Running Orca for DC3b

The following explains how to run Orca for DC3b using a production level policy file. It assumes you are familiar with pipeline policy files.

Production Policy file

The following is an example of setting up a job office, four ISR pipelines and two CcdAssembly pipelines. Details of this production policy are described below.

```
#<?cfg paf policy ?>
#
# Orchestration Layer Policy for a DC3 production
#
shortName:           DRProduction
eventBrokerHost:     lsst8.ncsa.uiuc.edu
repositoryDirectory: /home/srp/working/pipeline_new
productionShutdownTopic:       productionShutdown

# if a special productionRunConfigurator class is needed, specify it
# here; otherwise, the default will be instantiated
# configurationClass:  lsst.ctrl.orca.BasicProductionRunConfigurator
#
# if there is special configuration information associated with the
# the class, it will go in here:
# configuration: {
#
# }

# this is production-level database configuration.  Note that there can
# be multiple database occurances.  The WorkflowConfigurator will create a
# DatabaseConfigurator for each entry.  Each must also be copied into the
# pipeline config file.
#
database: {

    name: dc3bGlobal

    system: {
        authInfo: {
            host: lsst10.ncsa.uiuc.edu
            port: 3306
        }

        runCleanup: {
            daysFirstNotice: 7  # days when first notice is sent before run can be deleted
            daysFinalNotice: 1  # days when final notice is sent before run can be deleted
        }
    }

    configurationClass: lsst.ctrl.orca.db.DC3Configurator
    configuration: {
        globalDbName: GlobalDB
        dcVersion: DC3b
        dcDbName: DC3b_DB
        minPercDiskSpaceReq: 10   # measured in percentages
        userRunLife: 2            # measured in weeks
    }
}
```

```
workflow: {
    shortName: MyWorkflow
    platform:  @platform/nfs/lsstcluster.paf
    shutdownTopic:       workflowShutdown

    configurationClass: lsst.ctrl.orca.GenericPipelineWorkflowConfigurator
    configuration: {
        deployData: {
            dataRepository: /lsst/DC3/data/obstest
            collection: CFHTLS/D2
            script: /home/srp/working/pipeline_new/deployData.sh
        }
        # launchScript:
    }

    # this is workflow-level database setup.  Note that there can
    # be multiple database occurances.  The PipelineConfigurator will create a
    # DatabaseConfigurator for each entry.  Each must also be copied into the
    # pipeline config file.
    #
    # database: {
    #
    # }

    pipeline: {
       shortName:     joboffices
       definition:    @joboffice.paf
       runCount: 1
       deploy: {
          # the schema for this node is determined by the above
          # configuration class.  In this case, we can assign exactly the
          # number of processes to run on each node.  The
          #

          # an assignment of processes to nodes.  The first entry will
          # be host the Pipeline process.  Note that while the name
          # is the same as in the IPSD workflow, the value is interpreted
          # differently by virtue of the fact that different
          # WorkflowConfigurator class is used.
          #
          processesOnNode: lsst8:1

       }
       launch: true
    }
    pipeline: {
       shortName:     isr
       definition:    @cfht-isr-master.paf
       runCount: 4
       deploy: {
          # the schema for this node is determined by the above
          # configuration class.  In this case, we can assign exactly the
          # number of processes to run on each node.  The
          #

          # an assignment of processes to nodes.  The first entry will
          # be host the Pipeline process.  Note that while the name
          # is the same as in the IPSD workflow, the value is interpreted
          # differently by virtue of the fact that different
          # WorkflowConfigurator class is used.
          #
          processesOnNode: lsst8:1
        }
        launch: true
    }
    pipeline: {
        shortName:     ccdassembly
        definition:    @cfht-ca-master.paf
        runCount: 2
        deploy: {
            # the schema for this node is determined by the above
            # configuration class.  In this case, we can assign exactly the
            # number of processes to run on each node.  The
            #

            # an assignment of processes to nodes.  The first entry will
            # be host the Pipeline process.  Note that while the name
            # is the same as in the IPSD workflow, the value is interpreted
            # differently by virtue of the fact that different
            # WorkflowConfigurator class is used.
            #
            processesOnNode: lsst8:1

        }
        launch: true
    }
}
```

The **configurationClass** entry names the Orca object that will be used to deploy the workflow. In this case, "lsst.ctrl.orca.GenericPipelineWorkflowConfigurator?" deploys to the LSST cluster, which is a set of machines with a shared NFS filesystem.

The **configuration** node contains a node called **deployData**. The purpose of this node is to name a script that will link data files from a repository into a production runs' directory area.

This node contains three entries. The dataRepository entry describes the root location of a data repository. The collection entry describes a collection subdirectory below that root directory. The script entry is the name of a shell script to execute. Orca will run this script after it sets up the working directories for the pipeline. It will execute it in the form of "script dataRepository collection". The script listed here is entirely data collection dependent, so you must either create your own script from scratch, or modify the deployData.sh which is provided in the examples directory of ctrl_orca.

The pipeline nodes within the workflow node describe each of the pipelines in a similar fashion to how they were described in DC3a. The pipeline node's definition file names the pipeline policy to use.

The pipelines are executed by orca in the order they are specified. If you are using a joboffice, you must have this as the first entry in your workflow. Here's that subsection, describing the joboffice pipeline:

```
    pipeline: {
        shortName:     joboffices
        definition:    @joboffice.paf
        runCount: 1
        deploy: {
            # the schema for this node is determined by the above
            # configuration class.  In this case, we can assign exactly the
            # number of processes to run on each node.  The
            #
        # an assignment of processes to nodes.  The first entry will
        # be host the Pipeline process.  Note that while the name
        # is the same as in the IPSD workflow, the value is interpreted
        # differently by virtue of the fact that different           # WorkflowConfigurator class is used
        #           processesOnNode: lsst8:1

     }
     launch: true
  }
```

This pipeline is interpreted as follows: A pipeline named "joboffices", using the definition joboffice.paf will be deployed to lsst8 and run once. The definition of joboffice.paf is:

```
#<?cfg paf policy ?>
framework: {
   type: standard
   environment: "$DATAREL_DIR/etc/setup.sh"
   exec: "$DATAREL_DIR/pipeline/joboffice.sh"
}

execute: {
   eventBrokerHost: "lsst8.ncsa.uiuc.edu"
   shutdownTopic: workflowShutdown
}
```

The environment setup for this is $DATAREL_DIR/etc/setup.sh. The script that is run is $DATAREL_DIR/pipeline/joboffice.sh. Orca currently runs this with the same command line options as all the other pipeline scripts, but for it's purposes only uses some of the arguments. Here is that script:

```

 the same location for launchPipeline.sh and are





' ]; then




RBOSITY -r $RUNID -b lsst8.ncsa.uiuc.edu -d $DC3ROOT/$RUNID/work $DATAREL_DIR/pipeline/CcdAssembly/cfht-ca-jobo
RBOSITY -r $RUNID -b lsst8.ncsa.uiuc.edu -d $DC3ROOT/$RUNID/work $DATAREL_DIR/pipeline/ISR/cfht-isr-joboffice.p

offices
```

This script runs two job offices, one for CcdAssembly, and one for ISR. If you want to run more joboffices, you need to make a copy of this script, modify it, and specify it in the framework.exec entry of the joboffice pipeline.

The next pipeline listed in the policy file is for isr:

```
   pipeline: {
      shortName:     isr
      definition:    @cfht-isr-master.paf
      runCount: 4
      deploy: {
         # the schema for this node is determined by the above
         # configuration class.  In this case, we can assign exactly the
         # number of processes to run on each node.  The
         #

         # an assignment of processes to nodes.  The first entry will
         # be host the Pipeline process.  Note that while the name
         # is the same as in the IPSD workflow, the value is interpreted
         # differently by virtue of the fact that different
         # WorkflowConfigurator class is used.
         #
         processesOnNode: lsst8:1

      }
      launch: true
   }
```

This pipeline is interpreted as follows: A pipeline named "isr", using the definition cfht-isr-master.paf will be run four times on lsst8.

The next pipeline listed in the policy file is for ccdassembly:

```
   pipeline: {
      shortName:     ccdassembly
      definition:    @cfht-ca-master.paf
      runCount: 2
      deploy: {
         # the schema for this node is determined by the above
         # configuration class.  In this case, we can assign exactly the
         # number of processes to run on each node.  The
         #

         # an assignment of processes to nodes.  The first entry will
         # be host the Pipeline process.  Note that while the name
         # is the same as in the IPSD workflow, the value is interpreted
         # differently by virtue of the fact that different
         # WorkflowConfigurator class is used.
         #
         processesOnNode: lsst8:1

      }
      launch: true
   }
```

This pipeline is interpreted as follows: A pipeline named "ccdassembly", using the definition cfht-ca-master.paf will be run two times on lsst8.

Please make a note of the entry:

```
platform:  @platform/nfs/lsstcluster.paf
```

in the workflow portion of the production policy. The **dir** section in that policy file is used to override all directory information in the pipeline policy. That section in this example looks like this:

```
dir: {
   shortName: "foo"
   # the default root directory all files read or written by pipelines
   # deployed on this platform.  This must be an absolute directory.
   # This can be overriden by any of the "named role" directories below.
   #
   defaultRoot:  /lsst/DC3root

   # the pattern to use for setting the root directory for a production
   # run.  The result is a directory relative to the default root directory
   # (set via defaultRoot).  The format is a python formatting string using the
   # following dictionary keys:
   #   runid     the unique identifier for the production run
   #
   runDirPattern:  "%(runid)s"

   # These indicate the directory that should be used for a named purpose.
   # If relative paths are given, the resulting directory will be relative
   # to the default run directory (determined by defaultRoot and the
   # runDirPattern).  These can be given as patterns specified in the same
   # format as runDirPattern.  (If a directory is given as an absolute path,
   # using a pattern is recommended in order to distinguish between different
   # production runs.)
   #
   work:     work    # the working directory, where the pipeline is started
   input:    input   # the directory to cache/find input data
   output:   output  # the directory to write output data
   update:   update  # the directory where updatable data is deployed
   scratch:  scr     # a directory for temporary files that may be deleted upon completion of the pipelin
}
```

So in this case, Orca will use this information to create a directory structure that looks like **<runid>/work**, **<runid>/input**, **<runid>/output**, **<runid>/update** and **<runid>/scr** in the directory **/lsst/DC3root**.

Running Orca

You can run Orca by issuing a command similar to the following. It's suggested to use this method to view logging output from Orca so that you can see what is happening.

```
$ orca.py -r $PWD -e ~/srp_stack.sh -V 10 -P 10 production.paf myrunid
```

The options used in this command line invocation are:

-r the location of the repository to use for policy files. **-e** a Bourne shell script which is used to setup the software environments for the pipelines. **-V** the debug level for Orca log messages **-P** the debug level for pipeline log messages

The two other parameters in the line above **production.paf** and **myrunid** are the production level policy and the runid for this run, respectively. The runid must be different for each run. I suggest using a combination of your initials, the current date and the number of the run you've done that day. (ie, xyz051501) That way it can help you look at the run directories themselves to determine which one you're looking for, without having to go into each individual directory and editing files. When you run this, you'll see a lot of logging messages from orca. Depending on how many pipelines you launch, this can take a while. When orca starts up all the pipelines, you'll see messages that look similar to:

```
          orca.manager.config.workflow DEBUG: GenericPipelineWorkflowLauncher:__init__
```

```
        orca.manager.launch DEBUG: StatusListener:__init__
        orca.manager.config.workflow DEBUG: WorkflowManager:runWorkflow
        orca.manager.config.workflow DEBUG: WorkflowManager:isRunnable
        orca.manager.config.workflow DEBUG: WorkflowManager:isDone
        orca.manager.config.workflow DEBUG: GenericPipelineWorkflowLauncher:launch
        orca.manager.config.workflow.monitor DEBUG: GenericPipelineWorkflowMonitor:__init__
        orca.manager.config.workflow.monitor DEBUG: WorkflowMonitor:addStatusListener
        orca.manager.config.workflow.monitor DEBUG: GenericPipelineWorkflowMonitor Thread started
        orca.manager DEBUG: listening for shutdown event at 0.2 s intervals
                  orca.manager DEBUG: checking for shutdown event
        orca.manager DEBUG: self._timeout = 10
```

Look in the /lsst/DC3root/<runid> directory to see the directories it created. In this case, you'll see:

```
input output scr update work
```

The input directory will have all the links created using the deployData.sh script described earlier.

```
$ ls input
bias  calibRegistry.sqlite3  D2  dark  flat  registry.sqlite3
$
```

The work directory contains the pipeline directories, the policy files (and their directories) and the setup script.

```
ls work
CcdAssembly    CcdAssembly-joboffice  ISR    isr_3          joboffice.paf
ccdassembly_1  cfht-ca-master.paf     isr_1  isr_4          joboffices_1
ccdassembly_2  cfht-isr-master.paf    isr_2  ISR-joboffice  srp_stack.sh
```

Look in the directories isr_1, isr_2, isr_3, isr_4, ccdassembly_1, and ccdassembly_2 for pipeline log files.

If this is the first time you're running, you should look at those log files to make sure that everything is running the way you expect.

If you don't have Orca run the announceDataset.py command, it will issue the following:

> "No data announced for PT1Workflow. Waiting for events from external source"

If you see this, execute the announceDataset.py command (which is in ctrl_sched):

```
announceDataset.py -r myrunid -b lsst8.ncsa.uiuc.edu -t RawAvailable cfht-isr-inputdata.txt
```

This will send events to the event broker on lsst8 on topic "RawAvailable?". These events are listened to by the joboffice processes, which are only listening for events that have the runid **myrunid** associated with it, and will start the Pipelines.

Alternatively, you can have Orca run this command for you, by adding the following section to the workflow.configuration section of the PAF file:

The format is like this:

```
        announceData: {
            script: $CTRL_SCHED_DIR/bin/announceDataset.py
            topic: RawAvailable
            inputdata: /home/srp/working/pipeline_new/ISR/cfht-isr-inputdata.txt
        }
```

At this point the data will be announced, and the pipelines will run.

To stop the pipelines, use the shutprod.py command (in ctrl_orca) with a severity level and the runid to shut down.

```
shutprod.py 1 <runid>
```

This sends an event to orca, which coordinates sending messages to each workflow to shut things down.

You'll see messages similar to the following:

```
orca.manager DEBUG: DONE!
orca.manager: Shutting down production (urgency=1)
          orca.manager.config.workflow DEBUG: WorkflowManager:stopWorkflow
          orca.manager.config.workflow.monitor DEBUG: GenericPipelineWorkflowMonitor:stopWorkflow
          orca.manager.config.workflow.monitor DEBUG: GenericPipelineWorkflowMonitor:handleEventCalled
          orca.manager DEBUG: Everything shutdown - All finished
```