Managing Flink on Kubernetes

Anand Swaminathan (*@anand12100*)
Ketan Umare (*@ketanumare*)

April 2, 2019

Agenda

1 **Kubernetes Primer**
Quick Introduction to concepts in Kubernetes

2 **Background**
Summary of Lyft’s legacy Flink Deployment

3 **Solution**
Flink Kubernetes Operator

4 **Demo**

5 **Ecosystem**

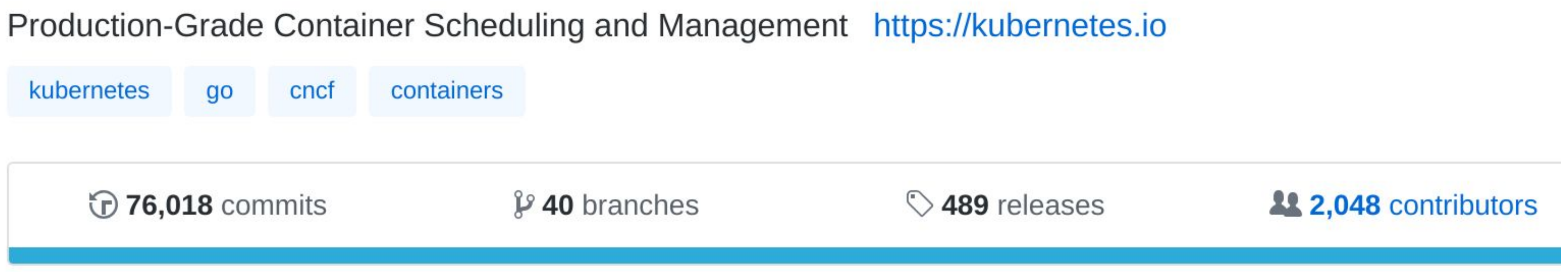
6 **Roadmap**

About us

Kubernetes Primer

History

- **Google's** internal infrastructure is containerized and runs on **Borg/Omega**
- K8s was **open sourced** in **2014**, re-incarnation of the internal infrastructure
- Kubernetes **automates** - **deployment**, **scaling** and **management** of containerized apps.
- Containers are scheduled based on **CPU**/**GPU**/**Memory**/Disk etc

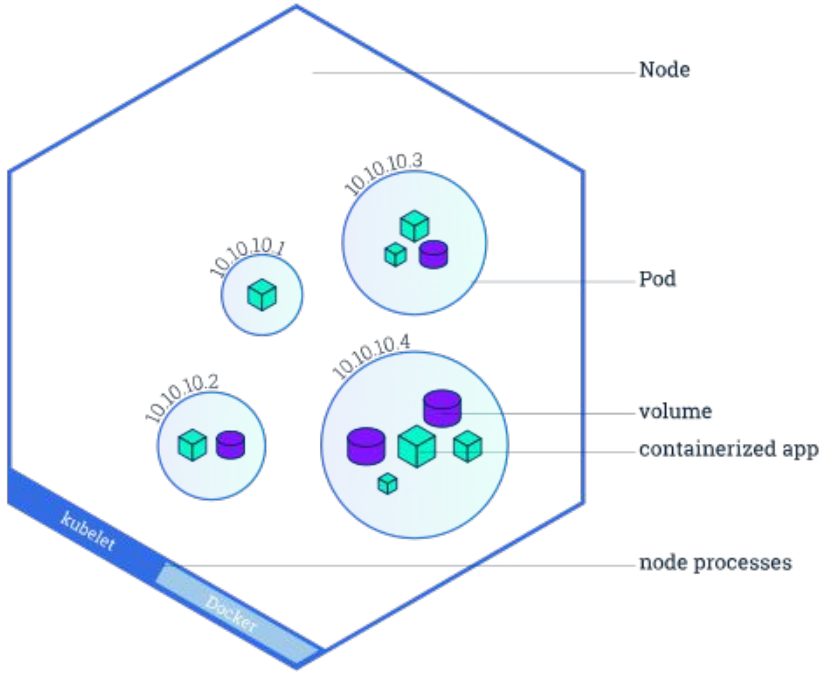

Production-Grade Container Scheduling and Management https://kubernetes.io

kubernetes go cncf containers

76,018 commits **40** branches **489** releases **2,048** contributors

Kubernetes Primer

Pods

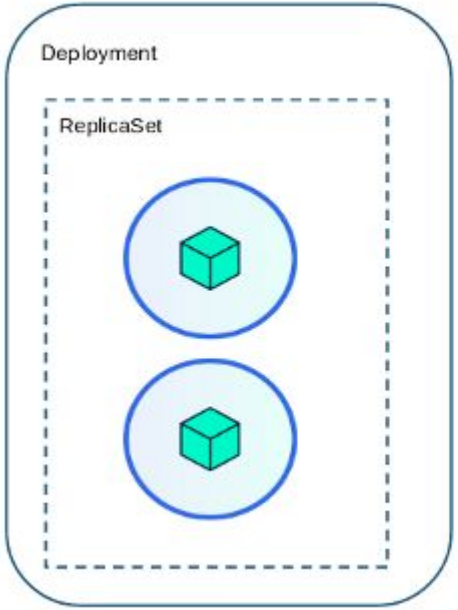


- A Pod is a group of **one or more Containers** as one unit
- Pods have **no durability** guarantees
- Each Pod has a **unique IP Address**
- **Containers** in a Pod can communicate using **localhost**
- **Multiple Pods** can be located on the **same node** - machine

Kubernetes Primer

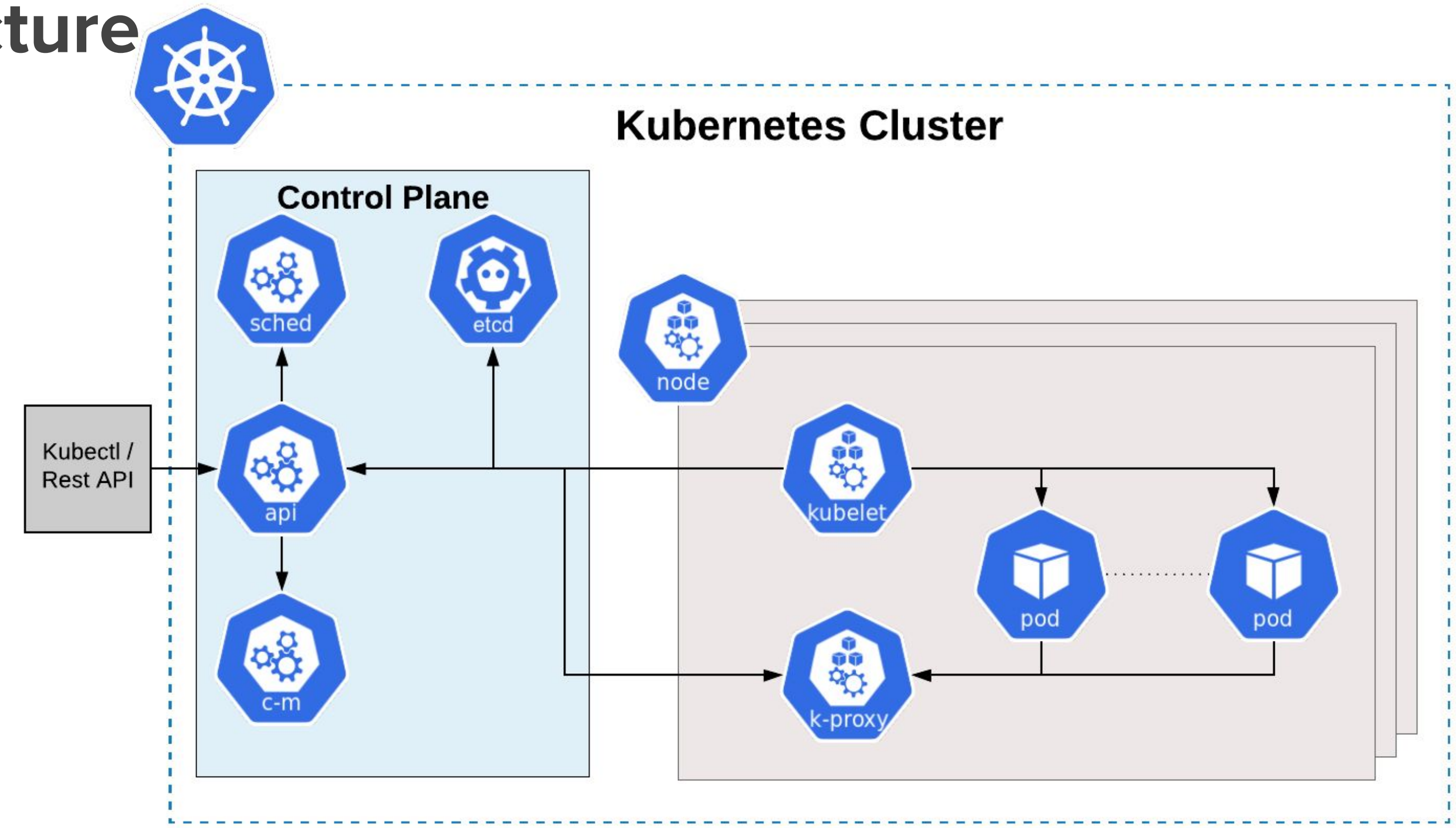
Other Concepts

- **Deployments** abstraction that enables rolling out changes to a set of pods
- **Service** abstraction to access sets of pods - like a load balancer within a k8s cluster
- **Ingress** abstraction to expose a service to the outside world (HTTP/HTTPS)
- **Controller** A reconciliation loop that drives current state towards desired state



Kubernetes Primer

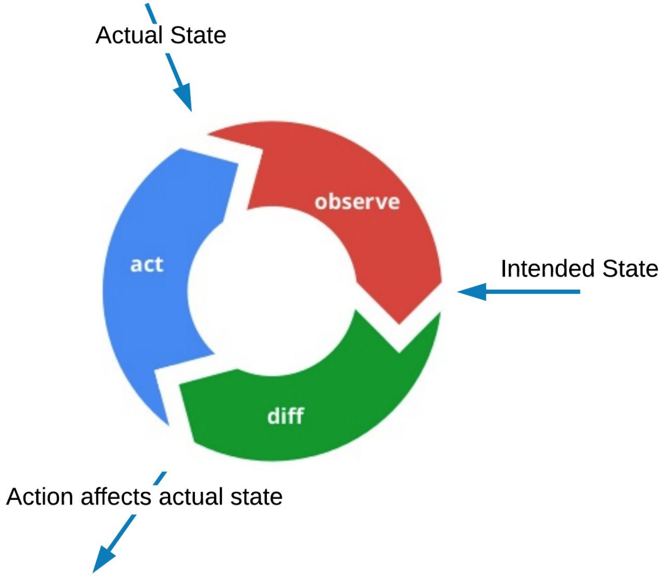
Architecture



Kubernetes Primer

Control Loops

- **Control loops** are fundamental building block industrial control systems
- **Desired State** refers to the intended state as **requested**
- **Current/Observed State** is the state of the system as **observed** by the controller
- Controller runs control loops
- Drive **Current State -> Desired State**
- This is the cornerstone of Kubernetes



Kubernetes Primer

Custom Resources

- Custom Resource Definitions (**CRD**) allow **extending** Kubernetes API
- Custom resources are **optional** extensions
- Custom resources can be **added**/removed **dynamically**
- They can be manipulated using known tools - **kubectl** & kube clients
- State stored in **etcd**
- Custom control loops (**controllers**) are used to manage the state of the resource.
- CRD is essentially the **desired state.**

Operators

- **Controller + CRD = Kubernetes Operator**
- Term coined by **CoreOS - 2017**
- **Manages** a **complex applications** lifecycle on Kubernetes.
- Core library to author operators @ SIG/controller-runtime

Background

OK how does this relate

- @Lyft we started working on **Flyte** - a modern take at Pipelines/Workflows
- Orchestration is pervasive **throughout various sectors** of our Industry
    - Machine learning
    - Data engineering and processing
    - ETL
- Kubernetes has a solution to many of our problems
    - Deployment, Versioning, cluster management etc
- In parallel **Streaming Platform** started working on Flink for streaming applications

Background

Legacy deployment of Flink @Lyft

- Hosted on **AWS**
- Separate **AutoScalingGroups** for Task Managers and Job Managers
- Machines provisioned and bootstrapped by **SaltStack**
- Every deployment needs **provisioning of machines**
- Users started running multiple jobs in the same Flink Cluster
- **Multi-tenancy hell !**

Introducing Flink-k8s-operator

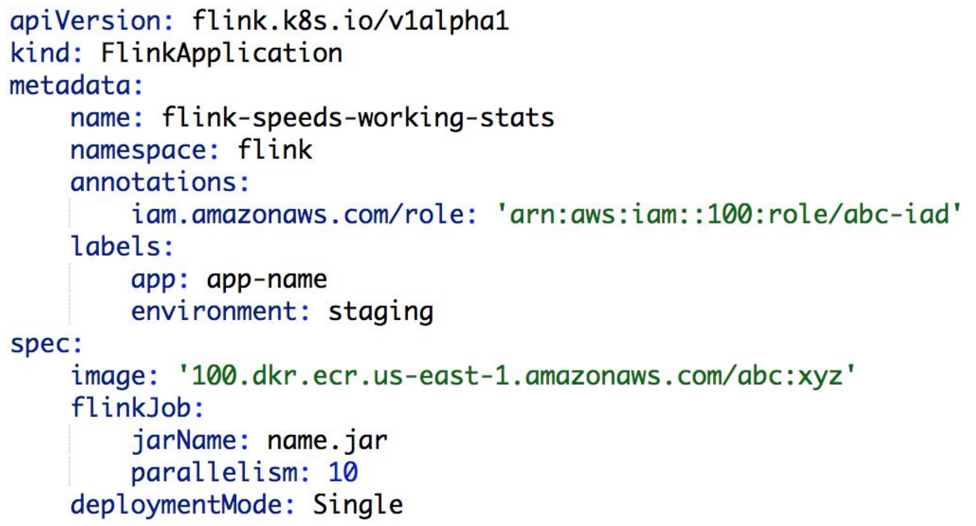
Goals

- Abstract out the complexity from application developers
    - **Hosting**
    - **Configuration**
    - **Management**
- Separate Flink cluster for each Flink application.
- **Deploy and rollback** support
- Support Flink application updates - **scaling**
- Simplified interface for instituting best practices
- **Scale to 100s** of flink applications

Solution

Flink Operator - CRD

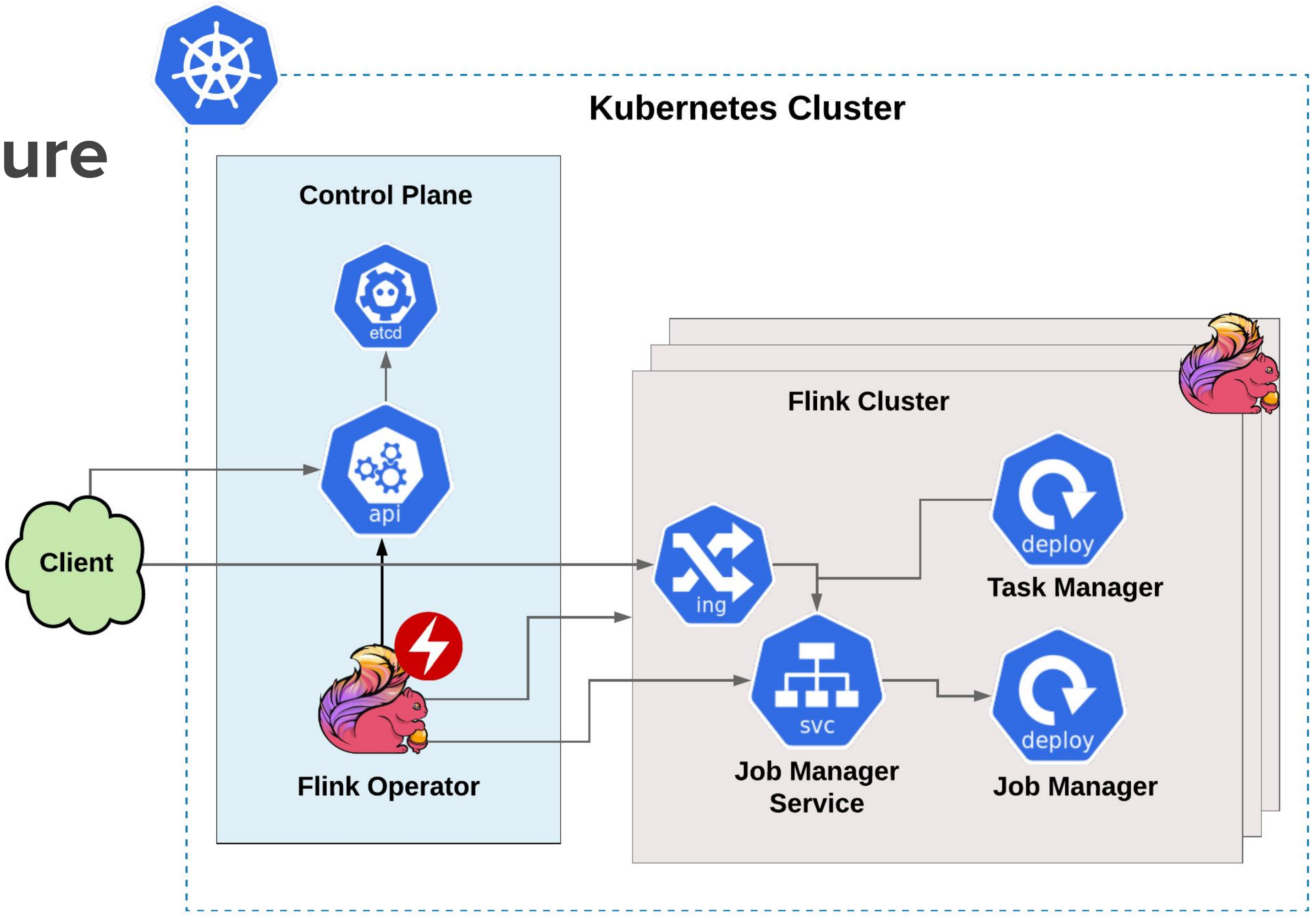
- Each custom resource corresponds to a Flink application
- Each Flink application runs a single **Flink job**
- **Docker image** should be runnable



```yaml
apiVersion: flink.k8s.io/v1alpha1
kind: FlinkApplication
metadata:
    name: flink-speeds-working-stats
    namespace: flink
    annotations:
        iam.amazonaws.com/role: 'arn:aws:iam::100:role/abc-iad'
    labels:
        app: app-name
        environment: staging
spec:
    image: '100.dkr.ecr.us-east-1.amazonaws.com/abc:xyz'
    flinkJob:
        jarName: name.jar
        parallelism: 10
    deploymentMode: Single
```

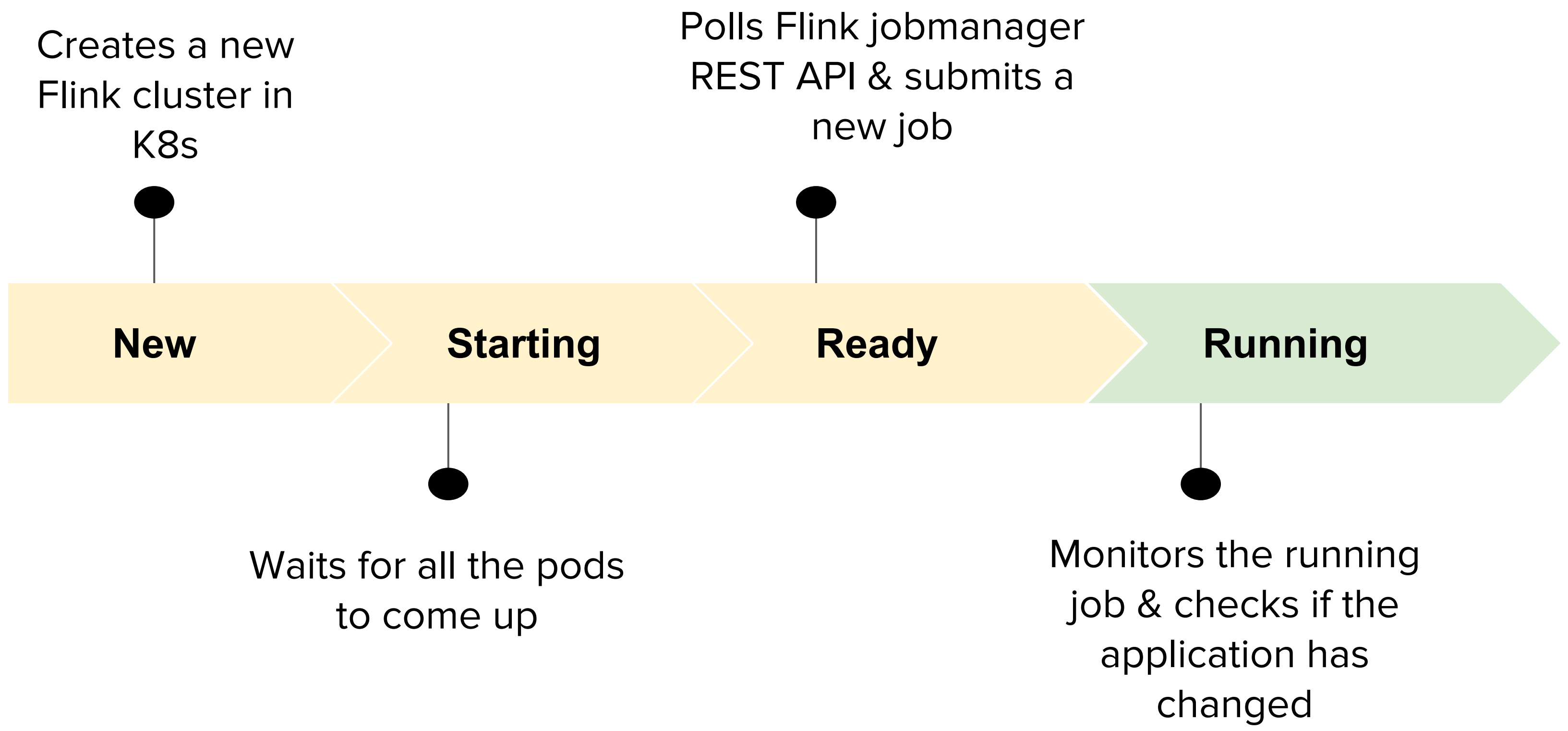
Solution

Architecture



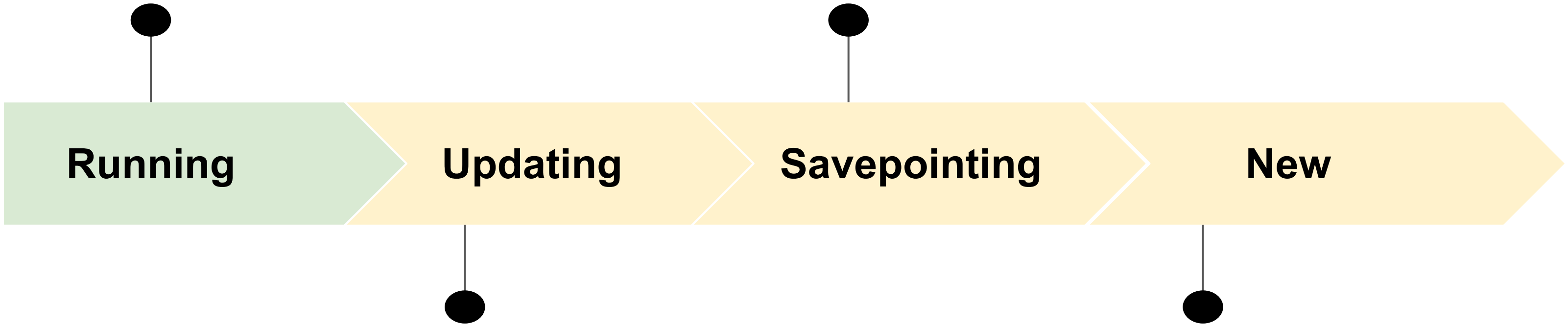
Solution

Operator Walkthrough


Creates a new Flink cluster in K8s
Polls Flink jobmanager REST API & submits a new job
New
Starting
Ready
Running
Waits for all the pods to come up
Monitors the running job & checks if the application has changed

Solution

Operator Walkthrough



Operator detects the update to CRD

If needed, updates cluster, cancels Job with savepoint

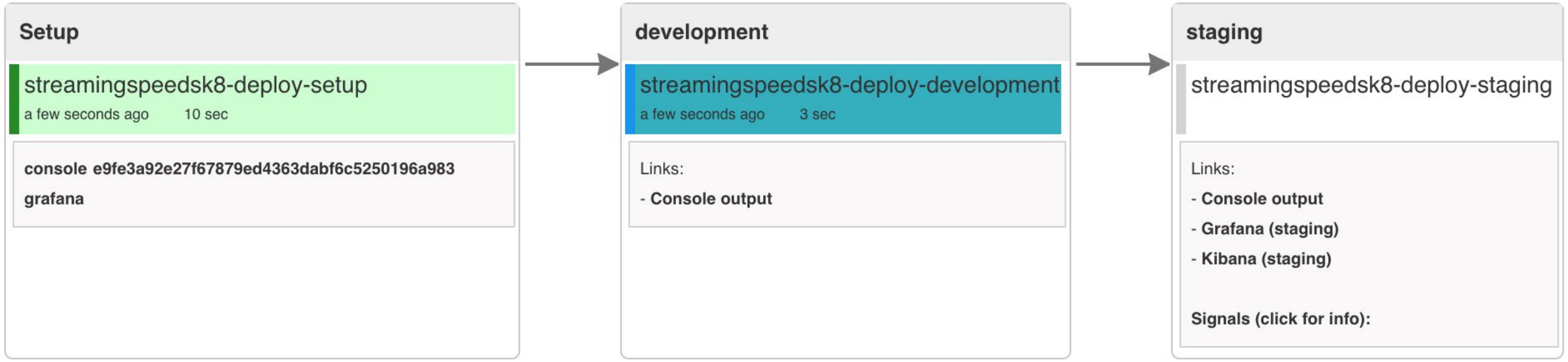
Waits for the savepoint to succeed, and updates savepoint location in CRD

Brings up a new cluster and tries to transition to ***Running***

Demo

Deployment @Lyft

- Jenkins based deployment
- Each stage creates or updates the resource in Kubernetes



Setup

streamingspeedsk8-deploy-setup
a few seconds ago 10 sec

console e9fe3a92e27f67879ed4363dabf6c5250196a983
grafana

development

streamingspeedsk8-deploy-development
a few seconds ago 3 sec

Links:
- Console output

staging

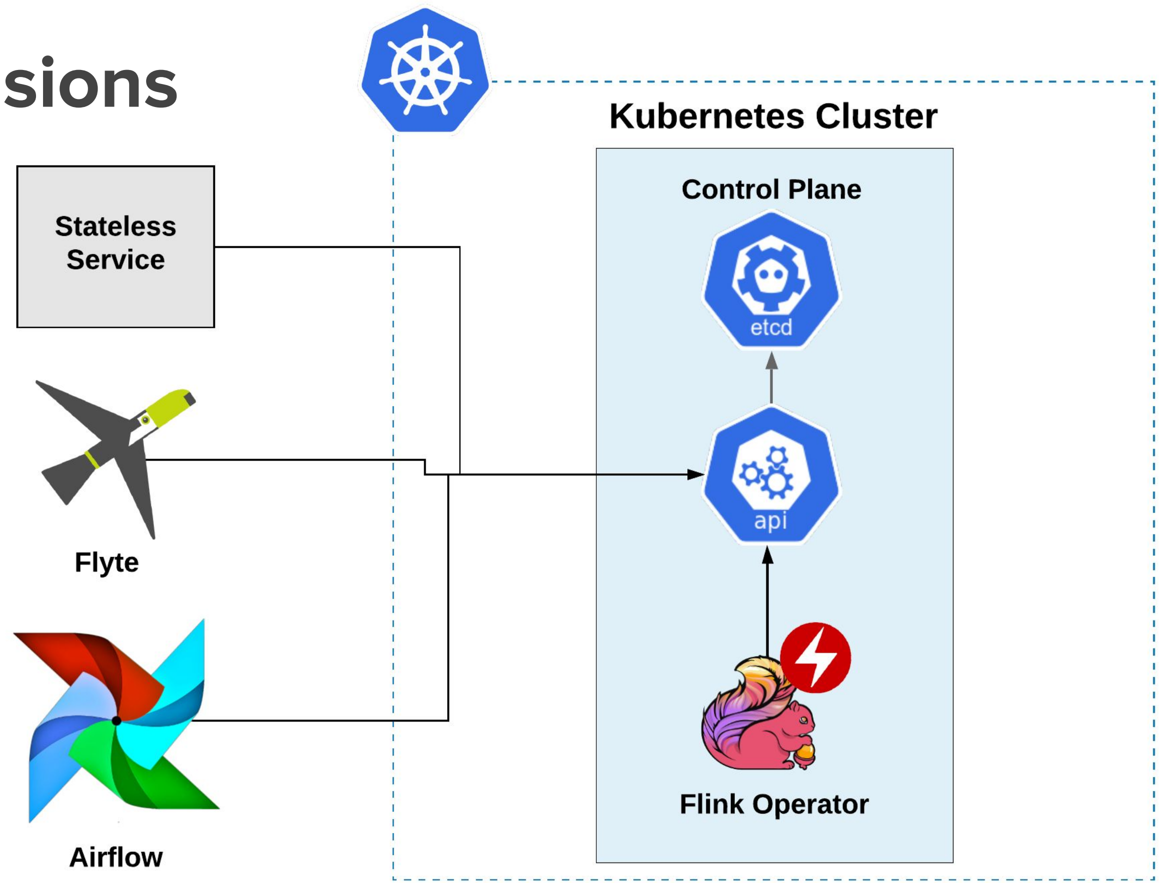
streamingspeedsk8-deploy-staging

Links:
- Console output
- Grafana (staging)
- Kibana (staging)

Signals (click for info):

Ecosystem

Future Extensions



Roadmap

Open Source

- **Last week of April***
- Project status: **Alpha**
- @Lyft:
  - Active development and testing in staging.
- Future
  - Flink Job failure handling
  - Tooling to manage CRD

Coming soon: *https://github.com/lyft/flinkk8soperator*

We're Hiring! Apply at www.lyft.com/careers

Data Engineering

Engineering Manager
San Francisco

Software Engineer
San Francisco, Seattle, & New York City

Data Infrastructure

Engineering Manager
San Francisco

Software Engineer
San Francisco & Seattle

Experimentation

Software Engineer
San Francisco

Observability

Software Engineer
San Francisco

Streaming

Software Engineer
San Francisco

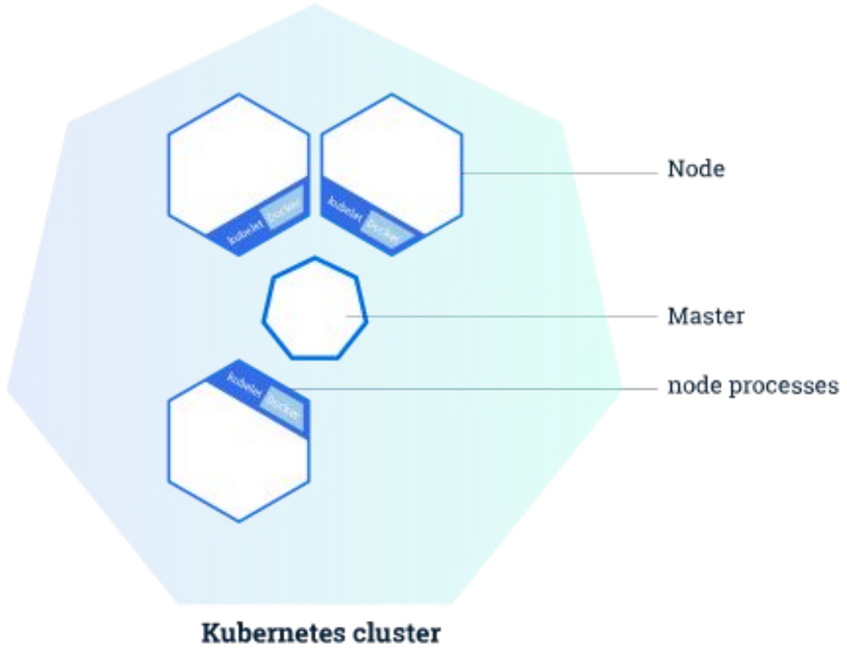
Thank you

Questions please!

Background

Example of Deployment

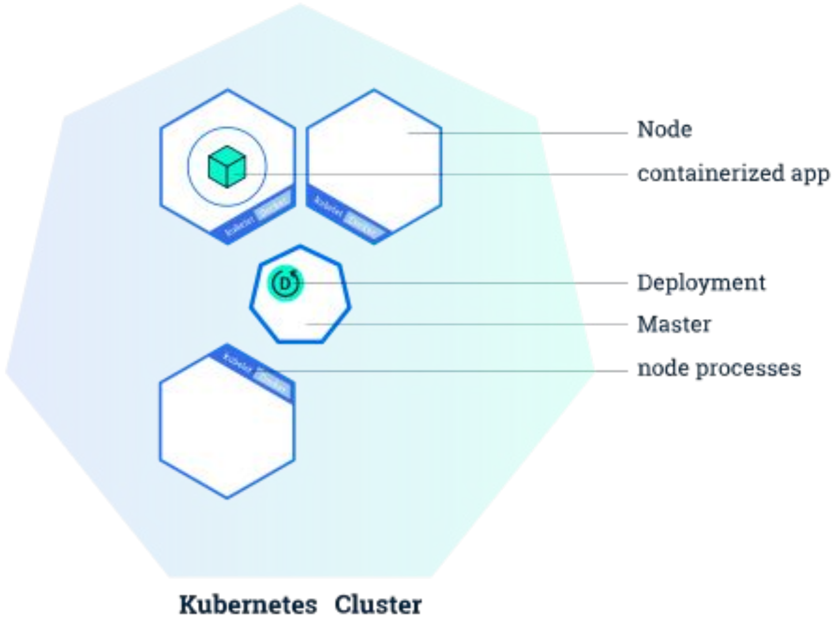
- User requests for a **Deployment** @ master
- Master accepts the request
- **Desired State:** 1 Pod running
- **Current State:** 0 Pods running



Background

Kubernetes 101

1. Master requests **Pod creation**
   - Current State: Deployment unhealthy
2. Master receives **pod created** event
   - Current State: Deployment healthy
3. Now if the pod crashes/dies etc
   - Current State: Deployment unhealthy
4. Goto 1


Node
containerized app
Deployment
Master
node processes
Kubernetes Cluster