Managing Containers with Helix

Kanak Biscuitwala
Jason Zhang
Apache Helix Committers @ LinkedIn
helix.apache.org
@apachehelix
Intersection of Job Types
Intersection of Job Types
Intersection of Job Types

- DB
- Backup
- ETL

- DB
- Backup
- ETL

HDFS
Intersection of Job Types

Long-running and batch jobs running together!
Cloud Deployment

Applications with diverse requirements running together in a datacenter
Cloud Deployment

Applications with diverse requirements running together in a datacenter
Processes on Machines
Processes on Machines

No Isolation

- Machine
- Process
- VM
- Container
Processes on Machines

No Isolation

VM-based Isolation

Machine

Process

VM

Container
Processes on Machines

- **No Isolation**
- **VM-based Isolation**
- **Container-based Isolation**

- **Machine**
- **Process**
- **VM**
- **Container**
Processes on Machines

- Run as individual processes
  - Poor isolation or poor utilization
- Virtual machines
  - Better isolation
  - Xen, Hyper-V, ESX, KVM
- Containers
  - cgroup
  - YARN, Mesos
  - Super lightweight, dynamic based on application requirements
Processes on Machines

Virtualization and containerization significantly improve process isolation and open up possibilities for efficient utilization of physical resources.
Container-Based Solution
## Container-Based Solution

### System Requirements

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>64 MB</td>
<td>128 MB</td>
<td>256 MB</td>
</tr>
<tr>
<td></td>
<td>64 MB</td>
<td>128 MB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>64 MB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Container-Based Solution

Allocation

Machine

Container

- 64 MB
- 64 MB
- 128 MB

- 256 MB
- 64 MB

- 128 MB
Container-Based Solution
 Allocation

Machine
Process
Container

A
A
B

A
C

B
Container-Based Solution

Allocation

Containerization is powerful!
Container-Based Solution

Allocation

Machine

Process

Container

Containerization is powerful!
But do processes always fit so nicely?
Container-Based Solution

Over-Utilization

256 MB
Over-Utilization

Container-Based Solution

Process 1
Container-Based Solution

Over-Utilization

Outcome: Preemption and relaunch
Over-Utilization

Outcome: Preemption and relaunch

Machine

Process

Container

384 MB
Container-Based Solution

Over-Utilization

Outcome: Preemption and relaunch
Container-Based Solution

Under-Utilization

- Machine
  - Container
  - Process

- 384 MB
- 128 MB

HELIX
Container-Based Solution

Under-Utilization

Outcome: Over-provisioned until restart
Container-Based Solution

Failure

Machine

Process

Container

A

A

B

A

C

B
Container-Based Solution

Failure

Machine

Process

Container

A

A

B

B
Container-Based Solution

Failure

Outcome: Launch containers elsewhere
What about stateful systems?
Container-Based Solution

Failure

SLAVE
SLAVE
B

MASTER

B
Container-Based Solution

Failure

Without additional information, the master is unavailable until restart.
Container-Based Solution

Scaling

Machine
Process
Container

256 MB
50%

256 MB
50%

50%

HELIx
Container-Based Solution

Scaling
Container-Based Solution

Scaling

Outcome: Relaunch with new sharding
## Container-Based Solution

<table>
<thead>
<tr>
<th>Feature</th>
<th>Container-Based Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Utilization</strong></td>
<td>Application requirements define container size</td>
</tr>
<tr>
<td><strong>Fault Tolerance</strong></td>
<td>New container is started</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Workload is repartitioned and new containers are brought up</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>Existence</td>
</tr>
</tbody>
</table>
Container-Based Solution

The container model provides flexibility within machines, but assumes homogeneity of tasks within containers.

We need something finer-grained.
Task-Based Solution
Task-Based Solution

System Requirements

A  complete in less than 5 hours

B  always have 2 containers running

C  response time should be less than 50 ms
Task-Based Solution

Allocation

Machine

Task

Container

A

B

C

A

C

B

C
Task-Based Solution

Over-Utilization

Machine

Task

Container

---

HELIX
Task-Based Solution
Over-Utilization

Task 1
Task-Based Solution

Over-Utilization

Machine

Task

Container

Task 1
Task-Based Solution

Over-Utilization

- Task 1
- Task 1
- Machine
- Container
Task-Based Solution

Over-Utilization

Hide the overhead of a container restart
Task-Based Solution

Under-Utilization

Machine

Task

Container

384 MB

128 MB
Task-Based Solution

Under-Utilization

Machine

Task

Container

Task 1

Task 2
128 MB
Task-Based Solution
Under-Utilization

Optimize container allocations based on usage
Task-Based Solution

Failure

Machine

Container

Task 1
Leader

Task 2
Standby

Task 3
Standby

Task 2
Leader

Task 3
Standby

Task 1
Standby

Task 3
Leader

Task 1
Standby

Task 2
Standby
Task-Based Solution

Failure

![Diagram showing task-based solution with Task 1, Task 2, and Task 3 in different states (Leader and Standby).]
Task-Based Solution

Failure

Some systems cannot wait for new containers to start
Task-Based Solution

Discovery

N1

Task 1
Leader

Task 2
Standby

N2

Task 2
Leader

Task 1
Standby

Task 1:
Leader at N1
Standby at N2

Task 2:
Leader at N2
Standby at N1
Task-Based Solution

Discovery

Task 1:
- Leader at N1
- Standby at N2

Task 2:
- Leader at N2
- Standby at N1

Learn where everything runs, and what state each task is in.
Task-Based Solution

Scaling

Machine

Task

Container

T1

T2

T3

T4

T5

T6
Task-Based Solution

Scaling

Machine

Task

Container

T1

T2

T3

T4

T5

T6
Task-Based Solution

Scaling

Machine

Task

Container

T1
T2

T4
T5

T3
T6
Task-Based Solution

Scaling

Machine

Task

Container

T1

T2

T4

T5

T3

T6
## Comparing Solutions

<table>
<thead>
<tr>
<th></th>
<th>Container Solution</th>
<th>Task + Container Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Utilization</strong></td>
<td>Application requirements define container size</td>
<td>Tasks are distributed as needed to a minimal container set as per SLA</td>
</tr>
<tr>
<td><strong>Fault Tolerance</strong></td>
<td>New container is started</td>
<td>Existing task can assume a new state while waiting for new container</td>
</tr>
<tr>
<td><strong>Scaling</strong></td>
<td>Workload is repartitioned and new containers are brought up</td>
<td>Tasks are moved across containers</td>
</tr>
<tr>
<td><strong>Discovery</strong></td>
<td>Existence</td>
<td>Existence and state</td>
</tr>
</tbody>
</table>
Comparing Solutions
Benefits of a Task-Based Solution

Container reuse
Minimize overhead of container relaunch
Fine-grained scheduling
Comparing Solutions
Benefits of a Task-Based Solution

Container reuse
Minimize overhead of container relaunch
Fine-grained scheduling

Task : Container :: Thread : Process
Task is the right level of abstraction
Comparing Solutions

We need a reactive approach to resource assignment.

Working at task granularity is powerful.
Comparing Solutions

We need a reactive approach to resource assignment.

Working at task granularity is powerful.

How can Helix help?
Comparing Solutions

We need a **reactive** approach to resource assignment

Working at **task granularity** is powerful

How can **Helix** help?

---

YARN/Mesos: containers bring flexibility in a machine

Helix: tasks bring flexibility in a container
Task Management with Helix
Application Lifecycle

Capacity Planning
- Allocating physical resources for your load

Provisioning
- Deploying and launching tasks

Fault Tolerance
- Staying available, ensuring success

State Management
- Determining what code should be running and where
Helix Overview

Cluster Roles

- Controller
- NODES (Participants)
- Spectators

Tasks

Manage

Helix Overview

Cluster Roles

- Controller
- NODES (Participants)
- Spectators

Tasks
Helix Controller
High-Level Overview

- “single master”
- “no more than 3 tasks per machine”

Constraints

Nodes

Rebalancer

Task Assignment
Based on the current nodes in the cluster and constraints, find an assignment of task to node
ResourceAssignment computeResourceMapping(
    RebalancerConfig rebalancerConfig,
    ResourceAssignment prevAssignment,
    Cluster cluster,
    ResourceCurrentState currentState);

Based on the current nodes in the cluster and constraints, find an assignment of task to node

What else do we need?
Helix Controller

What is Missing?

- Dynamic Container Allocation
- Automated Service Deployment
- Container Isolation
- Resource Utilization Monitoring
Helix Controller

Based on some constraints, determine how many containers are required in this system.

We’re working on integrating with monitoring systems in order to query for usage information.
Helix Controller

Target Provider

```java
TargetProviderResponse evaluateExistingContainers(
        Cluster cluster,
        ResourceId resourceId,
        Collection<Participant> participants);
```

class TargetProviderResponse {
    List<ContainerSpec> containersToAcquire;
    List<Participant> containersToRelease;
    List<Participant> containersToStop;
    List<Participant> containersToStart;
}

Based on some constraints, determine how many containers are required in this system.

We’re working on integrating with monitoring systems in order to query for usage information.
Helix Controller

Adding a Target Provider

Target Provider

Constraints

Nodes

Rebalancer

Task Assignment
Helix Controller
Adding a Target Provider

Target Provider ➔ Constraints ➔ Nodes ➔ Rebalancer ➔ Task Assignment

How do we use the target provider response?
Helix Controller
Container Provider

Given the container requirements, ensure that number of containers are running
Given the container requirements, ensure that number of containers are running
Helix Controller

Adding a Container Provider

Target Provider

Container Provider

Constraints

Nodes

Rebalancer

Task Assignment

Target Provider + Container Provider = Provisioner
Application Lifecycle

With Helix and the Task Abstraction

- Capacity Planning
  - Target Provider

- Provisioning
  - Container Provider

- Fault Tolerance
  - Existing Helix Controller (enhanced by Provisioner)

- State Management
  - Existing Helix Controller (enhanced by Provisioner)
System Architecture
System Architecture

Resource Provider
System Architecture

Client -> submit job -> Resource Provider
System Architecture

Client → submit job → Resource Provider

Controller Container
- App Launcher
- Provisioner
- Rebalancer
System Architecture

Client -> submit job -> Resource Provider

Controller Container
- App Launcher
- Provisioner
- Rebalancer

container request
System Architecture

Client → submit job → Resource Provider

Controller Container:
- App Launcher
- Provisioner
- Rebalancer

Participant Container:
- Participant Launcher
- Helix Participant
- App

container request → assign tasks
Helix + YARN
YARN Architecture

Client → submit job → Resource Manager

Client

Node Manager

Application Master

Container

Node Manager

App Package

HDFS/Common Area

container request

assign work

status

node status

node status

grab package
Helix + YARN

Helix + YARN Architecture

Client → Resource Manager

Node Manager
- Helix Controller
- Rebalancer
Application Master

Node Manager
- Helix Participant
- App
Container

Assign tasks

Status

Submit job

Node status

Container request

App Package

HDFS/Common Area
Helix + Mesos

Mesos Architecture

Scheduler

offer resources

offer response

node status

Mesos Master

node status

Mesos Slave

Mesos Executor

Slave Machine

grab executor

Executor Package

HDFS/Common Area
Helix + Mesos

Helix + Mesos Architecture

Scheduler Slave

Scheduler

Helix Controller

node status

node status

offer resources

offer response

assign tasks

Mesos Slave

Mesos Executor

Helix Participant/App

Slave Machine

grab executor

Helix Executor Package

HDFS/Common Area

Master

Slave Machine

Slave Machine
Example
Distributed Document Store

Overview

Partition 0
Partition 1
Partition 2

P2 Backup
ETL

Partition 0
Partition 1
Partition 2

P1 Backup
ETL

Partition 0
Partition 1
Partition 2

P0 Backup
ETL

HDFS

Master Slave
Distributed Document Store

Overview

Partition 0
Partition 1
Partition 2

P2 Backup
ETL

Partition 0
Partition 1
Partition 2

P1 Backup
P0 Backup
ETL

HDFS

Master
Slave

HELIX
Distributed Document Store

YAML Specification

```yaml
appConfig: {  config: { k1: v1 } }
appPackageUri: 'file://path/to/myApp-pkg.tar'
appName: myApp
services: [DB, ETL] # the task containers
serviceConfigMap:
  {DB: { num_containers: 3, memory: 1024 }, ...
   ETL: { time_to_complete: 5h, ... }, ...
}
servicePackageURIMap: {
  DB: 'file://path/to/db-service-pkg.tar', ...
}
...
```
appConfig: { config: { k1: v1 } }
appPackageUri: 'file://path/to/myApp-pkg.tar'
appAppName: myApp
services: [DB, ETL] # the task containers
serviceConfigMap:
  {DB: { num_containers: 3, memory: 1024 }, ...
    ETL: { time_to_complete: 5h, ... }, ...}
servicePackageURIMap: {
  DB: 'file://path/to/db-service-pkg.tar', ...
}
...
public class MyQueuerService
    extends StatelessParticipantService {
    @Override
    public void init() { ... }

    @Override
    public void onOnline() { ... }

    @Override
    public void onOffline() { ... }
}
public class BackupTask extends Task {
    @Override
    public ListenableFuture<Status> start() { ... }

    @Override
    public ListenableFuture<Status> cancel() { ... }

    @Override
    public ListenableFuture<Status> pause() { ... }

    @Override
    public ListenableFuture<Status> resume() { ... }
}
Distributed Document Store

State Model-Style Callbacks

```java
public class StoreStateModel extends StateModel {
    public void onBecomeMasterFromSlave() { ... }

    public void onBecomeSlaveFromMaster() { ... }

    public void onBecomeSlaveFromOffline() { ... }

    public void onBecomeOfflineFromSlave() { ... }
}
```
class RoutingLogic {
    public void write(Request request) {
        partition = getPartition(request.key);
        List<Participant> nodes =
            routingTableProvider.getInstance(partition, "MASTER");
        nodes.get(0).write(request);
    }

    public void read(Request request) {
        partition = getPartition(request.key);
        List<Participant> nodes =
            routingTableProvider.getInstance(partition);
        random(nodes).read(request);
    }
}
Helix at LinkedIn
Helix at LinkedIn

In Production

Over 1000 instances covering over 30000 database partitions

Over 1000 instances for change capture consumers

As many as 500 instances in a single Helix cluster

(all numbers are per-datacenter)
Summary

- Container abstraction has become a huge win
- With Helix, we can go a step further and make tasks the unit of work
- With the TargetProvider and ContainerProvider abstractions, any popular provisioner can be plugged in
## Questions?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Jason</strong></td>
<td><a href="mailto:zzhang@apache.org">zzhang@apache.org</a></td>
</tr>
<tr>
<td><strong>Kanak</strong></td>
<td><a href="mailto:kanak@apache.org">kanak@apache.org</a></td>
</tr>
<tr>
<td><strong>Website</strong></td>
<td>helix.apache.org</td>
</tr>
<tr>
<td><strong>Dev Mailing List</strong></td>
<td><a href="mailto:dev@helix.apache.org">dev@helix.apache.org</a></td>
</tr>
<tr>
<td><strong>User Mailing List</strong></td>
<td><a href="mailto:user@helix.apache.org">user@helix.apache.org</a></td>
</tr>
<tr>
<td><strong>Twitter</strong></td>
<td>@apachehelix</td>
</tr>
</tbody>
</table>