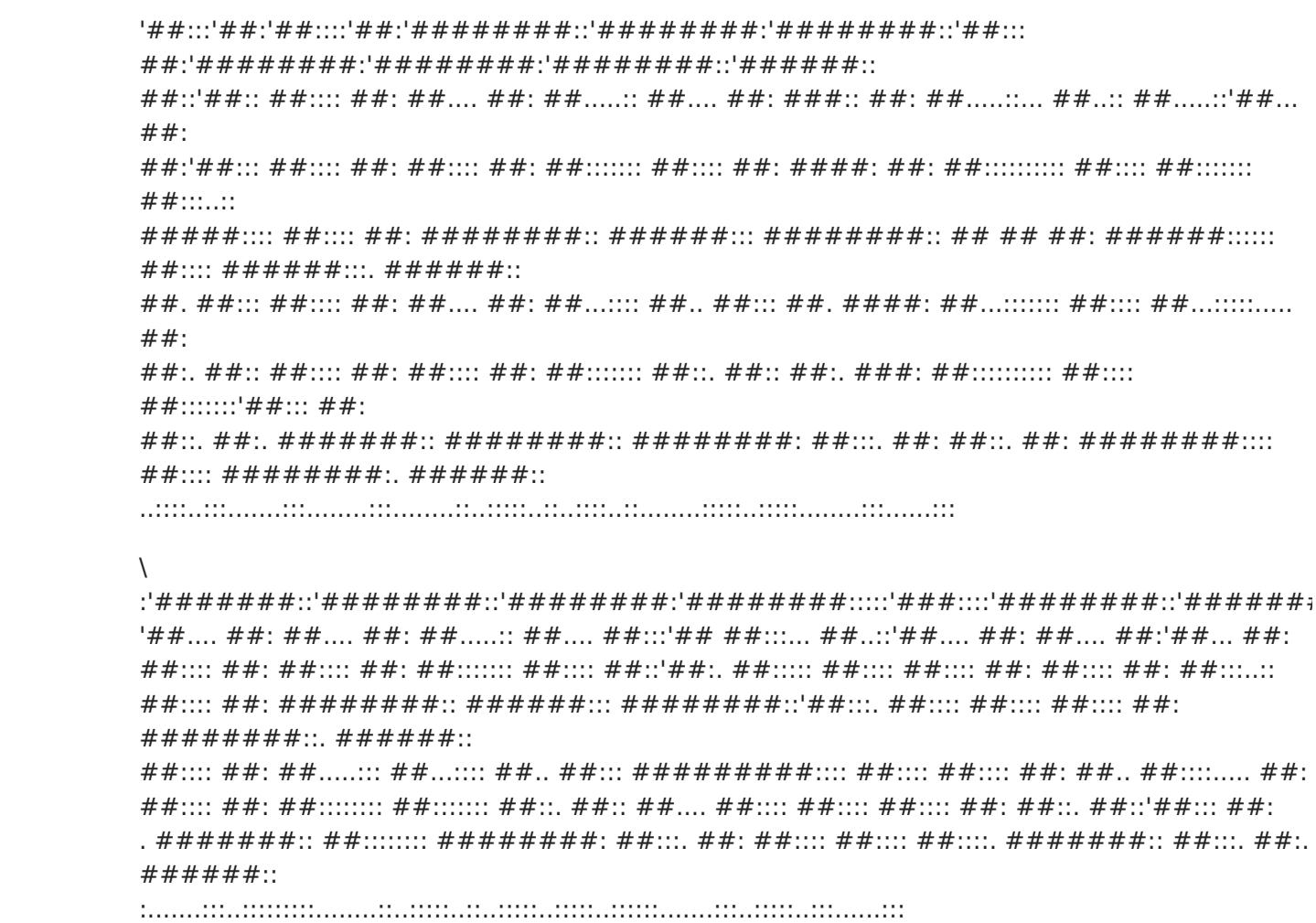
%title: [ContainerSched] London Septemeber 2017 - Operators: Automating Ourselves out of a Job with Kubernetes
%author: @lukeb0nd
%date: 2017-09-28



-> # [ContainerSched] London, September 2017
-> ## Luke Bond

WHO AM I?

- Developer turned DevOps engineer
- In recent years:
    - Mostly Node.js and Docker
    - Consulting, helping teams release more often with higher quality
    - Moving further down the stack over the years, now mostly Ops
- Co-Founder of *controlplane*, a London-based consultancy focusing on Kubernetes, security and continuous delivery
- Have been working with containers since 2014, when, like so many others, I built a Docker PaaS
- Hobbies include home-brewing and making headings with figlet

---

WHO IS THIS TALK FOR?

- Those wondering what operators are; what they're for and what they're not for
- Those who get the concept but unsure what building an operator entails
- Those interested in automation of operations on top of Kubernetes
- Those running stateful services in Kubernetes
- Those who want to know where to start working on operators
- This is an introductory talk - I'm not going to to into too much details on the coding side of things

---

WHAT ARE OPERATORS?

- Maybe you read the CoreOS Etcd Operator announcement blog post
- Maybe you watched some talks by Brandon Philips
- Maybe you listened to Brandon on the Cloudcast episode "Understanding Kubernetes Operators"

--> But maybe, like me, you were still left scratching your head a bit! <--

---

WHAT ARE OPERATORS?

There are some obvious things:

- Operators encapsulate operational knowledge in code
    - The kind of stuff a sysadmin knows about a service, but automated
- Operators leverage the Kubernetes API and primitives in order to do this

---

WHAT ARE OPERATORS?

But I was left with a few questions:

- Doesn't Kubernetes already magically look after my services and will restart and migrate them as necessary?
- Doesn't Kubernetes already have primitives such as StatefulSets and ReplicaSets to help with this stuff?
- How are these things actually built?

If I was confused about these things then maybe you are too. Hope this helps!

IN THIS TALK

- I aim to answer the questions of the previous slide
- I'll explain the relationship between Operators and Kubernetes primitives such as StatefulSets, ReplicaSets and Services
- I'll explain the scenarios where those primitives aren't enough- that's where Operators come in
- I'll give a tour of the tools and repos that will give you a starting point with operators
- Example use-cases of Operators

THE NICHE FOR OPERATORS - WHAT KUBERNETES DOES AND DOESN'T DO FOR YOU

- Let's say you have a 12-factor web app. Kubernetes will:
    - Keep it running; surviving crashes and node failures (ReplicaSets)
    - Scale it up and down when you want it to (ReplicaSets)
    - Internally load balance traffic to instances (Services)
- Stateless apps can be destroyed, moved and upgraded easily anytime
    - Existing Kubernetes primitives are perfect for this

THE NICHE FOR OPERATORS - WHAT KUBERNETES DOES AND DOESN'T DO FOR YOU

- Let's say you have a clustered database, however:
    - Can't be rescheduled on any host like stateless services
    - Instances need to stay with their data
    - Scaling may not be as simple as adding more nodes
    - Specialist knowledge is required to effectively manage and operate each database

---

THE NICHE FOR OPERATORS - WHAT KUBERNETES DOES AND DOESN'T DO FOR YOU

- This is where operators come in
- They fill the gap of the application-specific things that Kubernetes can't

do for you
- They extend and leverage existing Kubernetes primitives and functionality

> An Operator represents human operational knowledge in software,
> to reliably manage an application

- Complex, manual operational tasks become a single command or line of config

---

COREOS OPERATOR ANNOUNCEMENTS

> A Site Reliability Engineer (SRE) is a person that operates an application by writing software. They are an engineer, a developer, who knows how to develop software specifically for a particular application domain. The resulting piece of software has an application's operational domain knowledge programmed into it.

> We call this new class of software Operators. An Operator is an application-specific controller that extends the Kubernetes API to create, configure, and manage instances of complex stateful applications on behalf of a Kubernetes user. It builds upon the basic Kubernetes resource and controller concepts but includes domain or application-specific knowledge to automate common tasks.

-> -- Brandon Philips, "Introducing Operators", CoreOS blog November 3 2016

OPERATORS IN THE WILD

- The Etcd operator was the first
    - Released when the operator pattern was introduced/announced
- Prometheus Operator, from CoreOS
    - In beta
    - Automated deployment and management of Prometheus instances
- Rook - an orchestrator for cloud-native distributed storage systems
    - Installs as an operator, registering custom resources in Kubernetes
    - Create clusters via the operatoran orchestrator for cloud-native distributed storage systems
- Tectonic Operator, also from CoreOS
    - Everything in Tectonic is automated, from Container Linux to Etcd to Kubernetes

---

THE ETCD OPERATOR

The Etcd operator is a good place to start to see how they work

It has the following features:

- Create/Destroy
- Resize
- Backup
- Upgrade

It operates using the model: Observe, Analyse and Act

https://coreos.com/blog/introducing-the-etcd-operator.html#how-it-works (https://coreos.com/blog/introducing-the-etcd-operator.html#how-it-works)

THE ETCD OPERATOR

What is it doing under the hood?

- Registering a custom resource on startup: Etcd Cluster
    - Formerly TPR, now CRD
- Listens to Etcd for CRUD events on that API resource
- Acts on those events to affect the cluster
- Can be asked to perform certain operations, e.g. backup

CREATING OPERATORS

CoreOS have published some guidelines for creating operators:

https://coreos.com/blog/introducing-operators.html#how-can-you-create-an-operator (https://coreos.com/blog/introducing-operators.html#how-can-you-create-an-operator)

The Etcd codebase can be seen as a reference implementation of these guidelines.

https://github.com/coreos/etcd-operator (https://github.com/coreos/etcd-operator)

BUILDING OPERATORS

- This year, TPR became CRD. This blog posts explains the changes:

https://coreos.com/blog/custom-resource-kubernetes-v17 (https://coreos.com/blog/custom-resource-kubernetes-v17)

- Custom resources allow you to create your own resource types that you can manage and interact with in the same way that you can services, pods, secrets, etc. (i.e. with `kubectl`)

- Let's pretend we're creating a chaos-monkey operator. You've heard of `StatefulSets`, now we have `HatefulSets`! **groan**
- This is quite a contrived example, it isn't operating a stateful service, but I just want to show how they're created

---

BUILDING OPERATORS

- This is what the custom resource definition looks like:

```
$ cat hatefulset-crd.yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: hatefulsets.control-plane.io
spec:
  group: control-plane.io
  version: v1
  names:
    kind: HatefulSet
    plural: hatefulsets
  scope: Namespaced
```

BUILDING OPERATORS

- And here's what a HatefulSet resource might look like:

```
$ cat chaos-monkey.yaml
apiVersion: control-plane.io/v1
kind: HatefulSet
metadata:
  name: chaos-monkey
  namespace: default
spec:
  chaosLevel: 10
  interval: 300
```

BUILDING OPERATORS

- Here is how we register the custom resource:

```
$ kubectl create -f hatefulset-crd.yaml
customresourcedefinition "hatefulsets.control-plane.io" created
$ kubectl get customresourcedefinitions
NAME                          KIND
hatefulsets.control-plane.io
CustomResourceDefinition.v1beta1.apiextensions.k8s.io
```

- ...and create an initial instance of it:

```
$ kubectl create -f chaos-monkey.yaml
hatefulset "chaos-monkey" created
$ kubectl get hatefulsets
NAME           KIND
chaos-monkey   HatefulSet.v1.control-plane.io
```

BUILDING OPERATORS

- You can read more about custom reources and controllers here:

https://kubernetes.io/docs/concepts/api-extension/custom-resources/
(https://kubernetes.io/docs/concepts/api-extension/custom-resources/)

BUILDING OPERATORS

- We've just registered a new resource type and created an instance of it
- That second step generated a CREATED event for resource type HatefulSet
- Next we need to write code to watch Etcd to hear of these events
    - And also delete, and update
- At this point we have the basis of an Operator's *Observe, Analyse, Act* cycle

-> But this is a lot of boilerplate! <-
-> To define data model and watch Etcd <-

BUILDING OPERATORS

- Starting from this example code will give you a head start:

https://github.com/kubernetes/apiextensions-apiserver/tree/master/examples/client-go
(https://github.com/kubernetes/apiextensions-apiserver/tree/master/examples/client-go)

This example shows:

- How to register a new custom resource (custom resource type) using a CustomResourceDefinition
- How to create/get/list instances of your new resource type (update/delete/etc work as well but are not demonstrated)
- How to setup a controller on resource handling create/update/delete events

---

BUILDING OPERATORS

There are code generators to help you here.

https://github.com/kubernetes/gengo (https://github.com/kubernetes/gengo)

Documentation is scant. Until that's improved you're on your own figuring it all out.

See James Munnelly's excellent talk on the subject here (https://skillsmatter.com/skillscasts/10599-wrangling-kubernetes-api-internals#about) for more details.

EXAMPLE USE CASES OF OPERATORS

- Anything with application-specific operational/maintenance tasks
- Databases are the obvious choice
    - Postgres
    - Redis
    - Mongo
    - etc.
- Also "legacy" or non cloud-native applications
    - That old stateful Java enterprise monolith on which your business still depends
- Apps that don't like to be moved without some manual intervention

---

KUBERNETES-NATIVE APPLICATIONS

There is another class of applications for which we don't yet have a name, that extend the Kubernetes API (and therefore declare custom resources that can be managed with `kubectl`), yet don't specifically operate stateful services.

I'm calling these "Kubernetes-native applications", and have a few advantages over just running as an application on Kubernetes like any other.

- Can be discovered via the Kubernetes API
- Can declare custom resources that can be managed via `kubectl`
- Can leverage Kubernetes' RBAC for access to their API

This last item alone is probably enough to make them worthwhile as an ops tool

---

KUBERNETES-NATIVE APPLICATIONS

What kind of Kubernetes-native applications could be useful?

- Chaos-monkey operator with an API to trigger and configure
- System acceptance tests that run inside the cluster, spin up different pods at different versions and test contracts between them

    - The k8s-native equivalent of the trusty Bash/Fig combo!

- Security compliance pod that will try to do things it shouldn't be able to do inside a cluster, for use in CI
- An operator to steward releases with gradual roll-out

    - Something like https://github.com/controlplaneio/theseus (https://github.com/controlplaneio/theseus)
    - (But as an operator)

- Many other things, including a few things that we're working on at *controlplane*

AUTOMATING OURSELVES OUT OF A JOB

- Anything you can do with kubectl you can do with Operators
- With auth, from inside the cluster
- This is what I'm advocating: we now have the best place to run our tools
- In code, rather than with Bash calling kubectl

    - Not that there is anything wrong with that

A FUTURE WITH OPERATORS

- Operators are a step towards fully automated infrastructure
- Self-operating and self-healing systems and infrastructure already exist
- IaaS, Docker and Kubernetes enable a revolution in this space

    - The building blocks are now in place to make this possible

- The emergence of the Operator pattern is an early attempt at a standard way to build self-managing systems on top of Kubernetes
- People are still keeping databases out of Kubernetes

    - I think we're running out of excuses to do this

-> # Thanks!!

-> ## Any questions?

-> Control Plane (https://control-plane.io)
-> @controlplaneio (https://twitter.com/controlplaneio)

-> Slides can be found here: https://github.com/lukebond/containersched-london-operators-20170928 (https://github.com/lukebond/containersched-london-operators-20170928)