Scalable task distribution with Scala, Akka and Mesos

Dario Rexin
@evonox

mesosphere

What is Mesos?

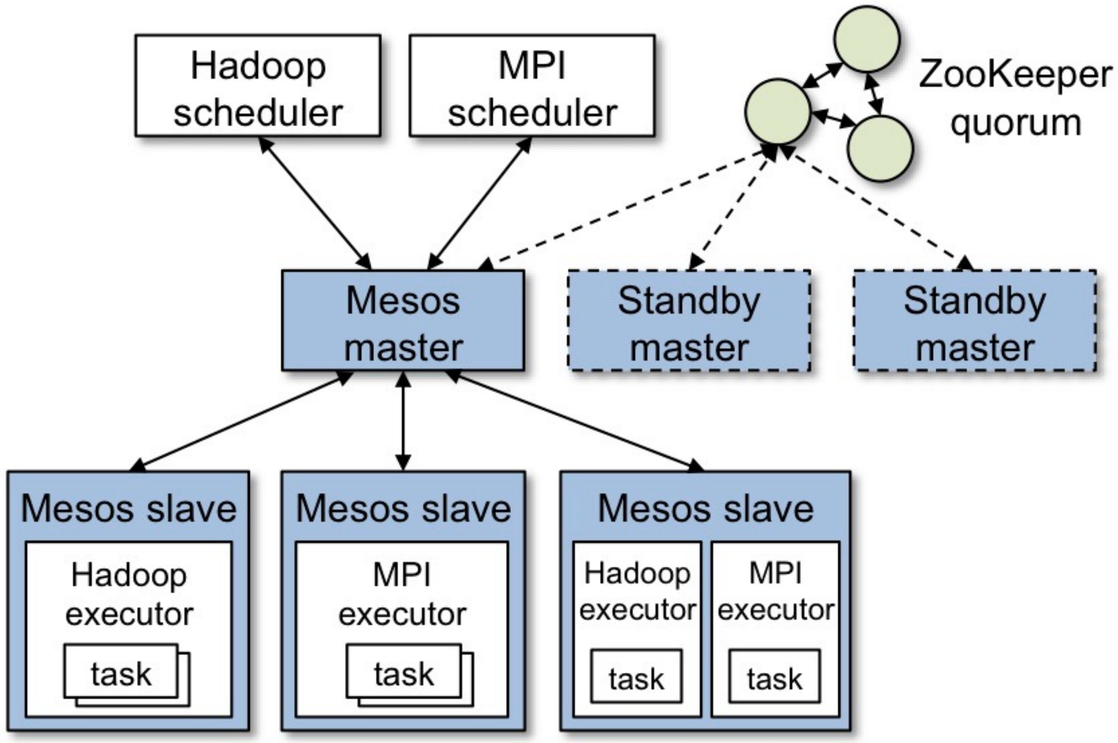
What is Mesos?

- Apache open source project
- Distributed systems kernel
- Multi resource scheduler (CPU, Memory, Ports, Disk)
- Scalable to 10,000s of nodes
- Fault tolerant
- First class Docker support

Distributed Systems Kernel

- Runs on every node
- Aggregates all resources in the cluster
- Provides applications with APIs for resource management and scheduling
- Offers resources to applications in a fair (configurable) manner

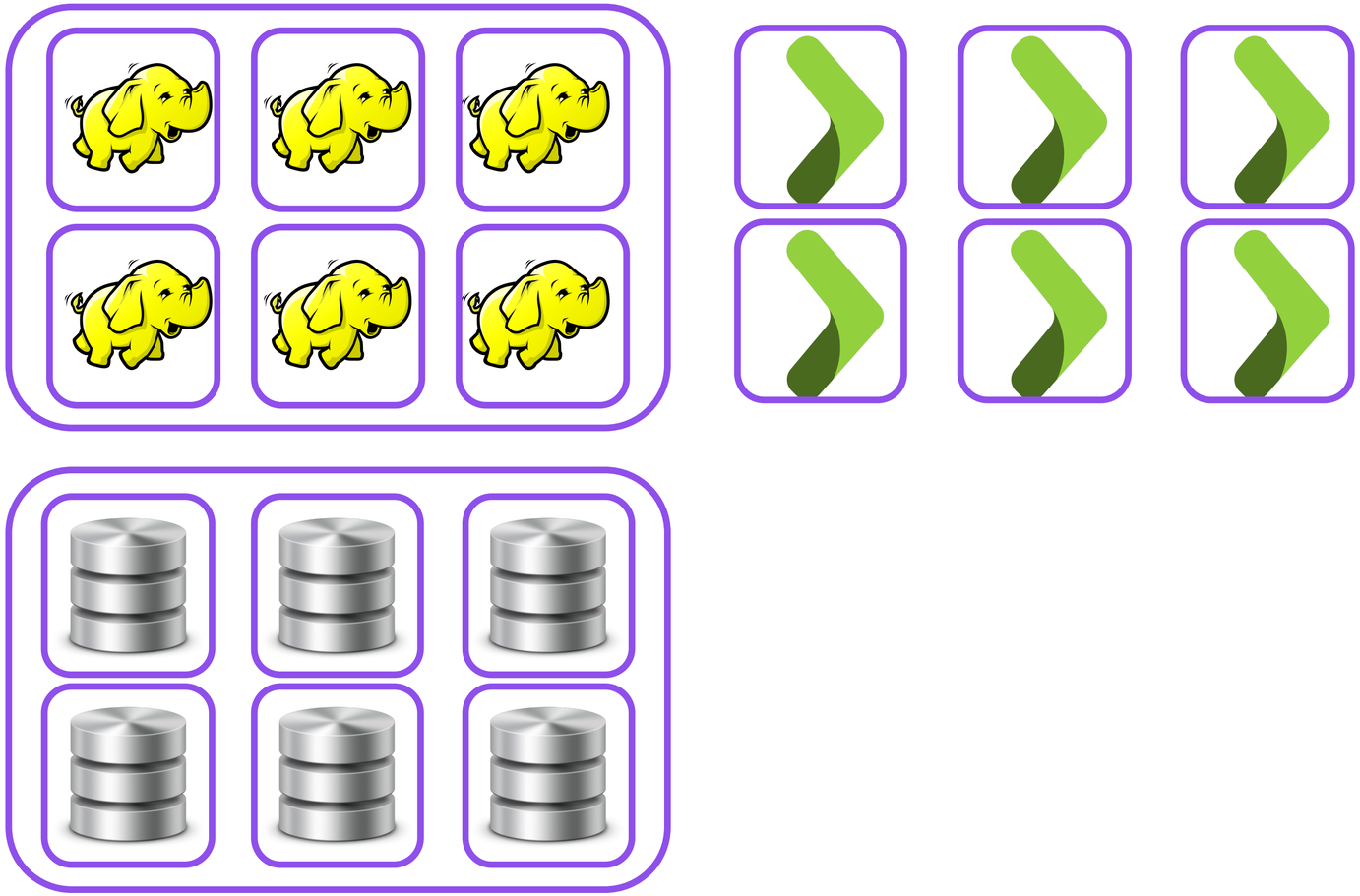
Fault Tolerance



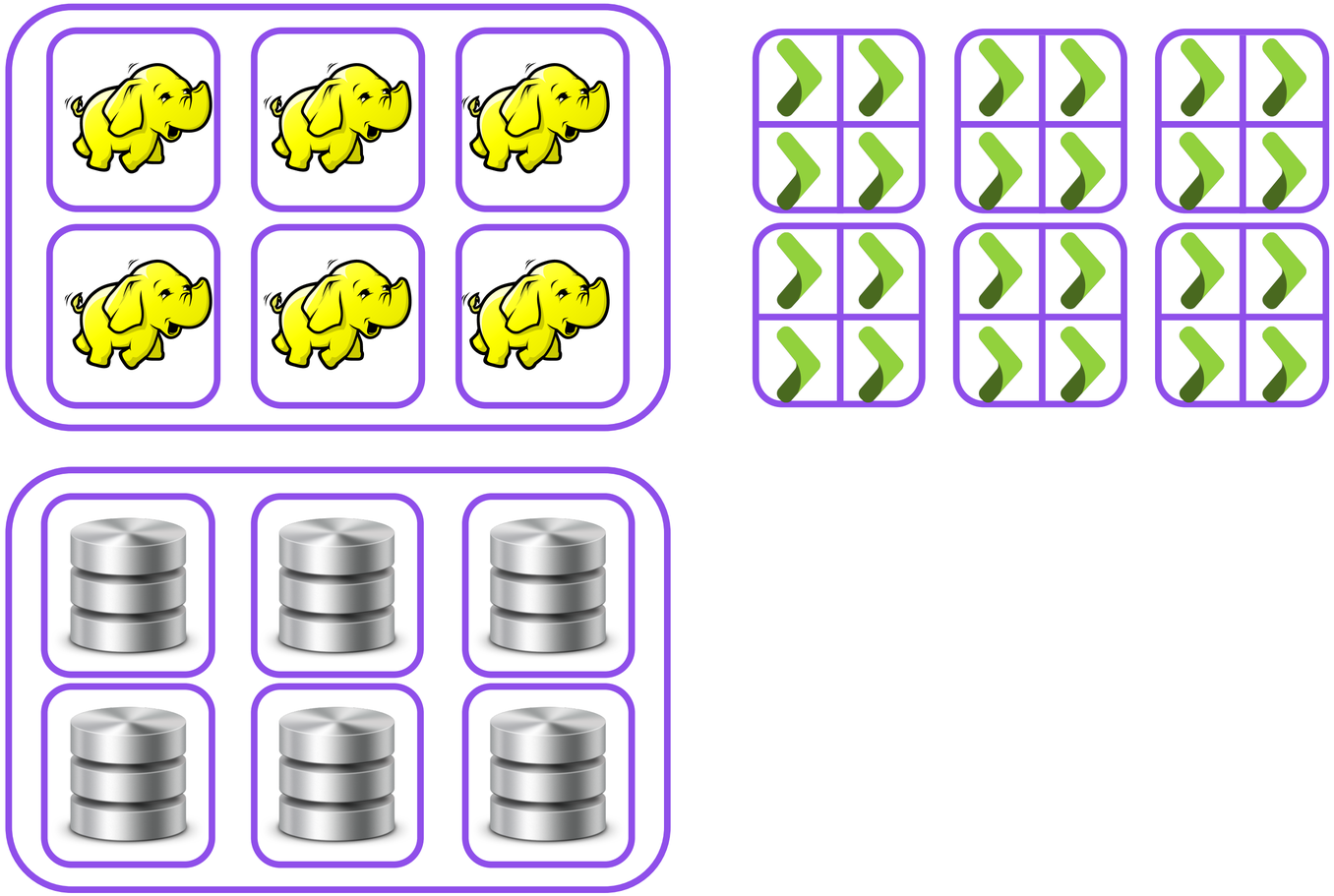
Why should I bother?

- Running many tasks on many machines
- Scaling up and down the number of tasks and machines
- Handling failures in the cluster
- Better resource utilization through multi-tenancy

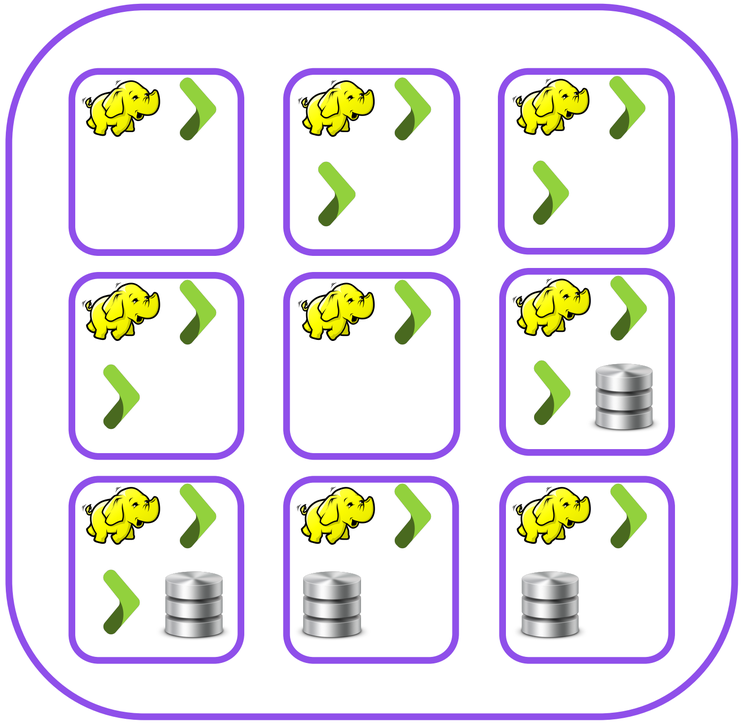
Static partitioning Waste of resources!



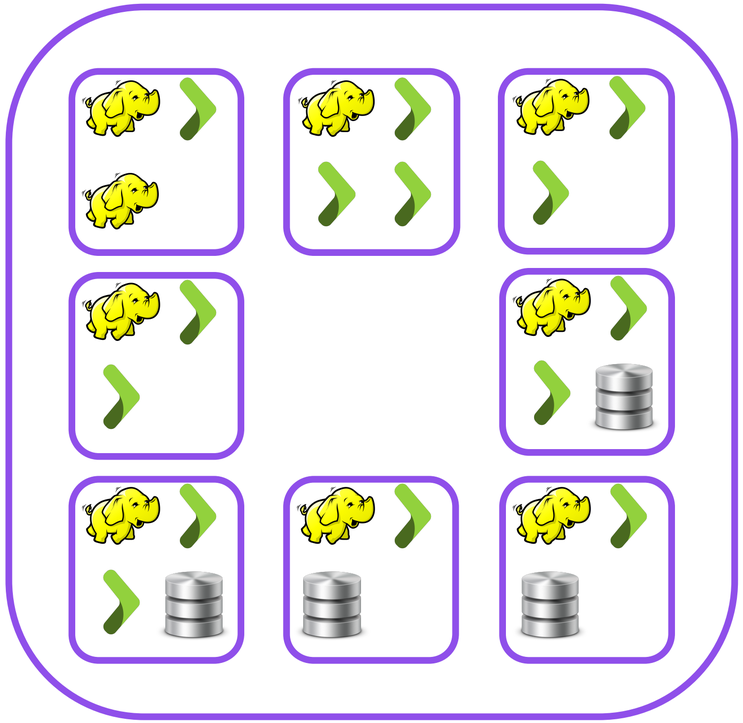
Virtualization Operational overhead!



Multi tenancy + automatic task distribution



Multi tenancy + automatic task distribution



Say hi to Marathon

What is Marathon?

- Distributed init system
- Fault tolerant
- Manages deployments
- Checks health of applications
- Manages task and machine failures
- Runs Docker containers

Deploying apps with Marathon

```
curl -XPOST -H “Content-Type: application/json” http://marathon:8080/v2/apps -d ‘{
    “id”: “my-app”,
    “cmd”: “python -m SimpleHTTPServer $PORT0”,
    “cpus”: 1,
    “mem”: 64,
    “instances”: 10,
    “ports”: [0]
}’
```

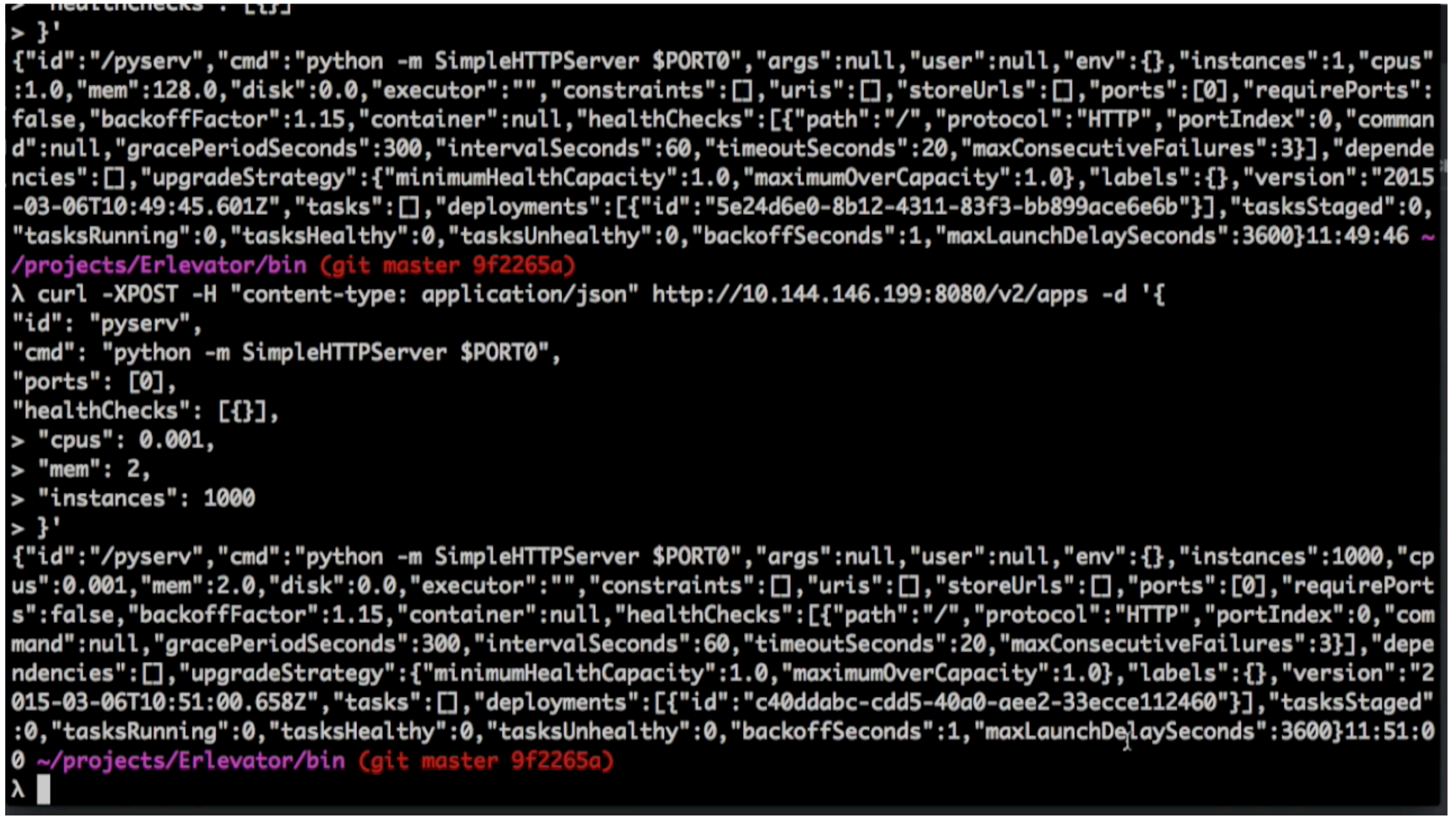
Deploying apps with Marathon

```
{
  “steps”: [
    {
      “actions”: [
        {
          “action”: ”StartApplication”,
          “app”:”/my-app”
        }
      ]
  }
}
```


Deploying apps with Marathon

ID ▲	Memory (MB)	CPUs	Tasks / Instances	Health	Status
/my-app	64	1	0 / 10		Deploying

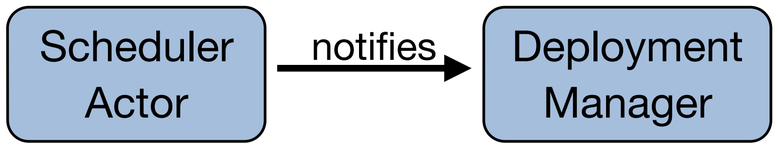
Deploying apps with Marathon



Deploying apps with Marathon

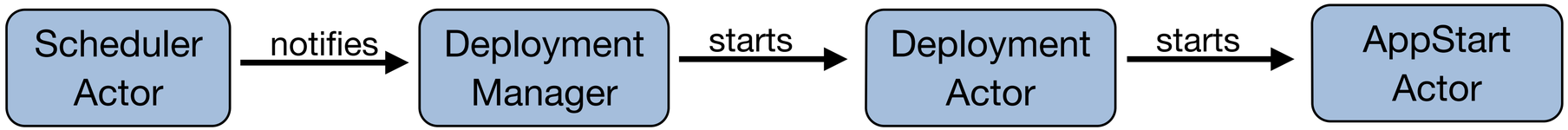
Scheduler Actor

Deploying apps with Marathon

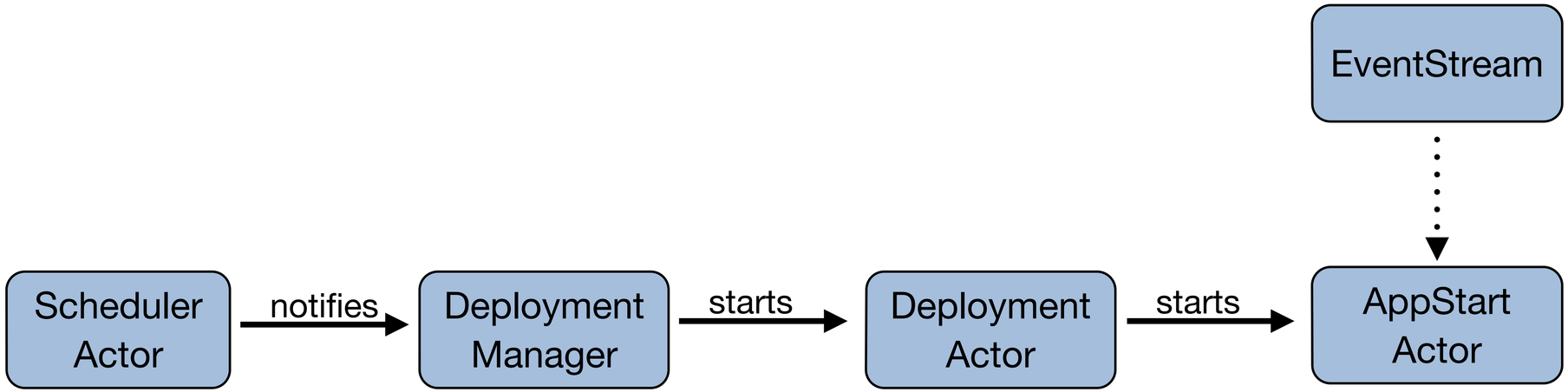


Deploying apps with Marathon

Deploying apps with Marathon



Deploying apps with Marathon



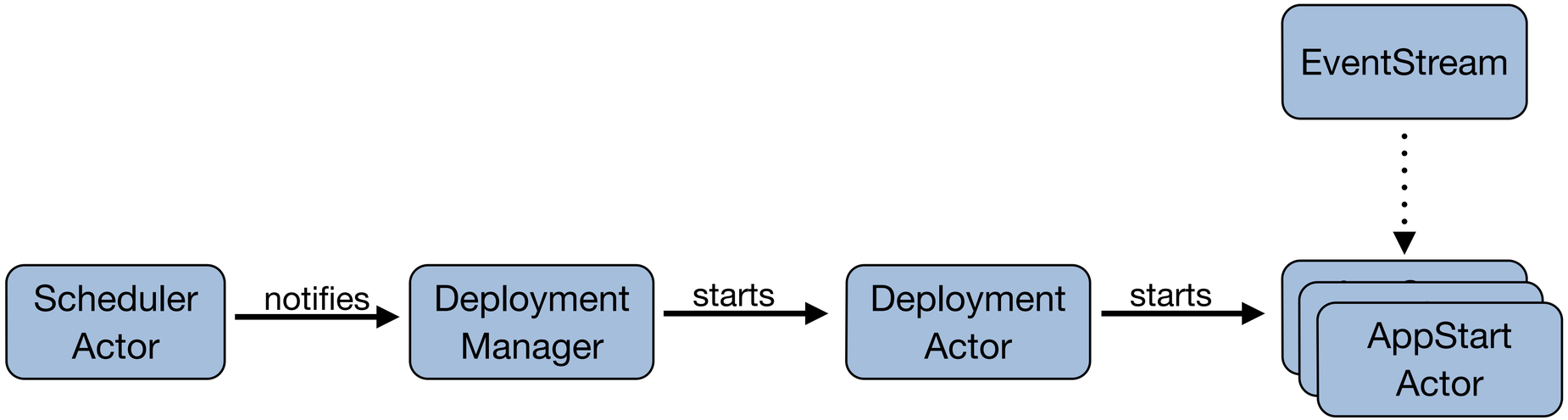
Deploying groups with Marathon

```
curl -XPOST -H “Content-Type: application/json” http://marathon:8080/v2/groups -d ‘{
    “id”: “my-group”,
    “apps”: [
        {“id”: “my-app”, … },
        {“id”: “my-other-app”, … },
        {“id”: “yet-another-app”, … }
    ]
}’
```

Deploying apps with Marathon

```
{
  “steps”: [
    {
      “actions”: [
        { “action”: ”StartApplication”, “app”:”/my-app” },
        { “action”: ”StartApplication”, “app”:”/my-other-app” },
        { “action”: ”StartApplication”, “app”:”/yet-another-app” }
      ]
  }
}
```

Deploying groups with Marathon



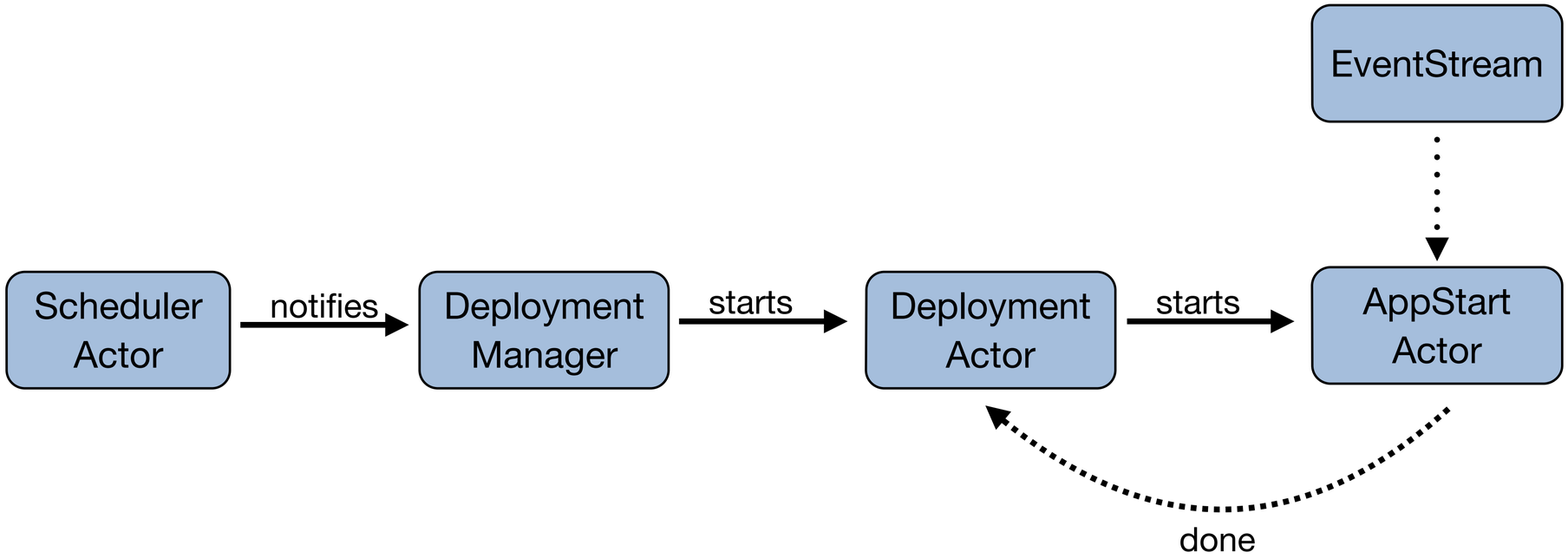
Dependencies between apps

```
curl -XPOST -H “Content-Type: application/json” http://marathon:8080/v2/groups -d ‘{
    “id”: “my-group”,
    “apps”: [
        {“id”: “my-app”, “dependencies”: [“my-other-app”] … },
        {“id”: “my-other-app”, “dependencies”: [“yet-another-app”] … },
        {“id”: “yet-another-app”, … }
    ]
}’
```

Deploying dependent groups with Marathon

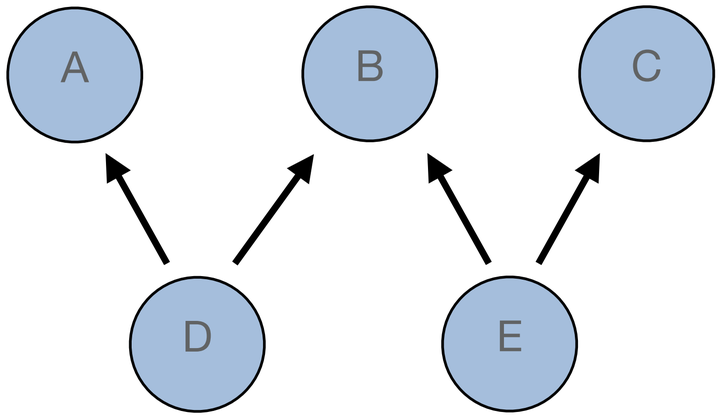
```
{
  “steps”: [
    {
      “actions”: [{ “action”: ”StartApplication”, “app”:”/yet-another-app” }]
    },
    {
      “actions”: [{ “action”: ”StartApplication”, “app”:”/my-other-app” }]
    },
    {
      “actions”: [ { “action”: ”StartApplication”, “app”:”/my-app” }]
    }
  ]
}
```


Deploying dependent groups with Marathon

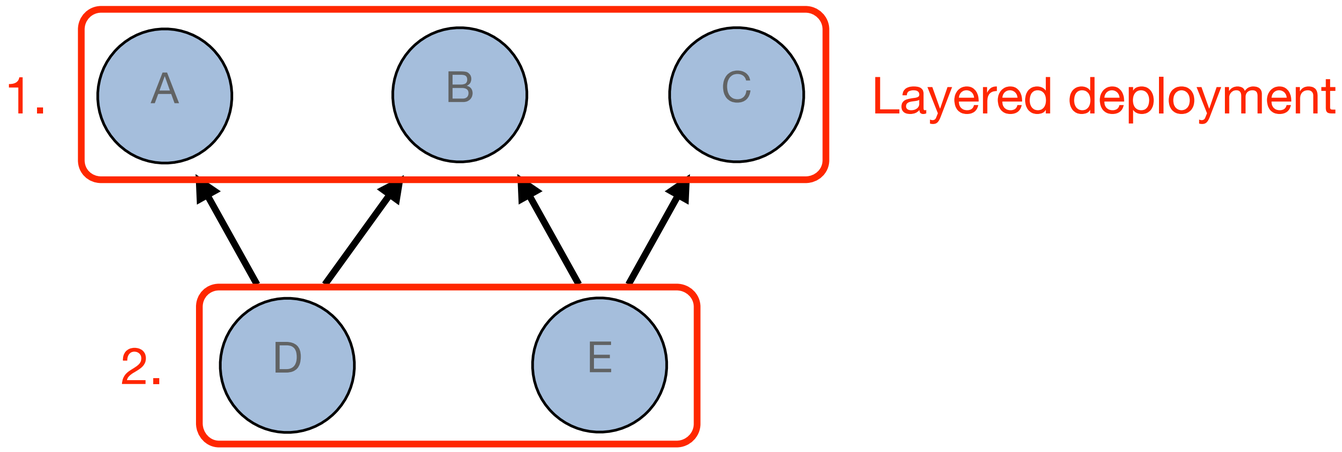


Deploying dependent groups with Marathon

= depends on



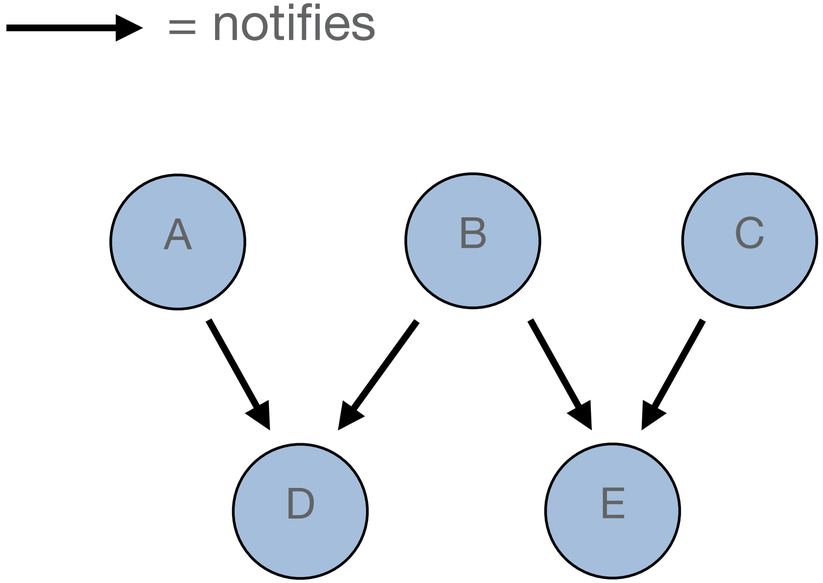
Deploying dependent groups with Marathon



Layered deployment

Can we do better?

We can do better!



Writing custom frameworks

Bindings for many languages

- C++
- Java
- Scala
- Clojure
- Haskell
- Go
- Python

Simple API

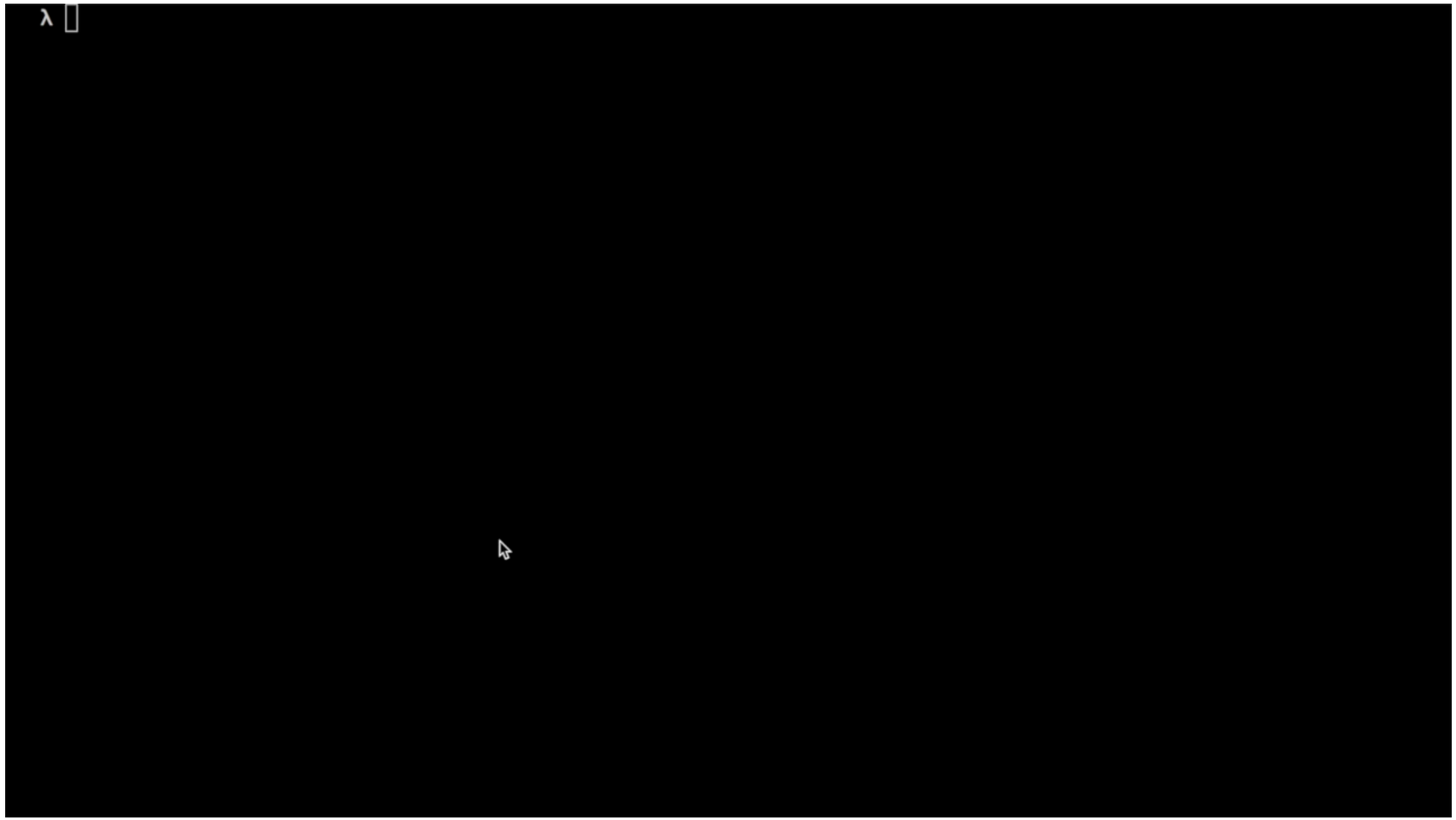
- registered
- reregistered
- resourceOffers
- offerRescinded
- statusUpdate
- frameworkMessage
- disconnected
- slaveLost
- executorLost
- error

akka-mesos (work in progress)

- Non-blocking
- Stream of messages instead of callbacks
- Wrappers around the protobuf messages
- No JNI

Auto scaling applications

Auto scaling applications



Auto scaling applications

```
system.actorOf(ClusterSingletonManager.props(
  singletonProps = Props(classOf[MesosScheduler]),
  singletonName = "consumer",
  terminationMessage = PoisonPill,
  role = Some("master")),
  name = "mesos-scheduler")
```

Auto scaling applications

```
val frameworkInfo = FrameworkInfo(
  name = "autoscale",
  user = "user",
  failoverTimeout = Some(5.minutes),
  checkpoint = Some(true)
)
```

Auto scaling applications

```
framework = Mesos(context.system)
    .registerFramework(
      Success(PID("127.0.0.1", 5050, "master")),
      frameworkInfo)

implicit val materializer = ActorFlowMaterializer()

framework.schedulerMessages.runForeach (self ! _)
Cluster(context.system).subscribe(self, classOf[ClusterMetricsChanged])
```

Auto scaling applications

```
case ClusterMetricsChanged(metrics) =>
  val (loadSum, heapSum) = metrics.foldLeft((0.0, 0.0)) {
    case ((loadAcc, heapAcc), metric) =>
      val load = metric.metrics.collectFirst {
        case x if x.name == "system-load-average" => x.value.doubleValue()
      } getOrElse 0.0
      val heap = metric.metrics.collectFirst {
        case x if x.name == "heap-memory-used" => x.value.doubleValue()
      } getOrElse 0.0

      (loadAcc + load, heapAcc + heap)
  }

  val loadAvg = loadSum / metrics.size
  val heapAvg = heapSum / metrics.size
```

Auto scaling applications

```
if ((loadAvg > upperLoadThreshold ||
     heapAvg > upperHeapThreshold) &&
     scaleBackoff.isOverdue()) {
  log.info("Scaling up!")
  scaleUp = true
  scaleBackoff = 30.seconds.fromNow
}
```

Auto scaling applications

```
else if ((loadAvg < lowerLoadThreshold &&
          heapAvg < lowerHeapThreshold) &&
          scaleBackoff.isOverdue() &&
          runningTasks.nonEmpty) {
  log.info("Scaling down!")
  scaleUp = false
  val task = runningTasks.head
  framework.driver.killTask(task)
  scaleBackoff = 30.seconds.fromNow
}
```

Auto scaling applications

```
case ResourceOffers(offers) if scaleUp =>
  val matchingOffer = findMatchingOffer(offers)

  matchingOffer foreach { offer =>
    val taskInfo = buildTaskInfo(offer)
    framework.driver.launchTasks(Seq(taskInfo), Seq(offer.id))
    scaleUp = false
  }

  (offers diff matchingOffer.toSeq).foreach { offer =>
    framework.driver.declineOffer(offer.id)
  }
```

Auto scaling applications

```
case ResourceOffers(offers) =>
  log.info("No need to scale, declining offers.")
  offers.foreach(offer => framework.driver.declineOffer(offer.id))
```

Auto scaling applications

```
case StatusUpdate(update) =>
  update.status.state match {
    case TaskRunning =>
      runningTasks += update.status.taskId
    case TaskKilled | TaskError | TaskFailed | TaskFinished =>
      runningTasks -= update.status.taskId
    case _ =>
  }
  log.info(s"${update.status.taskId} is now ${update.status.state}")
  framework.driver.acknowledgeStatusUpdate(update)
```

Check out the projects

https://github.com/mesosphere/marathon

https://github.com/drexin/akka-mesos

https://github.com/apache/mesos

Launch a Mesosphere cluster on Google Compute Engine

https://google.mesosphere.com

mesosphere

Join the community

http://mesos.apache.org

https://mesosphere.github.io/marathon/

mesosphere

We are hiring!

http://mesosphere.com/jobs

Thanks for your attention

mesosphere