Teaching Apache Spark Applications to Manage Their Workers Elastically

Erik Erlandson
Trevor McKay
Red Hat, Inc.

Introduction

Trevor McKay

Principal SW Engineer; Red Hat, Inc.

Emerging Technologies Group

Erik Erlandson

Senior SW Engineer; Red Hat, Inc.

Emerging Technologies Group

Oshinko Development

Insightful Applications

Internal Data Science

Insightful Applications

Outline

- Trevor
  - container orchestration
  - containerizing spark
- Erik
  - spark dynamic allocation
  - metrics
  - elastic worker daemon
- Demo

Containerizing Spark

- Container 101
  - What is a container?
  - Docker, Kubernetes and OpenShift
- Why Containerize Spark?
- Oshinko
  - features
  - components
  - cluster creation example

What is a container?

- A process running in a namespace on a container host
  - separate process table, file system, and routing table
  - base operating system elements
  - application-specific code
- Resources can be limited through cgroups

Docker and Kubernetes

- “Docker is the world's leading software containerization platform” www.docker.com
  - Open-source tool to build, store, and run containers
  - Images can be stored in shared registries

- “Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts” kubernetes.io

OpenShift Origin

- Built around a core of Docker and Kubernetes
- Adds application lifecycle management functionality and DevOps tooling. www.openshift.org/
  - multi-tenancy
  - Source-to-Image (S2I)
- Runs on your laptop with “oc cluster up”

Why Containerize Spark?

- Repeatable clusters with no mandatory config
- Normal users can create a cluster
  - No special privileges, just an account on a management platform

Why Containerize Spark?

- Containers allow a cluster-per-app model
  - Quick to spin up and spin down
  - Isolation == multiple clusters on the same host
  - Data can still be shared through common endpoints
  - Do I need to share a large dedicated cluster?

Why containerize Spark?

- Ephemeral clusters conserve resources
- Kubernetes makes horizontal scale out simple
  - Elastic Worker daemon builds on this foundation
  - Elasticity further conserves resources

Deeper on Spark + Containers

Optimizing Spark Deployments for Containers: Isolation, Safety, and Performance

- William Benton (Red Hat)
- Thursday, February 9
- 11:40 AM – 12:10 PM
- Ballroom A

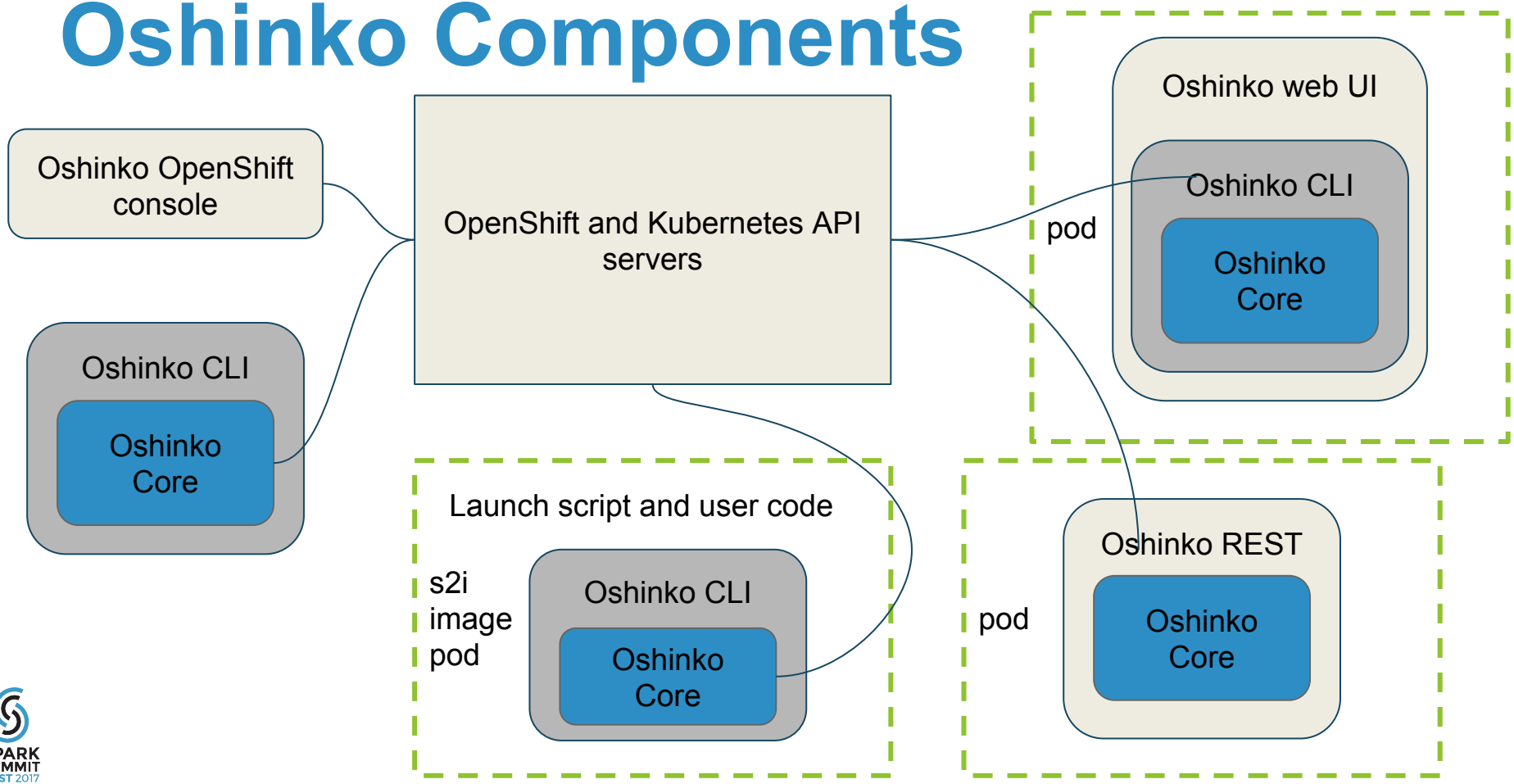
Oshinko: Simplifying further

- Containers simplify deployment but still lots to do ...
  - Create the master and worker containers
  - Handle spark configuration
  - Wire the cluster together
  - Allow access to http endpoints
  - Tear it all down when you're done
- Oshinko treats clusters as abstractions and does this work for us

Oshinko Features

- CLI, web UI, and REST interfaces
- Cluster creation with sane defaults (name only)
- Scale and delete with simple commands
- Advanced configuration
  - Enable features like Elastic Workers with a flag
  - Specify images and spark configuration files
  - Cluster configurations persisted in Kubernetes
- Source-to-Image integration (pyspark, java, scala)

Oshinko Components



Creating a Cluster

CLI from a shell …

```
$ oshinko-cli create mycluster --storedconfig=clusterconfig \
   --insecure-skip-tls-verify=true --token=$TOKEN
```

Using REST from Python …

```
import requests
r = requests.post("http://oshinko-rest/clusters",
                  json={"name": clustername,
                        "config": {"name": "clusterconfig"}
                       })
```

What is a cluster config?

```
$ oc export configmap clusterconfig

      apiVersion: v1
      data:
        metrics.enable: "true"
        scorpionstare.enable: "true"
        sparkimage: docker.io/manyangled/var-spark-worker:latest
        sparkmasterconfig: masterconfig
        sparkworkerconfig: workerconfig
      kind: ConfigMap
      metadata:
        creationTimestamp: null
        name: clusterconfig
```

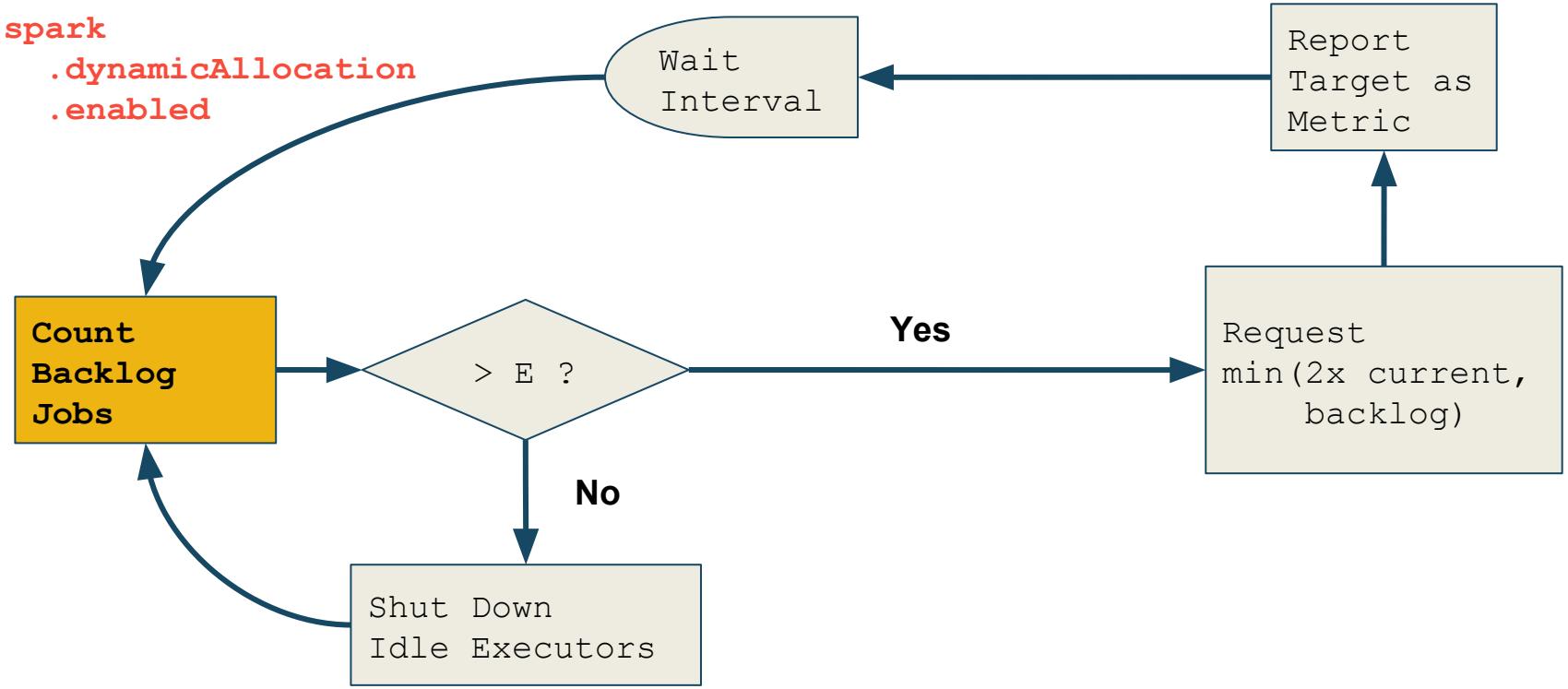
Source for demo's oshinko-rest

- Metrics implementation is being reviewed
  - using carbon and graphite today
  - investigating jolokia metrics

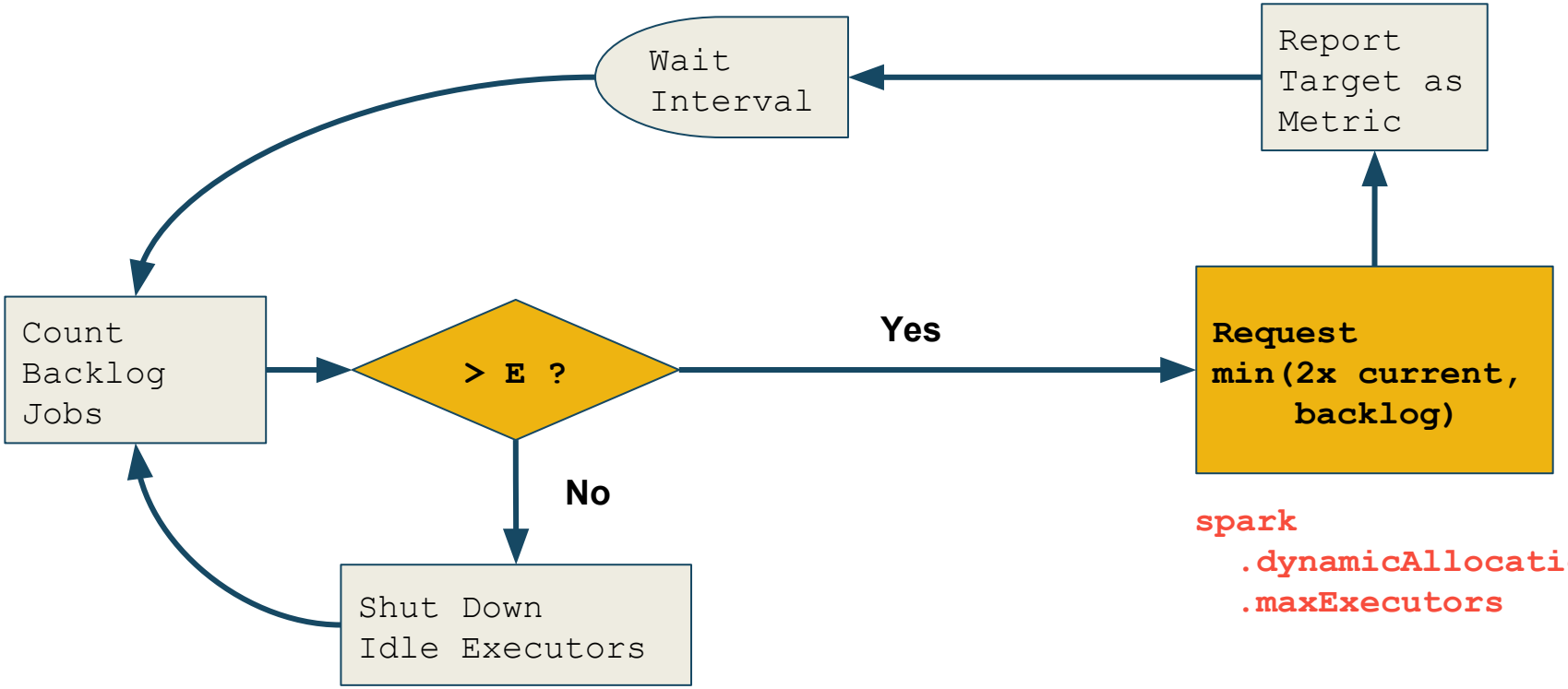
- Metrics and elastic workers currently supported at https://github.com/tmckayus/oshinko-rest/tree/metrics

- Both features will be merged to oshinko master soon

Dynamic Allocation

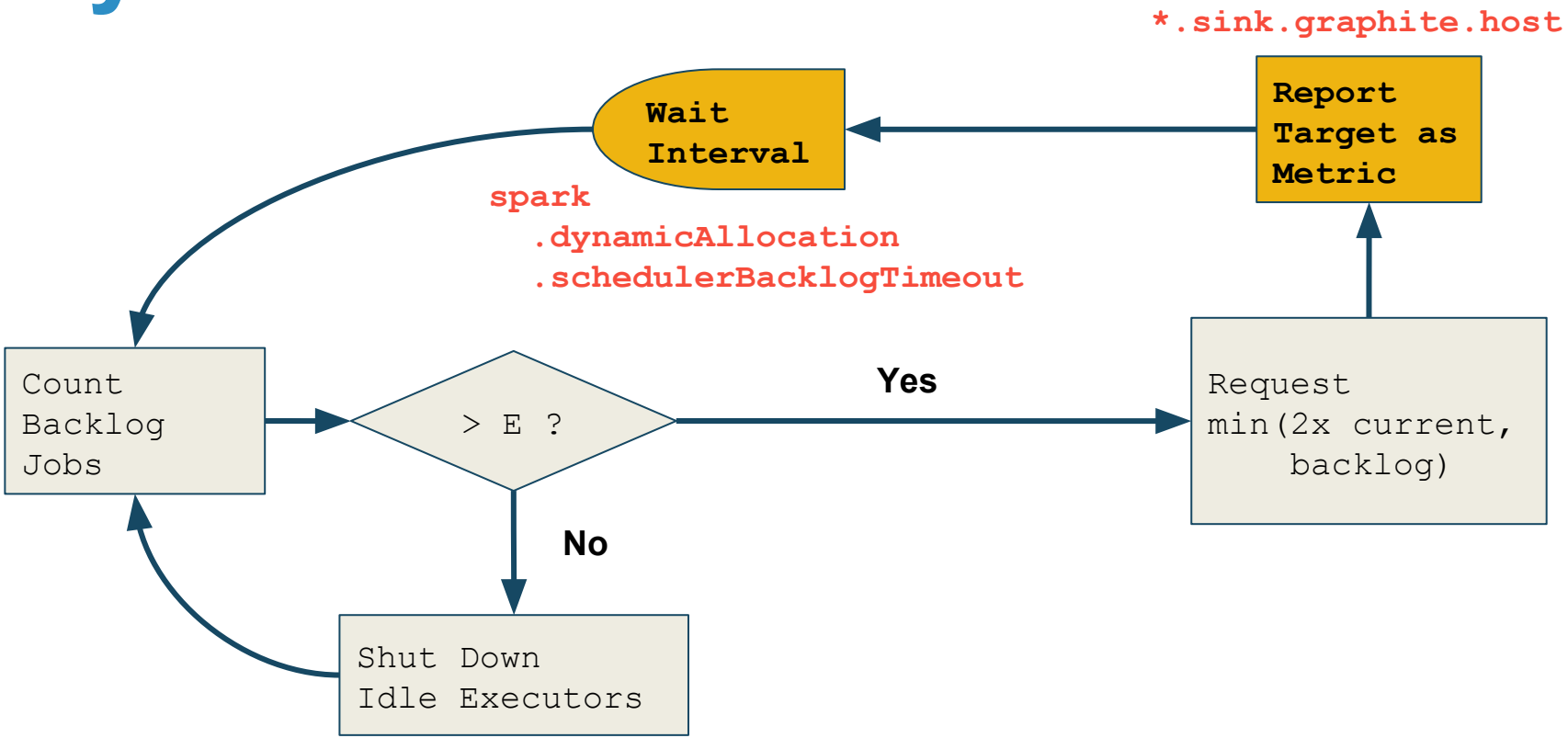


Dynamic Allocation

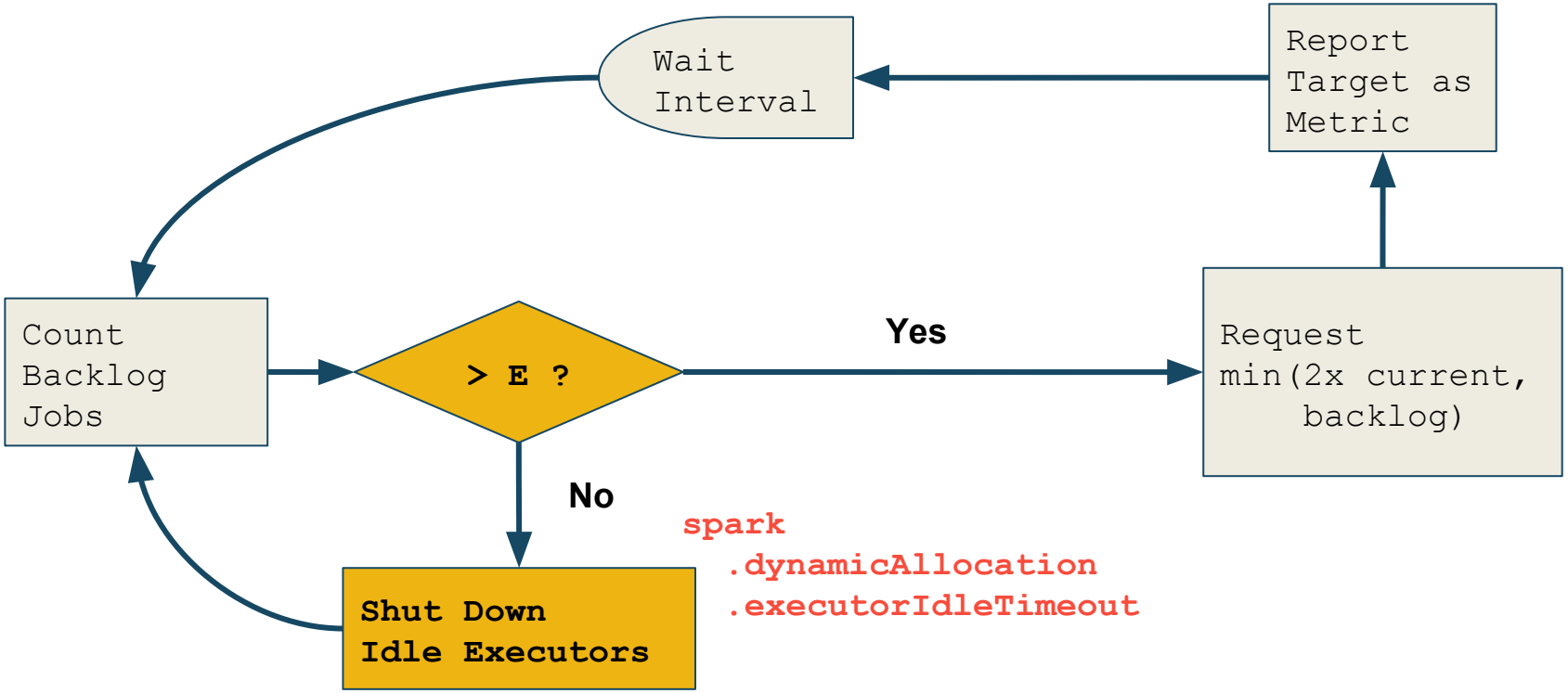



```
spark
  .dynamicAllocation
  .maxExecutors
```

Dynamic Allocation



Dynamic Allocation



Executor Scaling

```
spark.dynamicAllocation.initialExecutors

>= spark.dynamicAllocation.minExecutors

<= spark.dynamicAllocation.maxExecutors

<= backlog jobs (<= RDD partitions)
```

Shuffle Service

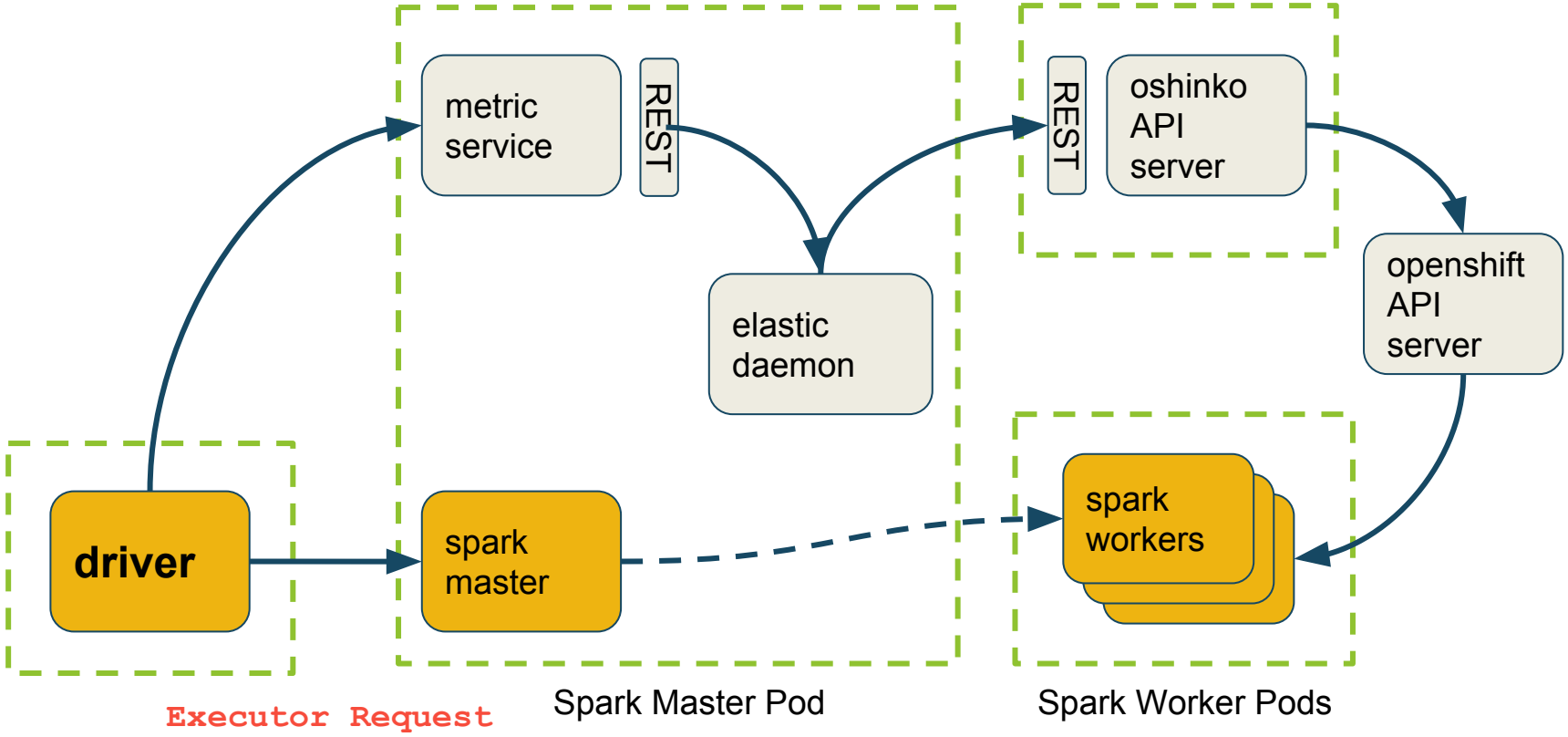
- Caches shuffle results independent of Executor
- Saves results if Executor is shut down
- Required for running Dynamic Allocation
- `spark.shuffle.service.enabled = true`

Dynamic Allocation Metrics

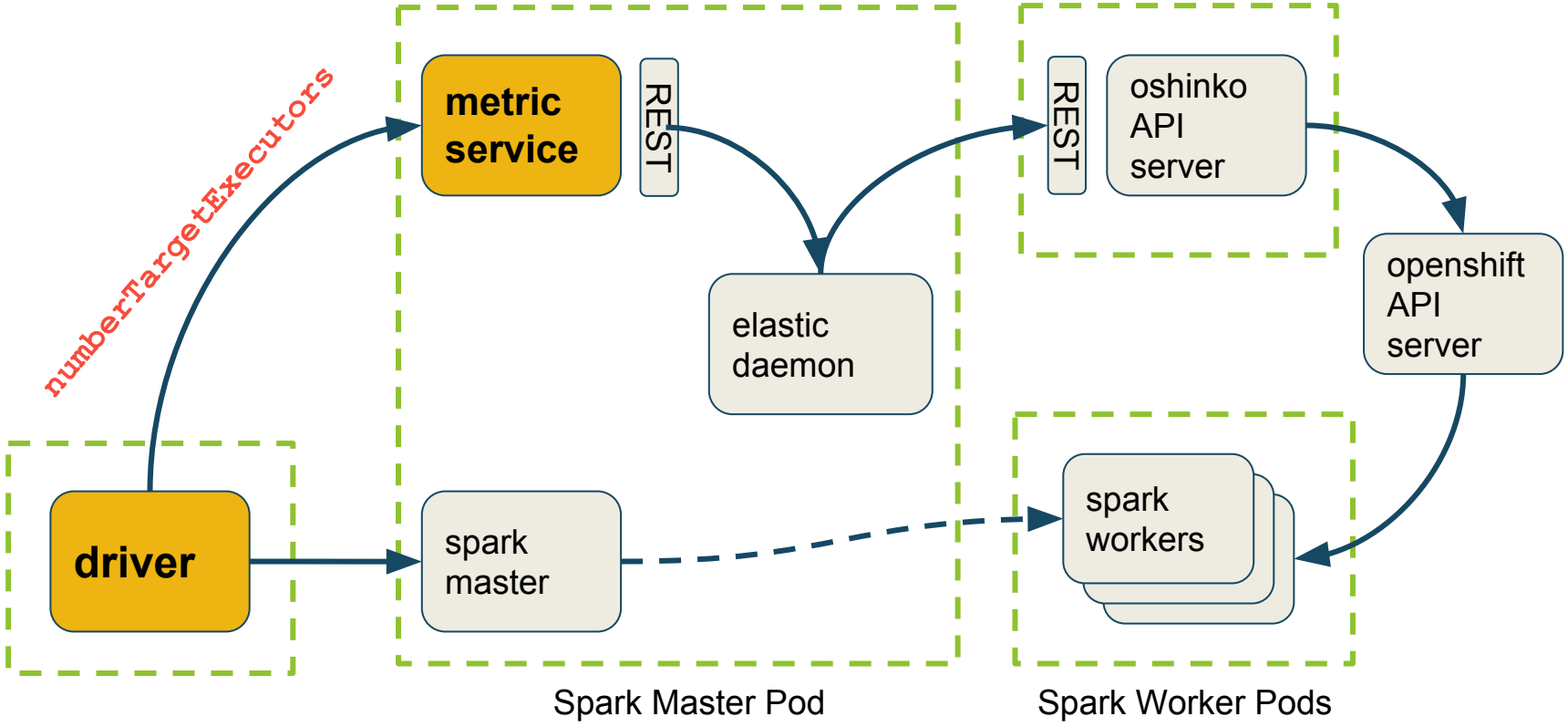
Published by the ExecutorAllocationManager

`numberExecutorsToAdd`	Additional executors requested
`numberExecutorsPendingToRemove`	Executors being shut down
`numberAllExecutors`	Executors in any state
`numberTargetExecutors`	**Total requested (current+additional)**
`numberMaxNeededExecutors`	Maximum that could be loaded

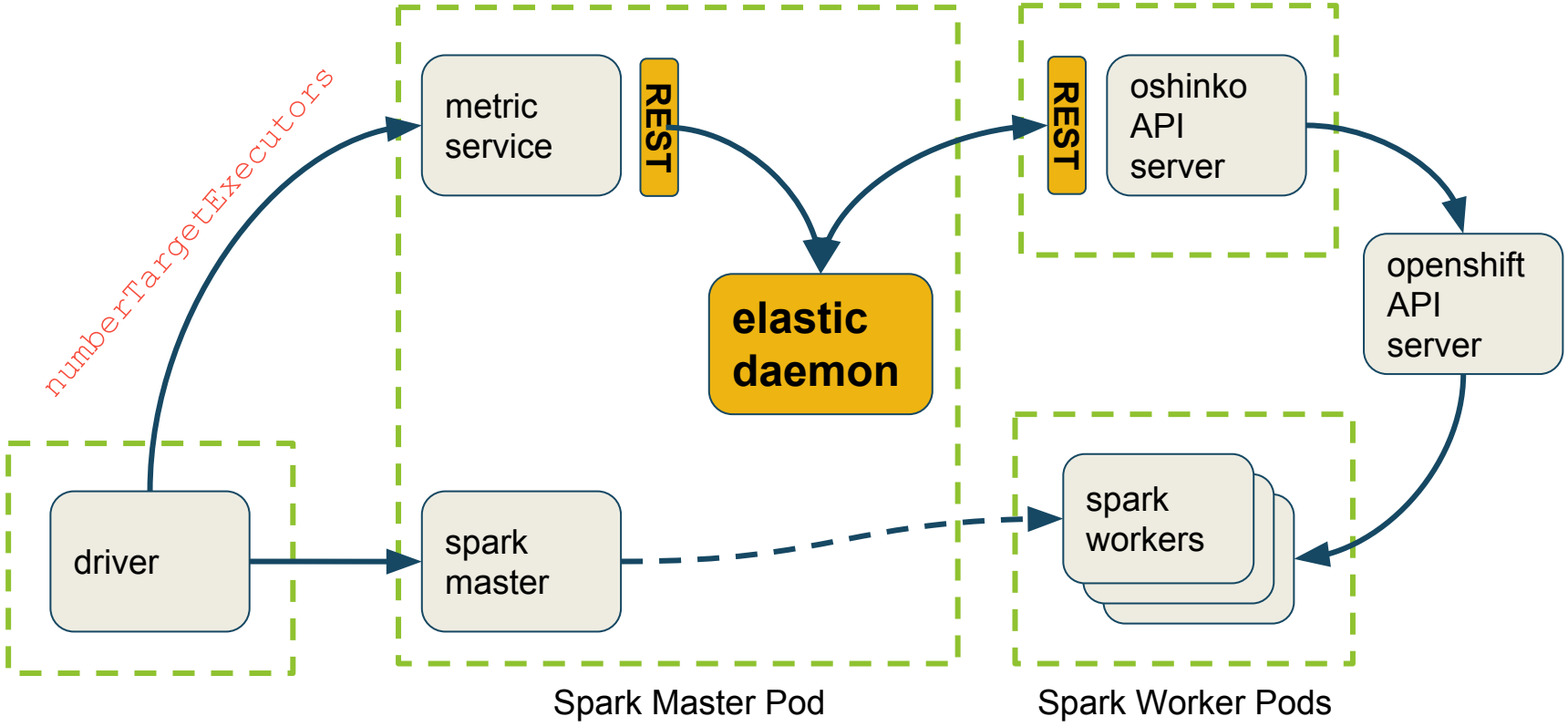
Elastic Worker Daemon



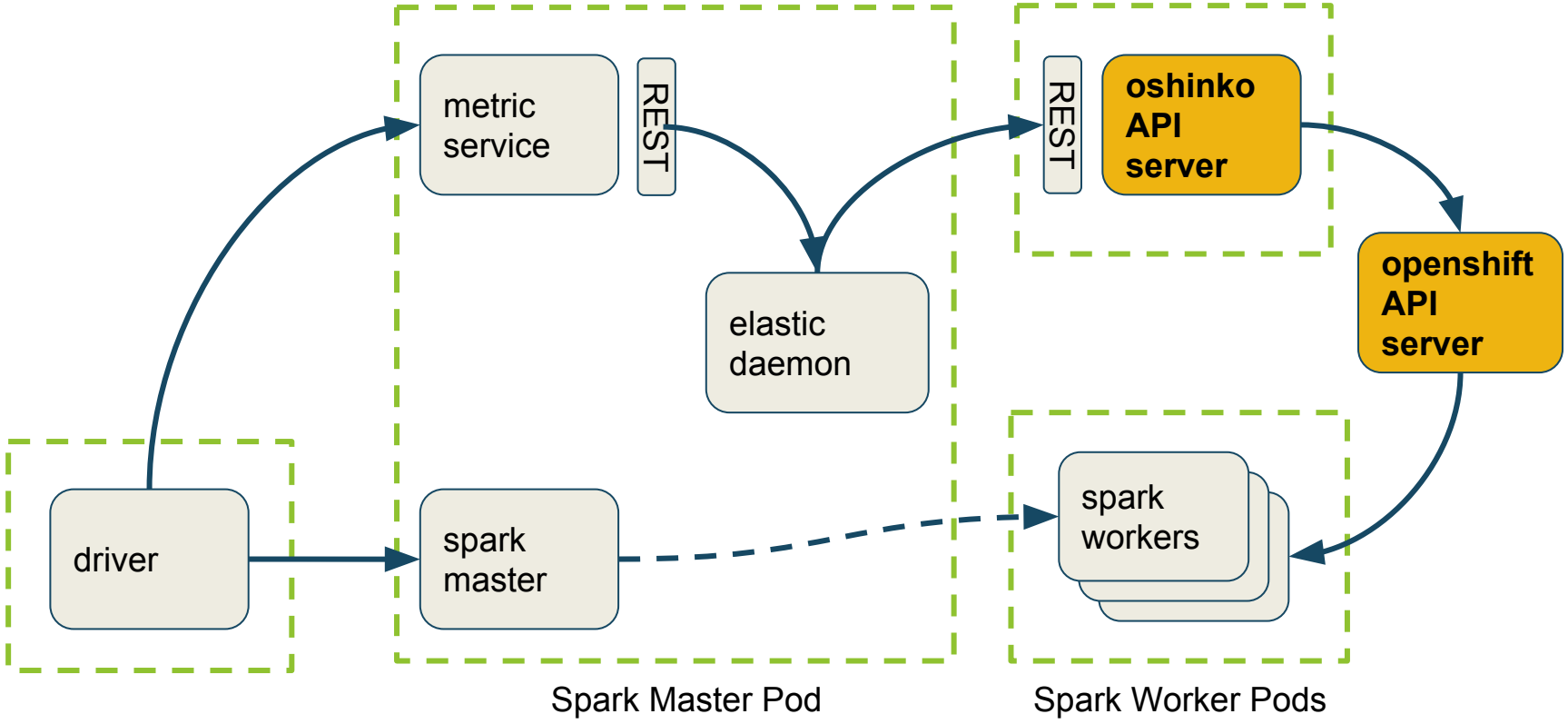
Elastic Worker Daemon



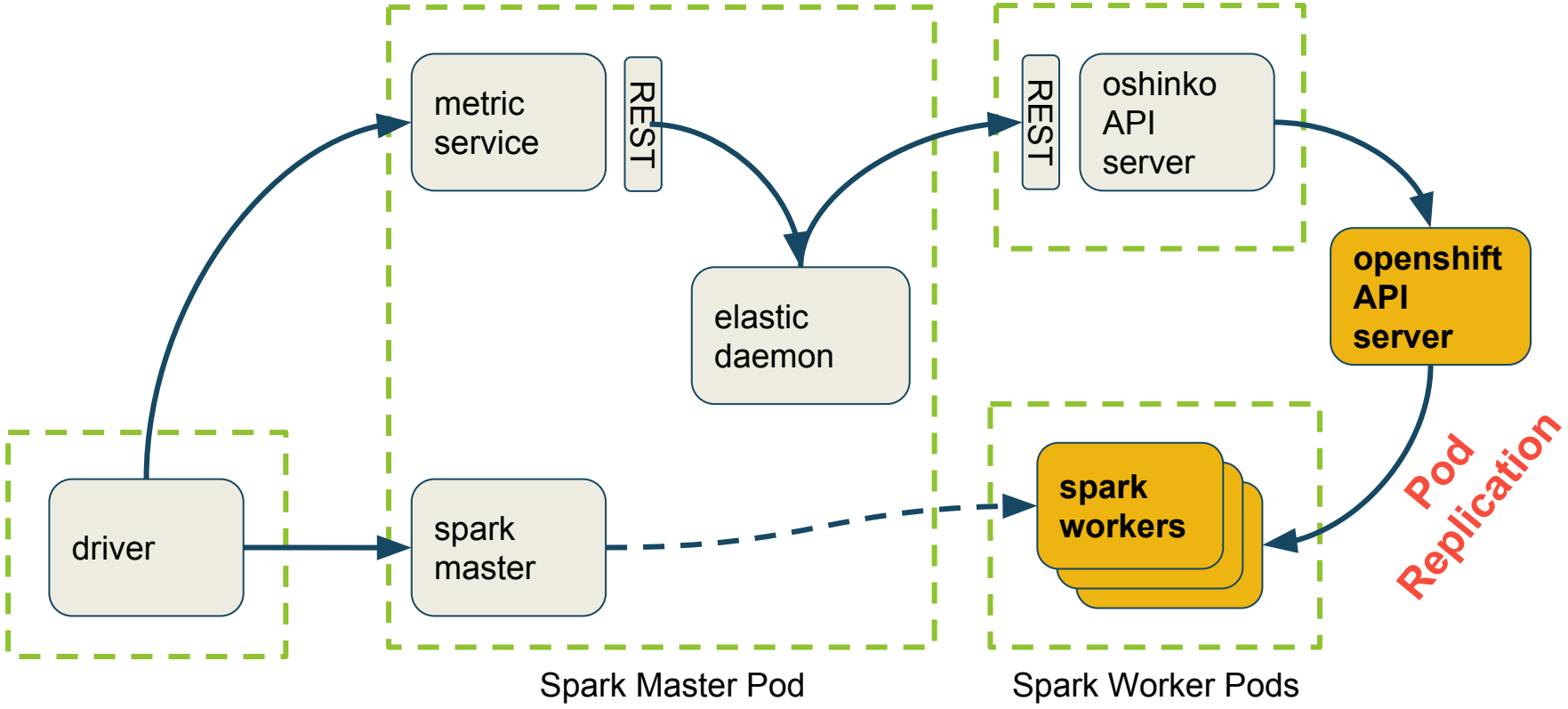
Elastic Worker Daemon



Elastic Worker Daemon



Elastic Worker Daemon



Demo

Demo

Radanalytics.io

New community landing page at

http://radanalytics.io/

Where to Find Oshinko

Oshinko and related bits:

http://github.com/radanalyticsio/

Docker images:

https://hub.docker.com/u/radanalyticsio/

Images and notebook for today's demo:

https://hub.docker.com/u/tmckay/

https://hub.docker.com/u/manyangled/

https://github.com/erikerlandson/var-notebook/pulls

Related Effort: Spark on K8s

- Native scheduler backend for Kubernetes
- https://github.com/apache-spark-on-k8s/spark
- Developer Community Collaboration

Thank You.

eje@redhat.com
tmckay@redhat.com