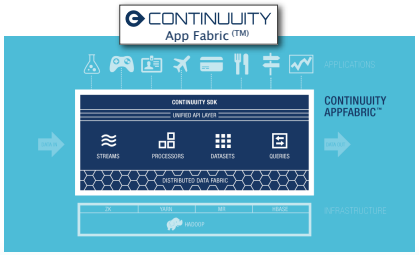
CONTINUITY

WEAVE - SENSIBLE YARN

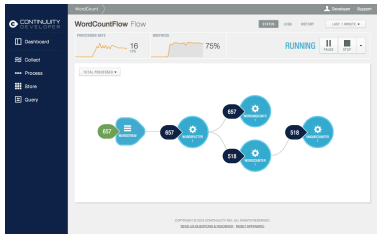
Nitin Motgi & Terence Yim

INTRODUCTION

- **Who we are ?**
  - Group of techies who have experienced Big Data in a variety of ways.
  - We understand the pain of building Big Data Applications - we have lived through it.
- **What do we do ?**
  - **Mission** : *Democratize the development of the next generation of Big Data applications.*
  - Building the industry's first Big Data Application Fabric(TM)
- **Products**
  - Developer Suite (public beta available now)
    - Downloadable developer edition. Includes SDK, tutorial, samples, docs, etc
  - Developer Sandbox
    - Free, Self-Service, hosted cloud environment
  - Virtual Private Cloud
    - Single tenant, private cloud PaaS. Fully Integrated BigData Application Platform
  - On-Premise



BIG FLOW



What is BigFlow?

- A core Processor; a real-time stream processing engine
  (Similar to Storm and S4)
- BigFlow provides
  - Tight integration with your existing Hadoop/HBase cluster
  - Exactly-once execution vs. At-least-once execution (your apps do not have to be idempotent)
  - Supports transactions for consistency
  - Direct integration with a data storage engine (Continuuity Data Fabric) and Datasets
  - Beautiful User interface for application management
  - Utilizes YARN for deployment and supports runtime elastic scalability of Flows

DEMO

CONTINUUITY


YARN IS NOT SIMPLE!

YARN PAIN POINTS

- YARN is not for novice Big Data programmers
  - High ramp-up time
- Lot of boiler plate code to build simple application
  - 80% of applications are generally simple
- YARN is built around applications that terminate
  - Logs are available only on completion of application
- No standard support for
  - Application lifecycle management
  - Communication between Container(s) & Application Master
  - Handling Application level errors


WEAVE!

MOTIVATION

- Running in YARN should be as simple as running Threads in Java
  - Not designed to be a replacement for YARN
- Designed to simplify building, debugging & running of YARN Applications
  - A simple programming model
- Hide messy details of YARN
  - Simplified API for specifying, running & managing an application
  - Simplified way to specify and manage stage(s) of a Application lifecycle
  - Generic Application Master to better support simple applications
  - Simpler archive management
  - Better control over application errors

SIMPLICITY

Distributed Shell - Vanilla YARN

~ 500 lines

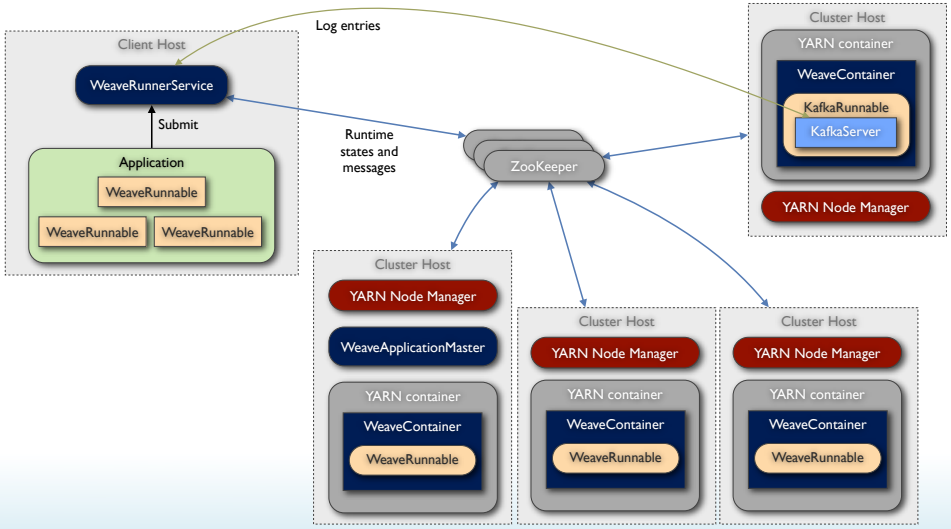
```
140. public class ApplicationMaster {
141.
 .
150.   private AMRMClientAsync resourceManager;
 .
435.    public boolean run() throws YarnRemoteException {
436.      LOG.info("Starting ApplicationMaster");
 .
498.    }
 .
500.    public void finish() {
504.      for(Thread launchThread: launchThreads) {
 .          ...
511.      }
517.      FinalApplicationStatus appStatus;
518.      String appMessage = null;
538.    }
 .
790.    public ContainerRequest setupContainerAskForRM(int numContainers) {
 .
800.      Resource capability = Records.newRecord(Resource.class);
801.      capability.setMemory(containerMemory);
 .
803.      ContainerRequest request = new ContainerRequest(capability, ..);
807.    }
808. }
 .
 .
1102.   public class Client extends YarnClientImpl {
 .
1336.     public boolean run() throws IOException {
 .
1354.       GetNewApplicationResponse newApp = super.getNewApplication();
1355.       ApplicationId appId = newApp.getApplicationId();
 .
1583.       return monitorApplication(appId);
1585.     }
 .
1594.     private boolean monitorApplication(ApplicationId appId) throws ... {
 .
1596.       while(true) {
 .
1606.         ApplicationReport report = super.getApplicationReport(appId);
 .
1647.       }
1648.     }
 .
1657.     private void forceKillApplication(ApplicationId appId) throws ... {
 .
1664.       super.killApplication(appId);
 .
1665.     }
 .
1667.   }
```

Distributed Shell - WEAVE

~ 50 lines

```
901. public class DistributedShell extends AbstractWeaveRunnable {
902.   static Logger LOG = LoggerFactory.getLogger(DistributedShell.class);
903.   private String command;
904.
905.   public DistributedShell(String command) {
906.     this.command = command;
907.   }
908.
909.   @Override
910.   public WeaveRunnableSpecification configure() {
911.     return WeaveRunnableSpecification.builder().with()
912.       .setName("Executor " + command)
913.       .withArguments(ImmutableMap.of("cmd", this.command))
914.       .build();
915.   }
916.
917.   @Override
918.   public void initialize(WeaveContext context) {
919.     this.command = context.getSpecification().getArguments("cmd");
920.   }
921.
922.   @Override
923.   public void run() {
924.     try {
925.       Process process = Runtime.getRuntime().exec(this.command);
926.       InputStream stdin = proc.getInputStream();
927.       BufferedReader br = new BufferedReader(new InputStreamReader(stdin));
928.
929.       String line;
930.       while( (line = br.readline()) != null) {
931.         LOG.info(line);
932.       }
933.     } catch (Throwable t) {
934.       LOG.error(t.getMessage());
935.     }
936.   }
937. }
938.
939. public static main(String[] args) {
940.   String zkStr = args[0];
941.   String command = args[1];
942.   yarnConfig = ...
943.   WeaveRunnerService weaveService = new YarnWeaveRunnerService(yarnConfig, zkStr);
944.   weaveService.startAndWait();
945.   WeaveController controller = weaveService.prepare(new DistributedShell(command))
946.    .addLogHandler(new PrinterLogHandler(new PrintWriter(System.out)))
947.    .start();
948. }
```

ARCHITECTURE



RUNNABLE

```
// Defines a Task
public class DistributedShell implements WeaveRunnable
{
  static Logger LOG = ...
  private String cmd;

  //....

  @Override
  public void run() {
    try {
      new ProcessBuilder(cmd).start().waitFor();
    } catch (Exception e) {
      LOG.error(e.getMessage(), e);
    }
  }
}
```

- Class implements WeaveRunnable is a single task.
- SLF4J logger can be used for logging
  - logs are collected and sent back to client using Kafka
  - An extra container is started to run Kafka server.
- WeaveRunnable can be run in
  - Thread
  - YARN Container
- Easy! - Piece of cake.

RUNNER

```
// Starts a task
WeaveRunner runner = ...;
WeaveController controller =
    runner.prepare(new DistributedShell("ls -al"))
          .addLogHandler(new PrinterLogHandler(
                          new PrintWriter(System.out)
                         )
          ).start();
//...
controller.sendCommand(Commands.create("flush"));
//...
controller.stop();
```

- WeaveRunner
  - Can run Runnable or Application
  - Packages the class and it's dependencies along with additional resources specified
  - Attaches log collector for collecting all the logs from containers
- WeaveController is used to manage the lifecycle of a runnable or application
- Controller also provides ability to send commands to running containers

APPLICATION

```
// Defines an Application
public class WebServer implements WeaveApplication {
  @Override
  public WeaveSpecification configure() {
    return WeaveSpecification.Builder.with()
      .setName("Jetty WebServer")
      .withRunnables()
         .add(new JettyWebServer())
         .withLocalFiles()
            .add("html-pages.tgz", pages, true)
            .apply()
         .add(new LogsCollector())
      .anyOrder()
      .build();
  }
}
```

- WeaveApplication defines a collection of runnables and their behavior
- Application is specified by a WeaveSpecification
- Application can specify any additional local directories to be made available for container to run.
- Weave starts containers in no order.

APPLICATION

```
// Defines Application with specific tasks ordering
public class DataApplication implements WeaveApplication {
  @Override
  public WeaveSpecification configure() {
    return WeaveSpecification.Builder.with()
      .setName("Cool Data Application")
      .withRunnables()
         .add("reader", new InputReader())
         .add("processor", new Processor())
         .add("writer", new OutputWriter())
      .order()
         .first("reader")
         .next("processor")
         .next("writer")
      .build();
  }
}
```

DEMO

CONTINUUITY

ROADMAP

- First release at the end of April
- Features for upcoming releases (in no order)
  - API support for managing lifecycle hooks of application & it's containers
  - Extendable Application Master
  - Support for command dispatching
  - Ability to programmatically control resource allocation
  - Advanced error handling capabilities
    - E.g. Fail application if 50% of my containers fail
    - E.g. Fail application if it take more 'x' seconds to allocated containers for application
  - Contribs
    - Running HBase using Weave
    - Running Jetty HTTP Server using Weave
  - Support for application level metrics
  - WeaveRunnerServer for managing multiple application
  - Better UI
  - Ability to remote debug a running container
  - LXC & QoS
  - Improve! APIs, Improve! APIs
  - . . .

THANK YOU

git@github.com:continuuity/weave.git