Packaging Simplification

- Motivation
- Background
  - Dissecting the .zip package
    - metainfo.xml
    - optional files (appConfig*.json and resources*.json)
    - configuration folder
    - package folder
      - files
      - templates
      - scripts
- Proposal
  - Scenarios
    - Scenario 1: Simple command invocation
      - Using slider
    - Scenario 2: Simple invocation with configurations
      - Using slider
      - Accessing supplied config (appConfig.json)
      - Scenario 2.a: App can advertise allocated hosts and ports
    - Scenario 3: Simple invocation with configurations against supplied libraries
      - Using slider
- Details of Changes

Motivation

- Do not mandate the need of an app package .zip file
- Allow simpler specification for applications not requiring much setup and configuration (e.g. run a command, run a java program with few inline configs and couple of jars)
- Align *with* docker requirements for configs and commands

Background

Creating an instance of a Slider app, today, requires at least three items

1. application package .zip file
2. application instance configuration (appConfig.json)
3. resource configuration for the app instance (resources.json)

Details of an application package can be found at http://slider.incubator.apache.org/docs/slider_specs/index.html.

Dissecting the .zip package

Lets dissect the application package. The most critical file is metainfo.xml - in fact its the only critical file. All other files are optional and the structure is a layout that is useful for the scripts embedded in the package.

metainfo.xml

This is a mandatory file describing the metadata of the application, its components, exports if any, command execution orders if any, location of the binaries, and for each component the “script” that can be invoked for lifecycle operations.

optional files (appConfig*.json and resources*.json)

These files essentially act as sample input files.

configuration folder

Default configuration associated with the application.

package folder

package folder includes all other files including the binaries for the application. It has three sub folders.

files

May contain any file including folder with jars, a tarball or .zip file, etc

templates

Contains templates that may be used to create app specific files

scripts

Contains scripts that can be used to execute any command.

Proposal

Enhancements being proposed are:

- An application package should not be required
  - metainfo.xml can be an input (migrate to metainfo.json)
- A structured local folder on the client machine may be specified in lieu of the application package (think of the folder as an expanded .zip file)
  - --application.def /usr/work/my_hbase_def
- --template will be an optional parameter
  - If no config is required then no need to supply one
- [Stretch goal] metainfo can be auto constructed from parameters provided in the command line

Lets examine the proposal against some scenarios.

Scenarios

Scenario 1: Simple command invocation

Start a simple java process where target hosts (where YARN can allocate containers) have all requirements pre-installed at known locations. No configuration is expected from the user.

@command prompt:

```
java -classpath /usr/myapps/memcached/*:/usr/lib/hadoop/lib/*
com.thimbleware.jmemcached.Main --memory=256
```

Using slider

```
slider create -name cl1 --metainfo metainfo.json --resources resources.json,
```

where

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "2.1",
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "commands": [
                        {
                            "command": {
                                "exec": "java -classpath
/usr/myapps/memcached/*:/usr/lib/hadoop/lib/* com.thimbleware.jmemcached.Main --memory=256"
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

resources.json

```
{
 "schema" : "http://example.org/specification/v2.0.0",
 "components": {
   "MEMCACHED": {
     "yarn.role.priority": "1",
     "yarn.component.instances": "1",
     "yarn.memory": "256"
   }
 }
}
```

Scenario 2: Simple invocation with configurations

Start a simple java process where target hosts (where YARN can allocate containers) have all requirements pre-installed at known location. User wants provide certain configurations.

@command prompt:

```
java -Xmx 256 -Xms 128 -classpath /usr/myapps/memcached/*:/usr/lib/hadoop/lib/*
com.thimbleware.jmemcached.Main --memory=256 --port=11222
```

Using slider

`slider create -name cl1 --metainfo metainfo.json --resources resources.json --template appConfig.json`, where

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "2.1",
        "application": {
            "name": "MEMCACHED",
            "components": [
                {
                    "component": {
                        "name": "MEMCACHED",
                        "commands": [
                            {
                                "command": {
                                    "exec": "java -Xmx{$conf:@//site/global/xmx_val}
-Xms{$conf:@//site/global/xms_val} -classpath /usr/myapps/memcached/*:/usr/lib/hadoop/lib/*
com.thimbleware.jmemcached.Main --memory={$conf:@//site/global/memory_val}
--port={$conf:@//site/global/port}"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
```

Accessing supplied config (appConfig.json)

{$conf:@//site/<type>/<variable name>} may be used to refer to a supplied config variable.

Similarly, {$env:<environment variable name>} can be used to refer to an environment variable on the target host.

appConfig.json

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
    "site.global.xmx_val": "256m",
    "site.global.xms_val": "128m",
    "site.global.memory_val": "200M",
    "site.global.port": "${MEMCACHED.ALLOCATED_PORT}{PER_CONTAINER}"
  }
}
```

resources.json (same as above)

```
{
 "schema" : "http://example.org/specification/v2.0.0",
 "components": {
   "MEMCACHED": {
     "yarn.role.priority": "1",
     "yarn.component.instances": "1",
     "yarn.memory": "256"
   }
 }
}
```

Scenario 2.a: App can advertise allocated hosts and ports

The app can choose to advertise allocated hosts and ports by adding the spec into the metainfo.json file.

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "2.1",
        "exportGroups": {
            "exportGroup": {
                "name": "Servers",
                "exports": [
                    {
                        "export": {
                            "name": "host_port",
                            "value": "${MEMCACHED_HOST}:${site.global.port}"
                        }
                    }
                ]
            }
        },
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "compExports": "Servers-host_port",
                    "commands": [
                        {
                            "command": {
                                "exec": "java -Xmx{$conf:@//site/global/xmx_val}
-Xms{$conf:@//site/global/xms_val} -classpath /usr/myapps/memcached/*:/usr/lib/hadoop/lib/*
com.thimbleware.jmemcached.Main --memory={$conf:@//site/global/memory_val}
--port={$conf:@//site/global/port}"
                            }
                        }
                    ]
                }
            }
        }
    }
}
```

Scenario 3: Simple invocation with configurations against supplied libraries

Start a simple java process where target hosts (where YARN can allocate containers) have all requirements pre-installed at known location. Allow user to pass configurations as well as jar files.

Command prompt:

```
java -Xmx 256 -Xms 128 -classpath
/location_of_supplied_jars/*:/usr/lib/hadoop/lib/*
com.thimbleware.jmemcached.Main --memory=256 --port=11222
```

Using slider

slider create -name cl1 -- resources resources.json --template appConfig.json
--application.def /usr/work/memcached

```
Contents of /usr/work/memcached
./metainfo.xml
./package
./package/files
./package/files/jmemcached-1.0.0
./package/files/jmemcached-1.0.0/jmemcached-cli-1.0.0.jar
./package/files/jmemcached-1.0.0/jmemcached-core-1.0.0.jar
```

metainfo.json in the folder

```
{
    "metainfo": {
        "schemaVersion": "2.1",
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "commands": [
                        {
                            "command": {
                                "exec": "java -Xmx {$conf:@//site/global/xmx_val} -Xms
{$conf:@//site/global/xms_val} -classpath
{$conf:@//site/global/app_root}/*:/usr/lib/hadoop/lib/* com.thimbleware.jmemcached.Main
--memory={$conf:@//site/global/memory_val} --port={$conf:@//site/global/port}"
                            }
                        }
                    ]
                }
            },
            "packages": [
                {
                    "package": {
                        "type": "folder",
                        "name": "files/jmemcached-1.0.0"
                    }
                }
            ]
        }
    }
}
```

appConfig.json (same as above)

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
    "site.global.xmx_val": "256m",
    "site.global.xms_val": "128m",
    "site.global.memory_val": "200M",
    "site.global.port": "${MEMCACHED.ALLOCATED_PORT}{PER_CONTAINER}"
  }
}
```

resources.json (same as above)

```
{
 "schema" : "http://example.org/specification/v2.0.0",
 "components": {
   "MEMCACHED": {
     "yarn.role.priority": "1",
     "yarn.component.instances": "1",
     "yarn.memory": "256"
   }
 }
}
```

Details of Changes

The changes are limited to the metainfo.xml/json file to allow specification of simple application in-line.

```
{
    "metainfo": {
        "schemaVersion": "2.1",
        "application": {
            "name": "MEMCACHED",
            "components": [
                {
                    "component": {
                        "name": "MEMCACHED",
                        "commands": [
                            {
                                "command": {
                                    "exec": "java -Xmx {$conf:@//site/global/xmx_val} -Xms {$conf:@//site/global/xms_val} -classpath {$conf:configurations/global/app_root}/*:/usr/lib/hadoop/lib/* com.thimbleware.jmemcached.Main --memory={$conf:@//site/global/memory_val} --port={$conf:@//site/global/port}"
                                }
                            }
                        ]
                    }
                }
            ],
            "packages": [
                {
                    "package": {
                        "type": "folder",
                        "source": "files/jmemcached-1.0.0"
                    }
                }
            ]
        }
    }
}
```

Components can explicitly specify commands. The default command name is “START” and the default type is “SHELL” - see 12-14

A fully specified command section provides the command to execute, type of command, and the name of the command. “name” can be START/STOP/INSTALL etc. If the command is not a command understood by Slider (CONFIGURE, INSTALL, START, STOP, STATUS_CHECK) then it is considered a custom command. *Custom command execution is not yet supported in Slider*.

Slider inherently supports “PYTHON” scripts as command implementation. It will also support execution of arbitrary shell commands. *Docker commands will also be supported as a category to allow docker specific command support*.

```
"commands": [
  {
    "command": {
      "exec": "script for configure",
      "name": "CONFIGURE",
      "type": "PYTHON",
    }
  },
  {
    "command": {
      "exec": "command for Start",
      "name": "START",
      "type": "SHELL",
    }
  }
]
```

The application may specify packages which can be of type tarball, zip, or plain folders - see 22-26

Default install implementation deploys all packages in the app_root of the container. The deployed files can be referred to by INSTALL or CONFIGURE commands as needed.

```
"packages": [
                {
                    "package": {
                        "type": "folder",
                        "source": "files/jmemcached-1.0.0"
                    }
                },
                {
                    "package": {
                        "type": "tar",
                        "source": "URL for tarball",
                        "dest": "relative path from ${app_root}",
                    }
                }
            ]
```

A package is a tarball, zip file, or folder on the client host. The source path is relative to the location of the metainfo.json file. The source can be a URL from where a tarball or a zip file can be downloaded from. The packages are automatically copied into HDFS and localized at the container in a default location or a location specified via “dest”.