Features

Job Infrastructure Relaunching (relaunch of Carte)

Jobs failing due to a bad server or bad kettle, try them again up to a limit. Also monitor startup times, if exceed, tear down and start again.
(completed)

Job Relaunching attempts

Even if jobs fail, allow jobs to be automatically restarted. Need ability for jobs themselves to control and finely adjust the retry behavior.
(completed)

Client Global Parameters

Set per Client group of global (per client) variables that can be maintained outside of the scope of an individual job template.
(completed)

Manual job/slave launching

Ability to launch a single slave, and queue up a few jobs to run in a manual fashion. Add additional UI pieces as well.
Refactoring the specific SlavesController code for launching into a separate component class such that the url endpoints are still there in the controllers, but the actual work is done by calls to the new class. That way there aren't any ugly cross-controller-calls going on, and a simple check for a "now" schedule on template creation can instantiate the job and launch a slave.
(completed)

Independent monitoring of slave server's carte/kettle and slave itself

Part of job relaunching, have some services that can perform the above monitoring and take actions on failures.
(completed) (as far as displaying the status and shutting slave down if unavailable)

MySQL Internal IP binding

Change binding to be ONLY internal IP. Restricts outside development unless ssh'd in.
(completed)

CRUD improvements (Slaves)

Similar to above, show results of carte_url/kettle/status/?xml=y, should be very simple addition to SlaveController and the view.php view.
(completed)

VARIOUS DOC

CRUD

Web Interface

The Master Server supports a web interface with a CRUD editor built-in. This allows for manual CRUD actions on all data for which a table Model has been created.

It also allows manual job launching, job template creation (and immediate launching), manual slave launching, and manual slave teardown by deleting the slave from the slaves table.

Web application architecture

Framework

We used the Yii Framework for the PHP code. Yii is a MVC framework, with all application-specific code existing in the `app/` directory. Static content exists in the `html/` directory, along with the `html/index.php` file which can be used to control site-wide debugging. Other site-wide configuration is stored within `app/config/main.php`. Models, views, and controllers are stored in their respective directories.

Generating new models

Should the DDL change, it is a good idea to regenerate models within Yii. This is most simply done through the Gii interface, which appears in the navigation when debugging is enabled. The password is pre-defined to be simply ‘pass’. This blog: http://www.larryullman.com/2009/11/07/creating-models-views-and-controllers-in-yii-2/ provides a good introduction to working with Gii. **One caveat: controller regeneration should rarely be necessary, check the diff before overwriting any changes as most app-specific code exists in the controllers**. Also when controllers have been generated, for security reasons it’s a good idea to limit the access control to only allow admin users to do anything. See `ClientParamController.php` for a base example.

Finding and editing application specific code

Most important code is contained within the models and controllers of the data object of interest. The `SlavesController.php` file has several functions governing slave cloud server creation and destruction within the (used to be NephoScale API) Amazon

Web Services API. The `JobController.php` file governs the creation of new jobs from pre-existing job templates.

Look and Feel

The core application structure is defined within the files in `app/views/layouts/`, with each individual element having its own folder within the `views/` folder. Generic site views should be kept in the `sites/` folder.

Security

Passwords

All passwords related to the Master controller are stored in `app/config/auth.php` with in-production values living only on the Master server itself, not in source control. The values include the MySQL connection string and a MySQL user + password, an “actionAuth” token used to verify curl requests, and lastly the password for the web UI user ‘**admin**’ when logging in via the browser. The “actionAuth” token appears elsewhere only in the cronjob list.

In addition, you will need to specify an **AWS_KEY, AWS_SECRET_KEY, and AWS_ACCOUNT_ID** in the `app/components/AmazonWS.php` file to interact with Amazon. Note there may be other settings in this file you must edit for full ec2 customization. (Such as the zone.)

If AWS isn’t working, you may need to modify a PHP class preloader function. It should work out of the box, however. If it doesn’t, try step 43 in the master_install document.

SSL

A self-signed SSL certificate should be generated for the site, allowing you to use https. (Recommended.)

MySQL

MySQL should be configured with a single root user with the MySQL server accessible only through the internal private network.

Configuration

Master Config

This table contains a few options used by the Master application that may need to be adjusted over time, such as how many slave servers to launch in a given day. It also includes key options for selecting what kinds of Amazon instances to run.

Cron jobs

Cron jobs are controlled by the Master's crontab. Currently daily, weekly, and monthly jobs get scheduled at a particular time. The number of slaves to launch daily, defined in the master config, are launched near the same time daily jobs are scheduled. A monitoring service checks every five minutes for completed jobs and shuts down the slaves servers when all jobs have been completed.

Rerun a job

- Mark the job as status = unassigned, and clear out slave server lock id (in admin UI)
- Create a new "slave" in admin UI
- Launch a single slave (with same hostname as above) using Amazon's ec2 manager
- Prevent manual slave from being removed

Master Configurations Settings:
Ignore the following, it is obsolete.
DAILY_IMAGE_TYPE, DAILY_IMAGE_TO_START are Master Configuration parameters set in the Elastic Job Master UI. These must correspond to one of the id's from the following api calls:

- *DAILY_IMAGE_TYPE: https://api.nephoscale.com/server/type/cloud/ for server_type that defines your instance size. Default 11.*
- *DAILY_IMAGE_TO_START: https://api.nephoscale.com/image/server/ for the image (OS) you want to start. Default 11.*

The following is the real deal.
These are self-explanatory options for AWS, their values should be whatever one would send as an API call. They are strings.

```
    DAILY_IMAGE_TO_START   (string of stored image name)
    SLAVE_KEY_NAME         (security key for ssh)
    SLAVE_SECURITY_GROUP (security group)
```

```
SLAVE_INSTANCE_TYPE    (m1.small, etc.)
SLAVE_SHUTDOWN_BEHAVIOR  (restart, terminate, etc.)
```

Cloudfuse:
When you do a “df”, do you see the following error:
df: `/<mount>': Transport endpoint is not connected

if so, the Master lost its network connection to cloudstorage.
If that’s the case, you’ll want to unmount (or kill cloudfuse process) and then remount:
- umount /<mount>
- cloudfuse /<mount>

Sometimes, you may not be able to unmount, then you will need to look for the cloudfuse process and “kill -9 <process”.