Distributed systems for big data processing on Kubernetes

Concepts, architecture and implementation

Alena Hall - @lenadroid

dockercon SF 18

Agenda

- Distributed systems for big data
- Running them on Kubernetes?
- Getting them to “work”
- Challenges
- Work in progress and solutions
- Future
- Summary

Distributed systems for big data:

storage, processing, analysis

@lenadroid

dockercon SF 18


cassandra


kafka


Spark


elastic


Apache
HERON

Data systems on Kubernetes

@lenadroid

dockercon SF 18

How do we run systems like Kafka, Cassandra, Spark, etc. on Kubernetes?

@lenadroid

dockercon SF 18

Interpretation #1

What Kubernetes abstractions can we use to run these distributed systems?

Interpretation #2

What are the steps to simply get those distributed systems up and running based on these Kubernetes abstractions?

@lenadroid

Interpretation #3

How to get them to run and work according to what is considered correct behavior for those systems?

Answer #1:

Kubernetes abstractions

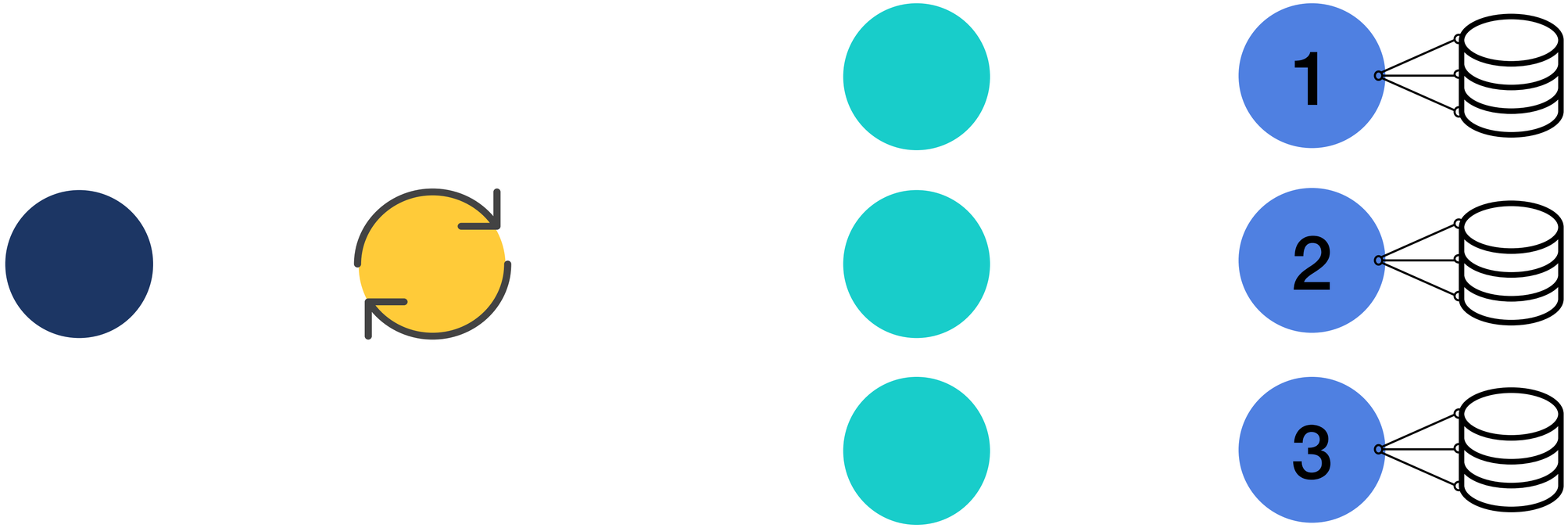
@lenadroid

Pod

Job

Replica Set

Stateful Set



And many more…

Answer #2:

Getting it up and running using built-in K8S abstractions

@lenadroid

Yaml files

Example: Stateful Sets + Cassandra

@lenadroid

cassandra-statefulset.yaml

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 5
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      terminationGracePeriodSeconds:
      1800
      containers:
      - name: cassandra
        image:
        gcr.io/google-samples/cassand
        ra:v13
        imagePullPolicy: Always
        ports:
        - containerPort: 7000
          name: intra-node
        - containerPort: 7001
          name: tls-intra-node
        - containerPort: 7199
          name: jmx
        - containerPort: 9042
          name: cql
        resources:
```

cassandra-statefulset.yaml

```yaml
        resources:
          limits:
            cpu: "2"
            memory: 3.5Gi
          requests:
            cpu: "2"
            memory: 3.5Gi
        securityContext:
          capabilities:
            add:
              - IPC_LOCK
        lifecycle:
          preStop:
            exec:
              command:
              - /bin/sh
              - -c
              - nodetool drain
        env:
          - name: MAX_HEAP_SIZE
            value: 512M
          - name: HEAP_NEWSIZE
            value: 100M
          - name: CASSANDRA_SEEDS
            value:
            "cassandra-0.cassandra.
            default.svc.cluster.
            local"
          - name:
          CASSANDRA_CLUSTER_NAME
            value: "chicago"
          - name: CASSANDRA_DC
            value: "DC1-chicago"
          - name: CASSANDRA_RACK
            value: "Rack1-chicago"
```

cassandra-statefulset.yaml

```yaml
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath:
                status.podIP
        readinessProbe:
          exec:
            command:
            - /bin/bash
            - -c
            - /ready-probe.sh
          initialDelaySeconds: 15
          timeoutSeconds: 5
        volumeMounts:
        - name: cassandra-data
          mountPath: /cassandra_data
  volumeClaimTemplates:
  - metadata:
      name: cassandra-data
    spec:
      accessModes: [
      "ReadWriteOnce" ]
      storageClassName:
      managed-premium
      resources:
        requests:
          storage: 1Gi
```

```
repeat while (true) {
```

UP AND RUNNING

!=

OPERATING CORRECTLY

```
}
```

@lenadroid

HELM

Get Helm Blog Docs

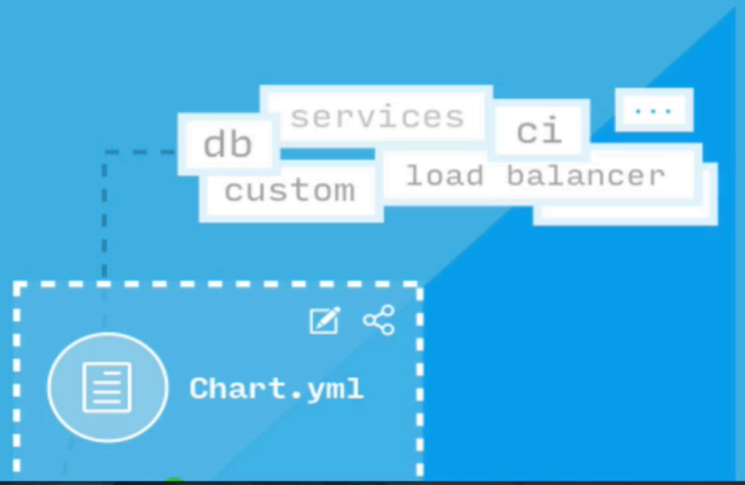
The package manager for Kubernetes

Helm is the best way to find, share, and use software built for Kubernetes.

What is Helm?

Helm helps you manage Kubernetes applications — Helm Charts helps you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste madness.



Helm

- Charts and Chart.yaml
- Values and Values.yaml
- Chart repositories
- Tiller, Releases and Rollbacks

Other examples of deployment

Example: Spark on Kubernetes

@lenadroid

Is this really enough?

Answer #3:

Making sure our systems operate according to what is considered correct behavior for those systems

@lenadroid

Things that need special care

Kubernetes is smart.

P.S. When we tell it how to treat our systems.

Kafka

Configuration

Rolling cluster restarts and upgrades

Scaling cluster up and down

Data rebalancing

Kafka configuration

Pods read configuration from ConfigMaps

Pod specific vs general configurations

Broker ID and rack assignments

Kafka cluster restarts/upgrades

No under-replicated partitions

Restart one broker at a time

Always wait for restarted broker to catch up to a leader

Scaling Kafka cluster

Partition assignment

Data rebalancing

Managing ConfigMaps


OPERATORS
@lenadroid

Operators

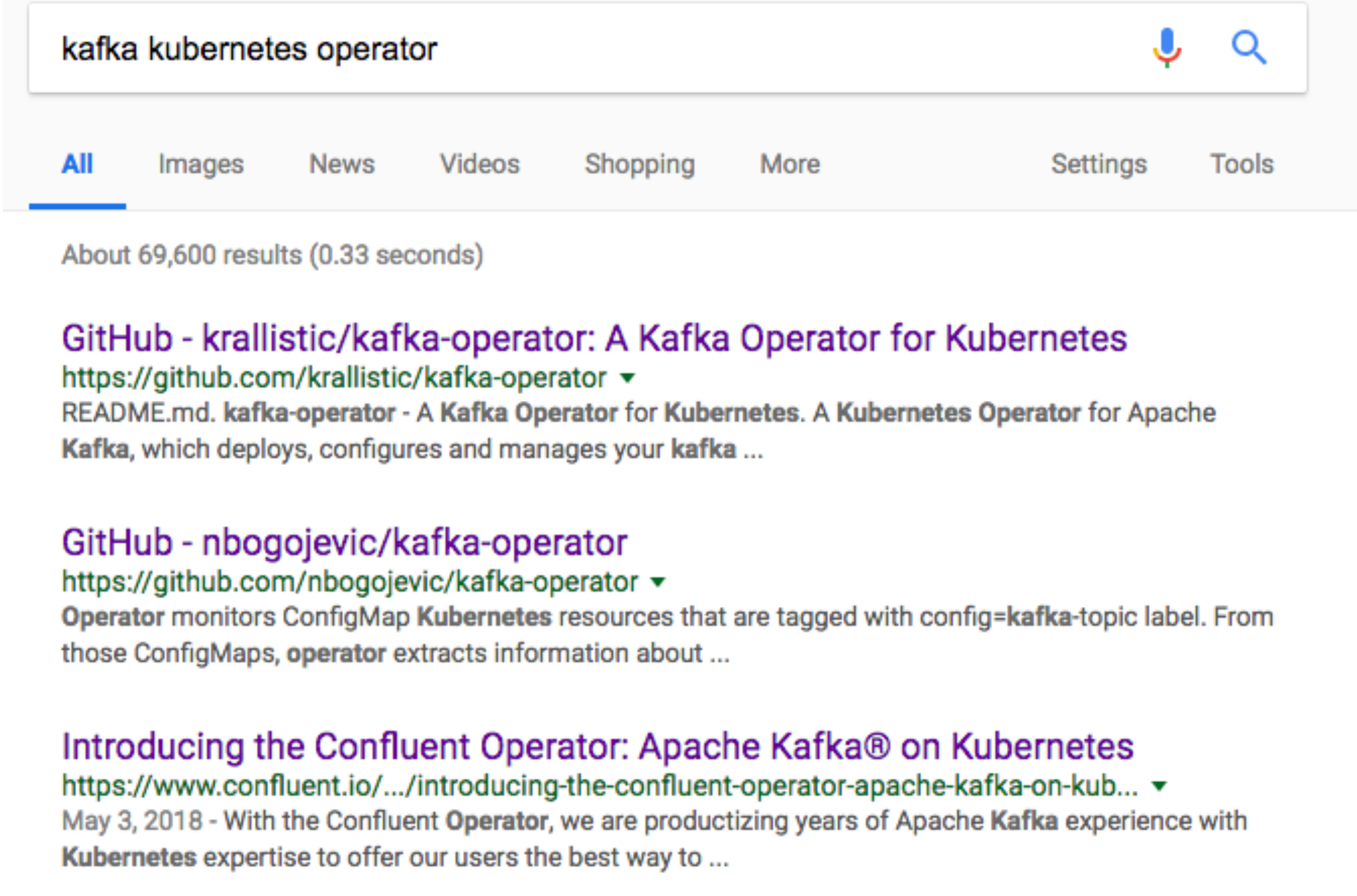
Custom Resource Definitions

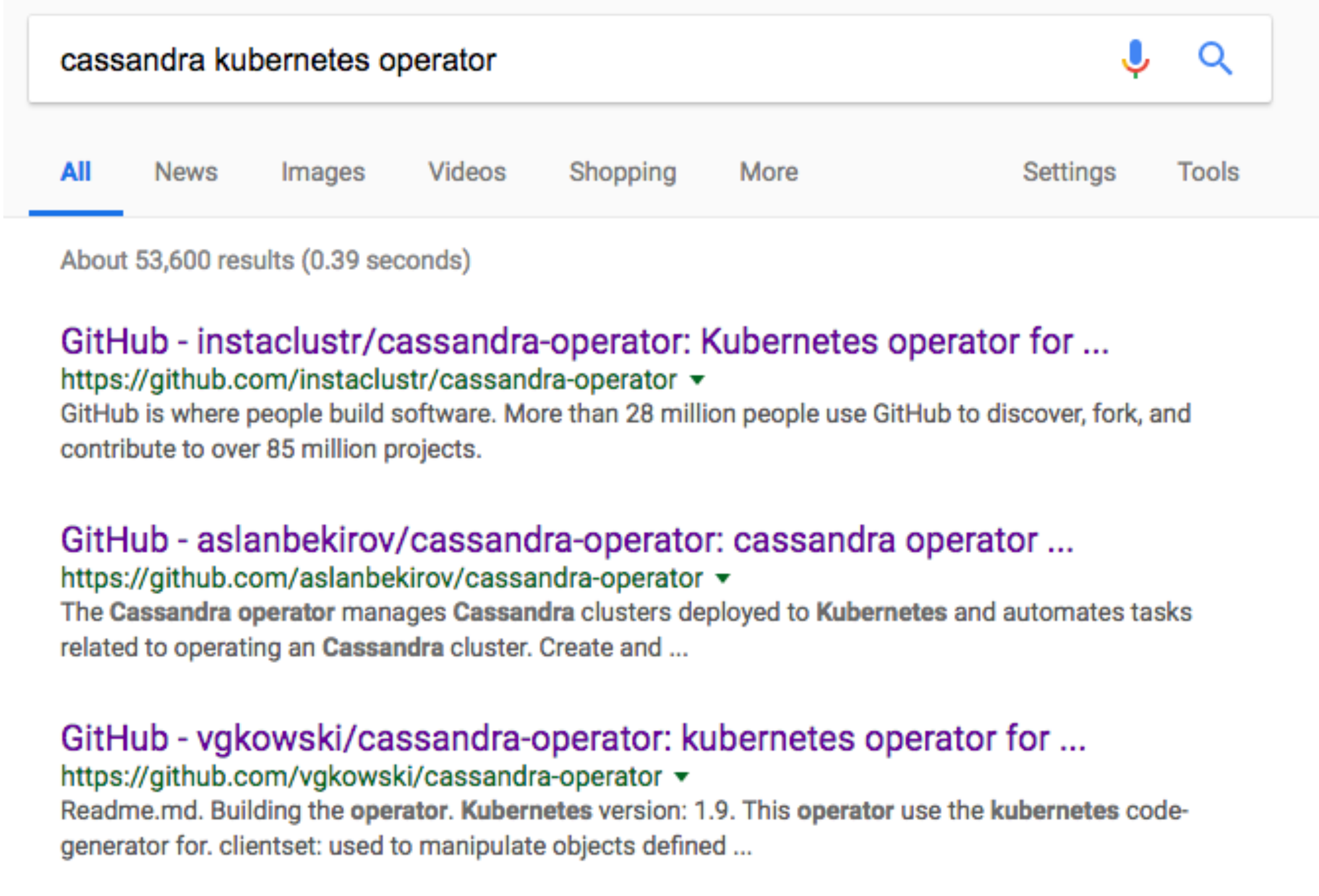
+

Custom Controllers

=

More Control


kafka kubernetes operator
All
Images
News
Videos
Shopping
More
Settings
Tools
About 69,600 results (0.33 seconds)
GitHub - krallistic/kafka-operator: A Kafka Operator for Kubernetes
https://github.com/krallistic/kafka-operator
README.md. kafka-operator - A Kafka Operator for Kubernetes. A Kubernetes Operator for Apache Kafka, which deploys, configures and manages your kafka ...
GitHub - nbogojevic/kafka-operator
https://github.com/nbogojevic/kafka-operator
Operator monitors ConfigMap Kubernetes resources that are tagged with config=kafka-topic label. From those ConfigMaps, operator extracts information about ...
Introducing the Confluent Operator: Apache Kafka® on Kubernetes
https://www.confluent.io/.../introducing-the-confluent-operator-apache-kafka-on-kub...
May 3, 2018 - With the Confluent Operator, we are productizing years of Apache Kafka experience with Kubernetes expertise to offer our users the best way to ...


cassandra kubernetes operator
All
News
Images
Videos
Shopping
More
Settings
Tools
About 53,600 results (0.39 seconds)
GitHub - instaclustr/cassandra-operator: Kubernetes operator for ...
https://github.com/instaclustr/cassandra-operator
GitHub is where people build software. More than 28 million people use GitHub to discover, fork, and contribute to over 85 million projects.
GitHub - aslanbekirov/cassandra-operator: cassandra operator ...
https://github.com/aslanbekirov/cassandra-operator
The Cassandra operator manages Cassandra clusters deployed to Kubernetes and automates tasks related to operating an Cassandra cluster. Create and ...
GitHub - vgkowski/cassandra-operator: kubernetes operator for ...
https://github.com/vgkowski/cassandra-operator
Readme.md. Building the operator. Kubernetes version: 1.9. This operator use the kubernetes code-generator for. clientset: used to manipulate objects defined ...

Spark (>=2.3)

Automatic job resubmission after specification update

Configurable restart policy

Automatic retries on failed submissions

Mounting specific ConfigMaps and volumes

Example: Spark Operator on Kubernetes

Anatomy of an Operator

CRD

Resource Structs/Types

Informer

Controller

Queue

TensorFlow Operator

CRD

```yaml
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: tfjobs.kubeflow.org
spec:
  group: kubeflow.org
  version: v1alpha1
  names:
    kind: TFJob
    singular: tfjob
    plural: tfjobs
```

TFJOB

```
// TFJob represents the configuration of signal TFJob
type TFJob struct {
        metav1.TypeMeta `json:",inline"`

        // Standard object's metadata.
        metav1.ObjectMeta `json:"metadata,omitempty"`

        // Specification of the desired behavior of the TFJob.
        Spec TFJobSpec `json:"spec,omitempty"`

        // Most recently observed status of the TFJob.
        // This data may not be up to date.
        // Populated by the system.
        // Read-only.
        Status TFJobStatus `json:"status,omitempty"`
}
```

```go
// TFJobSpec is a desired state description of the TFJob.
type TFJobSpec struct {
        // TFReplicaSpecs is map of TFReplicaType and TFReplicaSpec
        // specifies the TF replicas to run.
        // For example,
        //   {
        //     "PS": TFReplicaSpec,
        //     "Worker": TFReplicaSpec,
        //   }
        TFReplicaSpecs map[TFReplicaType]*TFReplicaSpec `json:"tfReplicaSpecs"`
}
```

```go
// TFReplicaSpec is a description of the TFReplica
type TFReplicaSpec struct {
        // Replicas is the desired number of replicas of the given template.
        // If unspecified, defaults to 1.
        Replicas *int32 `json:"replicas,omitempty"`

        // Template is the object that describes the pod that
        // will be created for this TFReplica.
        // We use RestartPolicy in PodTemplateSpec
        // to describe how the containers within the pod should be restarted.
        // Please set this restart policy carefully according to your code.
        Template v1.PodTemplateSpec `json:"template,omitempty"`

        // Restart policy for all TFReplicas within the TFJob.
        // One of Always, OnFailure, Never and ExitCode.
        // Default to Always.
        RestartPolicy RestartPolicy `json:"restartPolicy,omitempty"`
}
```

```
// TFReplicaType is the type for TFReplica.
type TFReplicaType string

const (
        // TFReplicaTypePS is the type for parameter servers of distributed TensorFlow.
        TFReplicaTypePS TFReplicaType = "PS"

        // TFReplicaTypeWorker is the type for workers of distributed TensorFlow.
        // This is also used for non-distributed TensorFlow.
        TFReplicaTypeWorker TFReplicaType = "Worker"

        // TFReplicaTypeChief is the type for chief worker of distributed TensorFlow.
        // If there is "chief" replica type, it's the "chief worker".
        // Else, worker:0 is the chief worker.
        TFReplicaTypeChief TFReplicaType = "Chief"

        // TFReplicaTypeEval is the type for evaluation replica in TensorFlow.
        TFReplicaTypeEval TFReplicaType = "Eval"
)
```

```go
// RestartPolicy describes how the TFReplicas should be restarted.
// Only one of the following restart policies may be specified.
// If none of the following policies is specified, the default one
// is RestartPolicyAlways.
type RestartPolicy string

const (
        RestartPolicyAlways    RestartPolicy = "Always"
        RestartPolicyOnFailure RestartPolicy = "OnFailure"
        RestartPolicyNever     RestartPolicy = "Never"

        // `ExitCode` policy means that user should add exit code by themselves,
        // `tf-operator` will check these exit codes to
        // determine the behavior when an error occurs:
        // - 1-127: permanent error, do not restart.
        // - 128-255: retryable error, will restart the pod.
        RestartPolicyExitCode RestartPolicy = "ExitCode"
)
```

```go
// TFJobStatus represents the current observed state of the TFJob.
type TFJobStatus struct {
	// Conditions is an array of current observed TFJob conditions.
	Conditions []TFJobCondition `json:"conditions"`

	// TFReplicaStatuses is map of TFReplicaType and TFReplicaStatus,
	// specifies the status of each TFReplica.
	TFReplicaStatuses map[TFReplicaType]*TFReplicaStatus `json:"tfReplicaStatuses"`

	// Represents time when the TFJob was acknowledged by the TFJob controller.
	// It is not guaranteed to be set in happens-before order across separate operations.
	// It is represented in RFC3339 form and is in UTC.
	StartTime *metav1.Time `json:"startTime,omitempty"`

	// Represents time when the TFJob was completed. It is not guaranteed to
	// be set in happens-before order across separate operations.
	// It is represented in RFC3339 form and is in UTC.
	CompletionTime *metav1.Time `json:"completionTime,omitempty"`

	// Represents last time when the TFJob was reconciled. It is not guaranteed to
	// be set in happens-before order across separate operations.
	// It is represented in RFC3339 form and is in UTC.
	LastReconcileTime *metav1.Time `json:"lastReconcileTime,omitempty"`
}
```

```go
type TFJobInformer interface {
	Informer() cache.SharedIndexInformer
	Lister() v1alpha2.TFJobLister
}

type tFJobInformer struct {
	factory internalinterfaces.SharedInformerFactory
}

// NewTFJobInformer constructs a new informer for TFJob type.
// Always prefer using an informer factory to get a shared informer instead of getting an independent
// one. This reduces memory footprint and number of connections to the server.
func NewTFJobInformer(client versioned.Interface, namespace string, resyncPeriod time.Duration, indexers cache.Indexers) cache.SharedIndexI
	return cache.NewSharedIndexInformer(
		&cache.ListWatch{
			ListFunc: func(options v1.ListOptions) (runtime.Object, error) {
				return client.KubeflowV1alpha2().TFJobs(namespace).List(options)
			},
			WatchFunc: func(options v1.ListOptions) (watch.Interface, error) {
				return client.KubeflowV1alpha2().TFJobs(namespace).Watch(options)
			},
		},
		&tensorflow_v1alpha2.TFJob{},
		resyncPeriod,
		indexers,
	)
}
```

```
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

// Package controller provides a Kubernetes controller for a TensorFlow job resource.
package controller

import (
        "errors"
        "fmt"
        "time"

        log "github.com/sirupsen/logrus"
        "k8s.io/api/core/v1"
        apierrors "k8s.io/apimachinery/pkg/api/errors"
        metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
        "k8s.io/apimachinery/pkg/util/runtime"
        utilruntime "k8s.io/apimachinery/pkg/util/runtime"
        "k8s.io/apimachinery/pkg/util/wait"
        "k8s.io/client-go/kubernetes"
        "k8s.io/client-go/kubernetes/scheme"
        typedcorev1 "k8s.io/client-go/kubernetes/typed/core/v1"
        "k8s.io/client-go/tools/cache"
        "k8s.io/client-go/tools/record"
        "k8s.io/client-go/util/workqueue"

        "github.com/juju/ratelimit"
        tfv1alpha1 "github.com/kubeflow/tf-operator/pkg/apis/tensorflow/v1alpha1"
        tfjobclient "github.com/kubeflow/tf-operator/pkg/client/clientset/versioned"
```

Takeaways

Built-in Kubernetes abstractions don’t solve all the issues

Tools like Helm really help structure and manage deployments

Operators = Custom Controllers + CRDs & …

…. a way to teach Kubernetes understand our needs

Natallia Dzenisenka

twitter.com/nata_dzen

co-presenter

Thank You!

My blog - **lenadroid.github.io**

My talk on Stateful Sets, Persistent Volumes, Persistent Volime Claims, Storage Classes - **aka.ms/gotochgo**

Thomas Stringer's post on Custom Controllers – **aka.ms/custom-controllers**

Alena Hall - lenadroid

- ✓ Works on Azure at Microsoft
- ✓ Lives in Seattle
- ✓ F# Software Foundation Board of Trustees
- ✓ Organizes @ML4ALL 
- ✓ Program Committee for Lambda World
- ✓ Has a channel: YouTube /c/AlenaHall