Motivation

- Enable YARN and Slider to have the capability to maintain applications defined as Docker image and to reuse a vast number of applications already packaged as Docker image
- Adopt the advantages of Docker to provide better isolation, packaging, and configurability to applications running on YARN and Slider

Goal

- Slider shall be able to deploy an application defined as Docker image, monitor its running status, fetching exported configs, and maintain its lifecycle
- Slider shall be able to pull Docker images from Docker Hubs
- Slider shall be able to aggregate application logs from and feed in input files into Docker containers
- Slider shall be able to launch applications containing multiple Docker images, as well as maintaining multiple Docker containers on the same physical host
- Users shall be able to specify the Docker images, configurations and if necessary, the special docker command to start/query/stop the container in Slider configuration files
- Slider shall be able to deploy security credentials into the container if the application is running in secure mode

Assumptions

The scope of this JIRA doesn't yet include:

1) application Docker containers integrated with Ambari client
2) applications deployed in secure mode
3) Running application Docker containers in OS other than Linux
4) Docker images stored in private Docker hub -- we would leave this work to administrators to configure Docker daemon to be able to pull from those hubs
5) application Docker containers started with 'docker run -i' option which requires interaction with the user
6) Docker images that require multiple initialization steps, more than just one 'Docker run'
7) YARN can only monitor resources consumed by SliderAgent, instead of that of the Docker containers

Sample Slider commands and configs

Scenario 1: Deploy a minimal Docker based application

In this scenario, we are deploying an application defined as an Docker image with minimal configurations. The application doesn't expose any port and use the default command path

‘/usr/bin/docker’ for docker commands. It also assumes running ‘docker run’ with ‘-d’ option. Given the simplicity of the configs required, no appConfig.json need to be provided and specified on ‘slider create’ command

Commands to run:
Slider create [app-name] --metainfo metainfo.json --resources resources.json

Configuration files:
metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "1.0",
        "exportGroups": {
            "exportGroup": {
                "name": "Servers",
                "exports": [{
                    "export": {
                        "name": "memcached_servers",
                        "value": "${MEMCACHED_HOST}:${site.global.listen_port}"
                    }
```

```
                }]
            }
        },
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "compExports": "memcached_servers",
                    "type": “docker”,
                    “containers”: {
                        “container”: {
                            “name”: “memcached”,
                            “image”: “borja/memcached”
                        }
                    }
                }
            }
        }
    }
}
```

Please note in metainfo.json (we are trying to migrate from metainfo.xml to metainfo.json as an improvement to the packaging approach), we are adding some new fields in the component section to support Docker based applications:

- type: if specified as ‘docker’, Slider will start Docker containers for the application; by default it is ‘process’, which means Slider will instantiate the application as normal process as today
- containers/container: the Docker image of the application, the name of which will be specified in ‘image’ field

resources.json

```
{
  "schema" : "http://example.org/specification/v2.0.0",
  "metadata" : {
  },
  "global" : {
  },
  "components": {
    "slider-appmaster": {
    },
    "MEMCACHED": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    }
  }
}
```

With the configuration files above, Slider will ask for the number of YARN containers from YARN as specified in resources.json. In each of those YARN containers, Slider will run 'docker pull' to download the Docker image specified in metainfo.json. In the scope of this JIRA ticket, we are not supporting Docker images stored in private Docker hubs that require credentials to run docker pull. After downloading completes, Slider will start the containers by running, in this case, '/usr/bin/docker run -d borja/memcached'

Scenario 2: Deploy simple Docker based application

In this scenario, we are deploying a simple application defined as an Docker image. It is a group of memcached servers that are complete in binary and configuration. The application doesn't require parameters passed into the container when it gets started; neither does options passed into 'docker run' command when the container gets started. The user just need to specify the Docker image name of the application in metainfo.json and the absolute path to the Docker command to be invoked in appConfig.json

Commands to run:

Slider create [app-name] --template appConfig.json --metainfo metainfo.json --resources resources.json

Configuration files:

appConfig.json

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
    “site.docker.docker.command_path”:”/usr/dockerUser/bin/docker”,
    "site.docker.docker_run_command.options":"-d",
    “site.global.listen_port”: “${MEMCACHED_PORT}”
  },
  "components": {
  }
}
```

Please note that in this example, we are specifying a different docker command path than the default ‘/usr/bin/docker’ in appConfig.json; we also specify the options that we need to include in the ‘docker run’ command(‘-d’ is by defauled included. this is just a demo as how to include docker run command options); we also need to expose a port to listen on for the application. Thus, we are specifying ‘site.global.listen_port’ in appConfig.json and the export config in metainfo.json below

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "1.0",
        "exportGroups": {
            "exportGroup": {
                "name": "Servers",
                "exports": [{
                    "export": {
                        "name": "memcached_servers",
                        "value": "${MEMCACHED_HOST}:${site.global.listen_port}"
                    }
                }]
            }
        },
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "compExports": "memcached_servers",
                    "type": “docker”,
                    “containers”: {
                        “container”: {
                            “name”: “memcached”,
```

```
                        “image”: “borja/memcached”
                    }
                }
            }
        }
    }
}
```

resources.json

```
{
  "schema" : "http://example.org/specification/v2.0.0",
  "metadata" : {
  },
  "global" : {
  },
  "components": {
    "slider-appmaster": {
    },
    "MEMCACHED": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    }
  }
}
```

Scenario 3: Deploy Docker based application that requires rich configuration

In this scenario, we are deploying an application that requires quite a few configurations. We are faking an application here that are composed of both redis servers and memcached servers, so multiple components are specified in metainfo.json. Also, we specify the parameters that we need to start the containers in the component sections in metainfo.json as well.

Commands to run:

Slider create [app-name] --template appConfig.json --metainfo metainfo.json --resources resources.json

Configuration files:

appConfig.json

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
```

```
  },
  "global": {
    “site.docker.docker.command_path”:”/usr/bin/docker”,
    "site.docker.options":"-d",
    “site.global.listen_port”: “${MEMCACHED_PORT}”
  },
  "components": {
      “MEMCACHED: {
          “start_command” : “memcached_server.sh”,
          “additional_param” : “--appendonly yes”
    }
  }
}
```

Please note we are adding a few configurations for the component ‘memcached’ in appConfig.json here. Those configurations, as opposed to the configurations in metainfo.json, are run time parameters of the component, instead of part of the Docker image definition as in the case of metainfo.json.

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "1.0",
        "exportGroups": {
            "exportGroup": {
                "name": "Servers",
                "exports": [{
                    "export": {
                        "name": "memcached_servers",
                        "value": "${MEMCACHED_HOST}:${site.global.listen_port}"
                    }
                }]
            }
        },
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "compExports": "memcached_servers",
                    "type": “docker”,
                    “containers”: [{
                        “container”: {
                            “name”: “memcached”,
                            “image”: “borja/memcached”
                        }
                    }]
                },
                "component": {
                    "name": "REDIS",
                    "compExports": "redis_servers",
                    "type": “docker”,
                    “containers”: [{
```

```
                        "container": {
                            "name": "redis",
                            "image": "dockerhub/redis",
                            "ports": [{
                                "port": {
                                    "containerPort" : "11211",
                                    "hostPort": "${conf:configuration/global/port1}"
                                }
                            }],
                            "mounts": [{
                                "mount": {
                                    "containerMount": "/tmp/confg",
                                    "hostMount":
"${conf:configuration/global/app_root}/conf"
                                }
                            }],
                            "options": "-net=bridge"
                        }
                    }]
                }

            }
        }
    }
}
```

In container section, there are some fields in metainfo.json to notice:

- port, containerPort and hostPort: the port of the container that will be binded to the hostPort field, which will be translated into '-p hostPort:containerPort' when starting the container with 'docker run'
- mount, containerMount and hostMount: the directories from the host that will be mounted into the container, which will be translated into '-v hostMount:containerMount' when starting the container
- options: allow users to specify any additional docker run command options to use

resources.json

```
{
  "schema" : "http://example.org/specification/v2.0.0",
  "metadata" : {
  },
  "global" : {
  },
  "components": {
    "slider-appmaster": {
    },
    "REDIS": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    },
```

```
    "MEMCACHED": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    }

  }
}
```

With the configuration files above, Slider will pull the image the same way as in Scenario 1.
For redis components, it will run:
docker run -d -p hostPort:containerPort -v hostMount:containerMount -net=bridge dockerhub/redis
For memcached components, it will run:
docker run -d borja/memcached memcached_server.sh --appendonly yet

Scenario 4: Deploy Docker based application with custom Docker commands

In this scenario, the application and its configurations are the same as the one in Scenario 2, except we can specify a customer command to query the status of the containers of component 'memcached'

Commands to run:
Slider create [app-name] --template appConfig.json --metainfo metainfo.json --resources resources.json

Configuration files:
appConfig.json

```
{
  "schema": "http://example.org/specification/v2.0.0",
  "metadata": {
  },
  "global": {
    "site.docker.docker.command_path":"/usr/bin/docker",
    "site.docker.options":"-d",
    "site.global.listen_port": "${MEMCACHED_PORT}"
  },
  "components": {
       "MEMCACHED: {
           "start_command" : "memcached_server.sh",
           "additional_param" : "--appendonly yes"
    }
  }
}
```

metainfo.json

```
{
    "metainfo": {
        "schemaVersion": "1.0",
        "exportGroups": {
            "exportGroup": {
                "name": "Servers",
                "exports": [{
                    "export": {
                        "name": "memcached_servers",
                        "value": "${MEMCACHED_HOST}:${site.global.listen_port}"
                    }
                }]
            }
        },
        "application": {
            "name": "MEMCACHED",
            "components": {
                "component": {
                    "name": "MEMCACHED",
                    "compExports": "memcached_servers",
                    "type": “docker”,
                    “containers”: [{
                        “container”: {
                            “name”: “memcached”,
                            “image”: “borja/memcached”,
                            “status_command”: “docker inspect ${container_id}"
                        }
                    }]
                },
                "component": {
                    "name": "REDIS",
                    "compExports": "redis_servers",
                    "type": “docker”,
                    “containers”: [{
                        “container”: {
                            “name”: “redis”,
                            “image”: “dockerhub/redis”,
                            “ports”: [{
                                “port”: {
                                    “containerPort” : “11211”,
                                    “hostPort”: “${conf:configuration/global/port1}”
                                }
                            }],
                            “mounts”: [{
                                “mount”: {
                                    “containerMount”: “/tmp/confg”,
                                    “hostMount”:
“${conf:configuration/global/app_root}/conf”
                                }
                            }]
                        }
                    }]
                }

            }
        }
    }
}
```

In container section, there are some fields in metainfo.json to notice:

- status_command: the command that Slider can use to query the status of the application component running in the container

resources.json

```
{
  "schema" : "http://example.org/specification/v2.0.0",
  "metadata" : {
  },
  "global" : {
  },
  "components": {
    "slider-appmaster": {
    },
    "REDIS": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    },
    "MEMCACHED": {
      "yarn.role.priority": "1",
      "yarn.component.instances": "2",
      "yarn.memory": "512"
    }

  }
}
```