Who are Jetstack?

We are a UK-based company that help enterprises in their path to modern cloud-native infrastructure. We develop tooling and integrations for Kubernetes to improve the user experience for customers and end-users alike.

Who are we?

@mattbates
@mattbates25

@munnerz
@JamesMunnelly
INTRODUCTION
Containers and distributed state

- Containers are here and here to stay and many of us are now using them for production services at scale
- Containers are ephemeral and can come and go - this is just for stateless applications, right?
- But a container is a... process
- Why should we treat stateful systems differently?
- Large-scale container management systems exist - why not use these systems to manage all workloads?
KUBERNETES

Anyone heard of it?

- Kubernetes handles server ‘Cattle’ to pick and choose resources
- Can be installed on many different types of infrastructure
- Abstracts away the servers so developers can concentrate on code
- Pro-actively monitors, scales, auto-heals and updates
Borg cells run a heterogeneous workload...
...
...long-running services that should “never” go down, and handle short-lived latency-sensitive requests (a few µs to a few hundred ms). Such services are used for end-user-facing products such as Gmail, Google Docs, and web search, and for internal infrastructure services (e.g., BigTable)...The workload mix varies across cells...

Our distributed storage systems such as GFS [34] and its successor CFS, Bigtable [19], and Megastore [8] all run on Borg

https://research.google.com/pubs/pub43438.html
KUBERNETES
Declarative systems management

- Declarative system description using application abstractions
  - Pods
  - Replica Sets
  - Deployments
  - Services
  - Persistent Volumes
  - Ingress
  - Secrets
  .. and many more!

An ocean of user containers
Scheduled and packed dynamically onto nodes
WORKLOADS ON KUBERNETES: PODS AND CONTAINERS
WORKLOADS ON KUBERNETES: REPLICA SET

Replica Set
WORKLOADS ON KUBERNETES: SERVICES

Replica Set

Service
WORKLOADS ON KUBERNETES: DEPLOYMENT

Deployment

Replica Set
RESOURCE LIFECYCLE
Reconciliation of desired state

Observe

Diff

Act
Consistent deployment between environments

- Systems often built for the environment they run in
  - e.g. cloud VMs, provisioned via Terraform/CloudFormation or manually
STATEFUL SERVICES

Why Kubernetes?

Visibility into management operations

- Upgrades
- Scale up/down
- Disaster recovery

Due to the way these applications are deployed, it can be difficult and inconsistent to record and manage cluster actions
STATEFUL SERVICES
Why Kubernetes?

Self-service distributed applications

- Who can perform upgrades? (authZ)
- How do we scale?
- These events must be coordinated with operations teams

Putting a dependence on central operations teams to coordinate maintenance events = time = money
STATEFUL SERVICES

Why Kubernetes?

Automated cluster actions

- HorizontalPodAutoscaler allows us to automatically scale up and down
- Teams can manage their own autoscaling policies
Why Kubernetes?

Centralised monitoring, logging and discovery

- Kubernetes provides these services already that we can reuse these for all kinds of applications
  - Prometheus
  - Labelling
  - Instrumentation
LAYING THE GROUNDWORK
Features developed by the project in previous releases

1.1 Volume plugins
PersistentVolume
PersistentVolumeClaim

1.2 Dynamic provisioning

1.3 PetSet (alpha)

1.4 StorageClasses
New volume plugins

1.5 StatefulSet (beta)

1.6 StatefulSet upgrades

1.7 Local storage (alpha)

1.8 Volume resize and snapshot

1.9 CSI (alpha)

Workloads API (apps/v1)
STATEFULSET
Unique and ordered pods

StatefulSet
Service

API Server

StatefulSet Controller

pet-0.
pet.default...
PV-0
PVC-0

pet-1.
pet.default...
PV-1
PVC-1

pet-2.
pet.default...
PV-2
PVC-2
“Helm is a tool for managing Kubernetes charts. Charts are packages of pre-configured Kubernetes resources.”

github.com/kubernetes/helm
# HELM CHARTS

Many integrations exist - e.g. see the Helm charts repo...

<table>
<thead>
<tr>
<th>Repo</th>
<th>Description</th>
<th>Days Ago</th>
</tr>
</thead>
<tbody>
<tr>
<td>acs-engine-autoscaler</td>
<td>fix typo in acs-engine-autoscaler-readme (#3889)</td>
<td>2</td>
</tr>
<tr>
<td>aerospike</td>
<td>[stable/aerospike] Add cmd and args options to Aerospike config (#3856)</td>
<td>5</td>
</tr>
<tr>
<td>anchor-engine</td>
<td>fix README.md typo (#3556)</td>
<td>21</td>
</tr>
<tr>
<td>artifactory</td>
<td>Update README.md with correct default value (#3877)</td>
<td>3</td>
</tr>
<tr>
<td>aws-cluster-autoscaler</td>
<td>Convert registry to k8s.gcr.io (#3180)</td>
<td>2 months</td>
</tr>
<tr>
<td>bitcoind</td>
<td>Add bitcoind cryptocurrency chart (#3644)</td>
<td>2</td>
</tr>
<tr>
<td>buildkite</td>
<td>[stable/buildkite] Change name of Docker credentials in Pod (#3627)</td>
<td>21</td>
</tr>
<tr>
<td>centrifugo</td>
<td>[stable/centrifugo] #1785 namespace defined templates with chart name (...)</td>
<td>6</td>
</tr>
<tr>
<td>cert-manager</td>
<td>cert-manager update with expanded docs. Remove creating TPR support. (...)</td>
<td>24</td>
</tr>
<tr>
<td>chaoskube</td>
<td>[stable/chaoskube] #1899 add nodeSelector for chaoskube (#3067)</td>
<td>2 months</td>
</tr>
<tr>
<td>chronograf</td>
<td>Fix typos: seperated -&gt; separated (#3712)</td>
<td>12</td>
</tr>
<tr>
<td>cluster-autoscaler</td>
<td>Allowing configurable sslCertPath for cluster autoscaler (#3247)</td>
<td>16</td>
</tr>
<tr>
<td>cockroachdb</td>
<td>Update readme to reflect move from incubator to stable for cockroachdb</td>
<td>5</td>
</tr>
<tr>
<td>concourse</td>
<td>[stable/concourse] fixed incorrect values for gitlab auth secrets (#3927)</td>
<td>2</td>
</tr>
<tr>
<td>consul</td>
<td>consul-readability - seperate resources (#3078)</td>
<td>3</td>
</tr>
<tr>
<td>coredns</td>
<td>CoreDNS chart: update to latest version (#2771)</td>
<td>3 months</td>
</tr>
<tr>
<td>cosscale</td>
<td>[stable/coscale] #1785 namespace defined templates with chart name (#...</td>
<td>6 months</td>
</tr>
<tr>
<td>dask-distributed</td>
<td>[stable/dask-distributed] #1785 namespace defined templates with chart...</td>
<td>5 months</td>
</tr>
<tr>
<td>datadog</td>
<td>[Datadog] Fix kubeStateMetrics.enabled in values.yaml (#3619)</td>
<td>21</td>
</tr>
</tbody>
</table>
STATEFUL SERVICES
All distributed systems are not equal

Leader elected quorum
(e.g. etcd, ZK, MongoDB)

Active-active / multi-master
(e.g. MySQL Galera, Elasticsearch)

etc..
HELM CHARTS

Problems encountered

Point-in-time management

- Resources are only modified when an administrator updates them
- This is a non-starter for self-service applications

We’re back to waking up at 3am to our pagers
HELM CHARTS
Problems encountered

Failure handling

- This requires an administrator to intervene
- Prone to errors, and requires specialist knowledge

We’re back to waking up at 3am to our pagers
No native provisions for understanding the applications state

- There’s no way to quickly see the status of a deployment in a meaningful way
Difficult to understand why and what is happening

- Opaque ‘preStop’ hook allows us to run a script before the main process is terminated

```yaml
lifecycle:
  preStop:
    exec:
      command: ["/bin/bash", "/pre-stop-hook.sh"]
```
OPERATOR PATTERN
Application-specific controllers that extend the Kubernetes API

“An Operator represents human operational knowledge in software to reliably manage an application.” (CoreOS)
OPERATOR PATTERN
Application-specific controllers that extend the Kubernetes API

- Follows the same declarative principles as the rest of Kubernetes
- Express desired state as part of your resource specification
- Controller ‘converges’ the desired and actual state of the world
OPERATOR PATTERN
Application-specific controllers that extend the Kubernetes API

Examples include:

- etcd-operator ([https://github.com/coreos/etcd-operator](https://github.com/coreos/etcd-operator))
- service-catalog ([https://github.com/kubernetes-incubator/service-catalog](https://github.com/kubernetes-incubator/service-catalog))
- metrics ([https://github.com/kubernetes-incubator/custom-metrics-apiserver](https://github.com/kubernetes-incubator/custom-metrics-apiserver))
- navigator ([https://github.com/jetstack/navigator](https://github.com/jetstack/navigator))
CUSTOM RESOURCES
Standing on the shoulders of Kubernetes

- API “as a service”
- Kubernetes API primitives for ‘custom’ types
  - CRUD operations
  - Watch for changes
  - Native authentication & authorisation

→ ~ kubectl get elasticsearchclusters
CUSTOM RESOURCES
Standing on the shoulders of Kubernetes

CustomResourceDefinition (CRD)

- Quick and easy. No extra apiserver code
- Great for simple extensions
- No versioning, admission control or defaulting

CUSTOM RESOURCES
Standing on the shoulders of Kubernetes

Custom API server (aggregated)

- Full power and flexibility of Kubernetes
  Similar to how many existing APIs are created
- Versioning, admission control, validation, defaulting
- Requires etcd to store data

Cassandra on Kubernetes

Let’s see it in action
WHAT’S GOING ON
Cassandra on Kubernetes

Native Kubernetes resources are created

- StatefulSets: cass-europe-west2-a, cass-europe-west2-b
- Load Balancers/Services: client-service, discovery-service
- Persistent Disks:
  - cass-europe-west2-a-0, cass-europe-west2-a-1, cass-europe-west2-a-2
  - cass-europe-west2-b-0, cass-europe-west2-b-1, cass-europe-west2-b-2
- Workload identities: cass-europe-west2-a, cass-europe-west2-b
WHAT’S GOING ON
Cassandra on Kubernetes

Custom ‘entrypoint’ code runs before Cassandra starts

StatefulSet

cass-europe-west2-a

Pod

cass-europe-west2-a-0

Pod

cass-europe-west2-a-1

Pod

cass-europe-west2-a-2

Pod

cass-europe-west2-a-3
Custom ‘entrypoint’ code runs before Cassandra starts

1. Lookup peers
2. Configure database with peers
3. Launch cassandra process

‘Operator’ loses control of process runtime
OPERATOR PATTERN

Problems encountered

Application state information collection is varied

- Kubernetes usually provides the ability to inspect with `kubectl describe`
OPERATOR PATTERN
Problems encountered

Reimplementing large parts of Kubernetes

- Limitations in StatefulSet result in the entire controller being reimplemented
- We should be building on these primitives, not recreating them
OPERATOR PATTERN
Problems encountered

Integrating with synchronous APIs reliably

- No easy way to see if ‘nodetool decommission’ succeeded
- Makes assuredly executing cluster infrastructure changes difficult

This is on account of the operator losing control after the process has started
Navigator
Co-located application intelligence
NAVIGATOR

Motivations

● Pro-actively monitor and heal applications

● Reduce the operational burden on teams by making management of complex applications as easy as any other Kubernetes resource

● Make it easy to understand the state of the system

● Re-use existing Kubernetes primitives - don’t reinvent the wheel

● Providing a reliable and flexible building block for integrating with the varied and sometimes difficult database APIs/management tools
Underlying orchestrator can be swappable (e.g. OpenShift, K8s, raw VMs, etc.)

Pilots talk only to ‘navigator-apiserver’ - this allows to easily embed in other envs

navigator-controller-manager creates resources (e.g. deployments, secrets) in target orchestrator

navigator-apiserver follows Kubernetes API conventions, so can be aggregated
Follows the ‘operator pattern’

Abstracts configuration of complex topologies (i.e. automated rack awareness, sharding)

Manages the lifecycle of applications over time

Provides a common and familiar interface for modifying applications

Validates configurations and helpfully rejects invalid requests
PILOTS - COLOCA TED INTELLIGENCE

Pilots alongside our processes

- Pilot ‘wraps’ the Elasticsearch process
- Performs operation on the underlying database node
- Updates the Navigator API with information about the state of the node
- ‘GenericPilot’ to make it easy to extend
- Similar to kubelet
Examples of information reported to Pilots:
- Node’s reported version
- Amount of data on node
- Node health

Leader elected Pilots also report overall cluster status

This information influences which ‘Action’ is taken
$ kubectl create -f elasticsearch-cluster.yaml
Providing sensible and safe defaults makes it easier for developers to consume complex applications ‘as a service’

```yaml
apiVersion: navigator.jetstack.io/v1alpha1
description: ElasticsearchCluster
metadata:
  name: demo
spec:
  ## Omitting the minimumMasters fields will cause navigator to automatically
  ## determine a quorum of masters to use.
  # minimumMasters: 2
```
Elasticsearch scale-up and upgrade

Actions in action
ACTIONS

Transitioning cluster state with Actions

- A small unit of work to perform
- Can be reasoned about and debugged by users through ‘kubectl describe’
What constitutes an Action?

- Upgrade
- Scale
- Backup
- Apply new configuration
- Create or delete a node pool
- Adjust resources assigned to a node pool
- Resize persistent disk
$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'
ACTIONS
Transitioning cluster state with Actions

$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'

1. Observes change

Elasticsearch upgrade action
https://github.com/jetstack/navigator/tree/master/pkg/controllers/elasticsearch/actions
ACTIONS
Transitioning cluster state with Actions

$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'

1. Observes change
2. Evaluates each ‘Pilot’ resource one at a time

Elasticsearch upgrade action

https://github.com/jetstack/navigator/tree/master/pkg/controllers/elasticsearch/actions
$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'

1. Observes change
2. Evaluates each ‘Pilot’ resource one at a time
   a. Is the node healthy?
   b. Is the node already at the desired version?
   c. Is the cluster healthy?

Elasticsearch upgrade action

https://github.com/jetstack/navigator/tree/master/pkg/controllers/elasticsearch/actions
ACTIONS
Transitoning cluster state with Actions

$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'

1. Observes change
2. Evaluates each ‘Pilot’ resource one at a time
   a. Is the node healthy?
   b. Is the node already at the desired version?
   c. Is the cluster healthy?
3. Inform the relevant Pilot it is to be upgrade

Elasticsearch upgrade action
https://github.com/jetstack/navigator/tree/master/pkg/controllers/elasticsearch/actions
ACTIONS
Transitioning cluster state with Actions

$ kubectl patch esc demo -p '{"spec":{"version":"6.1.3"}}'

1. Observes change
2. Evaluates each ‘Pilot’ resource one at a time
   a. Is the node healthy?
   b. Is the node already at the desired version?
   c. Is the cluster healthy?
3. Inform the relevant Pilot it is to be upgrade
4. Upgrade the node that needs to be upgraded

Elasticsearch upgrade action
https://github.com/jetstack/navigator/tree/master/pkg/controllers/elasticsearch/actions
Transitioning cluster state with Actions

Why do it this way?

- Controller can evaluate all actions to perform, and sequence them appropriately
- This allows one central ‘brain’ when making infrastructure changes
- Clearly defined and contained as a unit of work in code
- It can wait for ‘pre-conditions’ to be met e.g.
  - waiting for shards to be drained from an Elasticsearch node
  - waiting for a node to be decommissioned
ACTIONS

Transitioning cluster state with Actions

- Controller can evaluate all actions that need to be performed and sequence them safely
- Prevents accidental mistakes by administrators
- Upgrade, and scale once the cluster is in a healthy state.
THE FUTURE
What’s next for Navigator?

- Cutting a maintainable API - this will allow users to begin using Navigator for real
- Improving existing controller intelligence
- Supporting more database specific features (e.g. x-pack, rack awareness)
- Support ad-hoc administrator initiated Actions
- Automated OS and application patching through ‘managed versions’

- Custom ‘kubectl get’ output (from Kubernetes 1.10 onwards)
  - Makes custom resources ‘feel native’ in the system

```
$ kubectl get esc demo
NAMESPACE   NAME  HEALTH  MASTERS | DATA  | INGEST
red-team    demo  Green   3/3     4/4    4/4
blue-team   prod-cluster Yellow 3/3    3/4    3/4
```
Kubernetes provides us the building blocks to orchestrate and manage stateful systems.

Consistent deployment of stateless + stateful workloads across multiple environments means more efficiency and ability to deploy quicker without the complexities and overhead of centralised management.

Kubernetes is highly extensible: we can build on top of the API with custom resources and codify stateful operational logic into controllers.
CREDITS
To our other team members working on Navigator

Richard Wall
@wallrj

Louis Taylor
@kragniz
Thanks!

hello@jetstack.io
@JetstackHQ
github.com/jetstack/navigator
jetstack.io
KUBERNETES ALL THE THINGS
Stateless and stateful workloads in cluster co-existence

Cloud

Kubernetes API

nginx
mysql

Deployment
KUBERNETES ALL THE THINGS
Stateless and stateful workloads in cluster co-existence
KUBERNETES ALL THE THINGS

Stateless and stateful workloads in cluster co-existence - across cloud

Cloud

Kubernetes API

nginx

mysql
Kubernetes cluster

Managed Database
Pilots (sidecars)
Interacts with database
Application-aware
Leader-elected (if necessary)

Monitoring Engine
Prometheus

Database Controller
Navigator
Monitors Pilots
Manages resource lifecycle
Leader-elected

Elasticsearch Cluster
10GB PD
Over time we'll be able to codify that operational expertise into some universal control loop, but that's still a work in progress.
STATEFUL SERVICES

But there’s mixed option

Kubernetes has made huge improvements in the ability to run stateful workloads including databases and message queues, but I still prefer not to run them on Kubernetes.

https://twitter.com/kelseyhightower/status/963413508300812295
RESOURCE LIFECYCLE
From YAML to pods

$ kubectl apply -f deployment.yaml