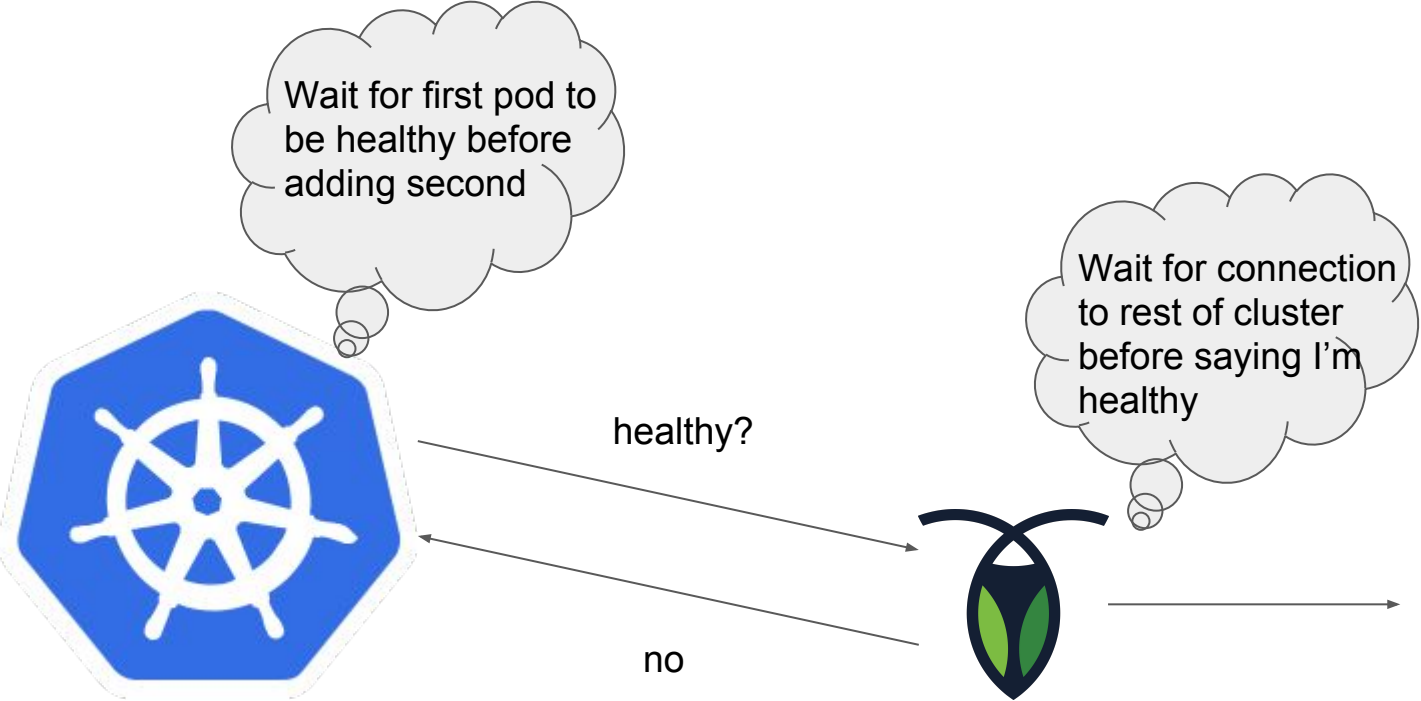
# If just one node were to fail



# If just one node were to fail



# After all nodes fail



# Solution to pod re-creation deadlock

- Keep basic liveness probe endpoint
  - Simply checks if process can respond to any HTTP request at all
- Create new readiness probe endpoint in Cockroach
  - Returns HTTP 200 if node is accepting SQL connections

# Solution to pod re-creation deadlock

- Keep basic liveness probe endpoint
  - Simply checks if process can respond to any HTTP request at all
- Create new readiness probe endpoint in Cockroach
  - Returns HTTP 200 if node is accepting SQL connections
- Now that it's an option, tell the StatefulSet to create all pods in parallel

## Other potential issues to look out for

- Set resource requests/limits for proper isolation and to avoid evictions
- No PodDisruptionBudgets by default (#35318)
- If in the cloud, don't depend on your nodes to live forever
  - Hosting services (I'm looking you, GKE) tend to just delete and recreate node VMs in order to upgrade node software
  - Be especially careful about using the nodes' local disks because of this
- If on-prem, good luck getting fast, reliable network attached storage



# Other potential issues to look out for

- If you issue TLS certificates for StatefulSet DNS addresses, don't forget to include the namespace-scoped addresses
  - "cockroachdb.default.kubernetes.svc.local" vs just "cockroachdb"
  - Needed for cross-namespace communication
  - Also don't put pod IPs in node certs - it'll work initially, but not after pod re-creation
- Multi-region stateful systems are really tough to make work
  - Both network connectivity and persistent addresses are hard to set up
  - Hopefully you went to yesterday's Cilium and Istio talks

How to get started  
Isn't this all a lot of work?

Gettings things right is far from easy

What should you do if you aren't an expert on the systems you want to use?

# How to get started

- You could take the time to build expertise

# How to get started

- You could take the time to build expertise
- But ideally someone has already done the hard work for you

# Off-the-shelf configurations

- There are great configurations available for popular OSS projects
- They've usually been made by someone who knows that project well
- They've often already been proven in production by other users

# Off-the-shelf configurations

- Kubernetes off-the-shelf configs are unfortunately quite limited
  - YAML forces the config writer to make decisions that would best be left to the user
  - No built-in method for parameter substitution
- How could a config writer possibly know your desired:
  - StorageClass
  - Disk size
  - CPU and memory requests/limits
  - Application-specific configuration options
  - etc.



# Enter: package managers

- Additional formats have been defined to make parameterizing easier
- Package creator defines set of parameters that can be easily overridden
- User doesn't have to understand or muck with YAML files
  - Just look through list of parameters and pick which need customizing

&lt;&gt; Code

! Issues 239

🔗 Pull requests 347

📊 Insights

Branch: master ▾


charts / stable /

Create new file

Upload files











Find file

History

 vsliouniaev and k8s-ci-robot Add rbac condition in alertmanager (#9007) ...

Latest commit deb5aa6 15 minutes ago

..

 <a href="#">acs-engine-autoscaler</a>	Enrich deploy. template for acs-engine-autoscaler (#5662)	6 months ago
 <a href="#">aerospike</a>	[stable/aerospike] Add cmd and args options to Aerospike config (#3856)	8 months ago
 <a href="#">anchore-engine</a>	add brady todhunter as approver to anchore-engine (#8614)	14 days ago
 <a href="#">apm-server</a>	[stable/apm-server] Elastic APM Server (#6058)	5 months ago
 <a href="#">ark</a>	[stable/ark] Quote configuration parameter backupStorageProvider.conf...	23 days ago
 <a href="#">artifactory-ha</a>	Deprecate JFrog charts (moved to https://github.com/jfrog/charts) (#7627)	2 months ago
 <a href="#">artifactory</a>	Deprecate JFrog charts (moved to https://github.com/jfrog/charts) (#7627)	2 months ago
 <a href="#">auditbeat</a>	upgrade auditbeat (#8277)	27 days ago
 <a href="#">aws-cluster-autoscaler</a>	Typo fix in aws-cluster-autoscaler/README.md (#4297)	8 months ago
 <a href="#">bitcoind</a>	typo fix in bitcoind (#4548)	7 months ago



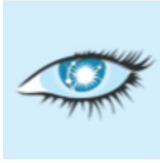
cadvisor

[Details](#)



calico

[Details](#)



cassandra

[Details](#)



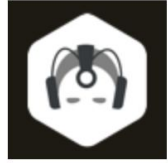
ceph

[Details](#)



ceph-dash

[Details](#)



cerebro

[Details](#)



chronos

[Details](#)



clair

[Details](#)



cockroachdb

[Details](#)



concord

[Details](#)



confluent-connect

[Details](#)



confluent-control-center

[Details](#)



# Pivotal Services Marketplace

The Pivotal Services Marketplace provides users with platform add-on services to enhance, secure, and manage applications. The catalog includes solutions from Pivotal, our Partners, and the Cloud Foundry community providing a curated selection of capabilities from data ... [Read More](#) ▾

## Most Viewed



MongoDB for  
PCF



Steeltoe



Pivotal Cloud  
Cache



Spring Cloud  
Services for  
PCF

## Our Newest Additions



Sentry



Snyk



CyberArk  
Conjur



Aqua Security

# Orchestrator package managers

- Kubernetes: Helm
  - [helm.sh](https://helm.sh)
  - [github.com/helm/charts/](https://github.com/helm/charts/)
- DC/OS: Universe
  - [universe.dcos.io](https://universe.dcos.io)
- Cloud Foundry: Pivotal Services Marketplace
  - [pivotal.io/platform/services-marketplace](https://pivotal.io/platform/services-marketplace)
- Docker: Application Packages (experimental)
  - CLI tool: docker-app

# Summary

Go forth and manage persistent state

Don't let configuration mistakes take down  
your production services

1. Understand your stateful application
2. Understand your orchestration system
3. Plan for the worst



1. Understand your stateful application
2. Understand your orchestration system
3. Plan for the worst

(or use a package manager)

# Thank You!

For more info:

[cockroachlabs.com](https://cockroachlabs.com)

[github.com/cockroachdb/cockroach](https://github.com/cockroachdb/cockroach)

[alex@cockroachlabs.com](mailto:alex@cockroachlabs.com) / [@alexwritescode](https://twitter.com/alexwritescode)

