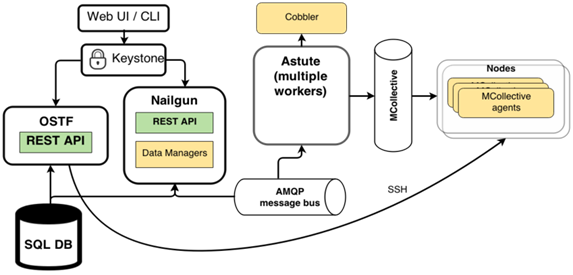
# Fuel 架构



Fuel由几个相互依赖的组件构成的，其中有一些是fuel定义的组件，还有一些是第三方服务提供的比如cobbler、puppet、mcollective等等。一些组件可以被重复利用不需要任何更改，而一些则需调整。

* UI 是一个用JaveScript写的页面应用，主要还是用bootstrap框架。源码: fuel-web/nailgun，运行态：nailgun容器。
* Nailgun是fuel的核心，当然是用python写的。它实现了REST API以及部署数据管理。这些数据包括磁盘卷配置、网络配置和其他使之部署成功必要的数据。它需要有编制逻辑去建造指令使部署能顺序性地完成。nailgun主要是使用SQL存储数据和AMQP服务来关联其他workers。Fuel的命令行也是由它提供的。源码: fuel-web/nailgun，运行态：nailgun容器。
* Astute是另外一个重要的组件，代表着nailgun的workers。它主要是根据nailgun提供的指令运行着某些操作。Astute实际上什么东西都没有只是一层封装着所有细节和相互影响的服务比如cobbler、puppet、shell scripts等等，和提供了异步通用接口给那些服务。它是通过其他基本本地协议（比如XML-RPC协议）来管理这些服务或者可以使用Mcollective agents去提供定义好的命令比如运行’puppet apply’在其他的远程节点上面或运行某些脚本。当然它是通过AMQP来与nailgun交互数据。源码: fuel-astute，运行态：astute容器。
* Cobbler 被用来提供快速网络安装的linux服务。源码: 无，运行态：cobbler容器。
* Mcollective ruby的编排框架，用于管理Puppet。源码: 无，运行态：Mcollective容器。
* Mcollective agents 实现具体的任务像设备清理，网络连接性测试等等。

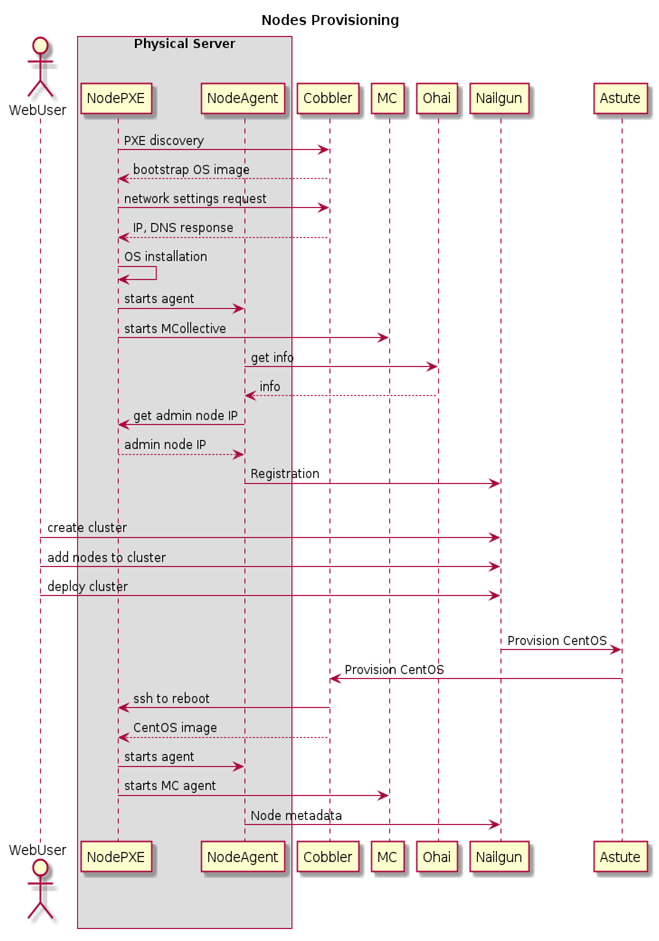
# Carrier流程剖析

## 节点发现及OS安装

Naligun的数据都是存储在postgreSQL（容器）数据库里。这些数据包含了新节点硬件配置信息、角色、环境配置、现状部署情况等。

管理节点里的PXE服务使用了特殊的bootstrap image来发现新的节点。这个bootstrap image运行着特殊的脚本——nailgun agent.这个脚本收集了服务器硬件信息并提交给了nailgun通过REST API。

部署程序是在用户配好了环境才开始启动的。Nailgun服务端会创建一个环境配置的JSON数据文件然后交给rabbitmq队列中。这个信息会被一个“任务执行者“——astute接受。最终由它去发送配置任务给其他组件安装部署节点。



**Fuel Master**

/opt/nailgun/bin/agent in ruby, rest api with nailgun

Mcollective docker

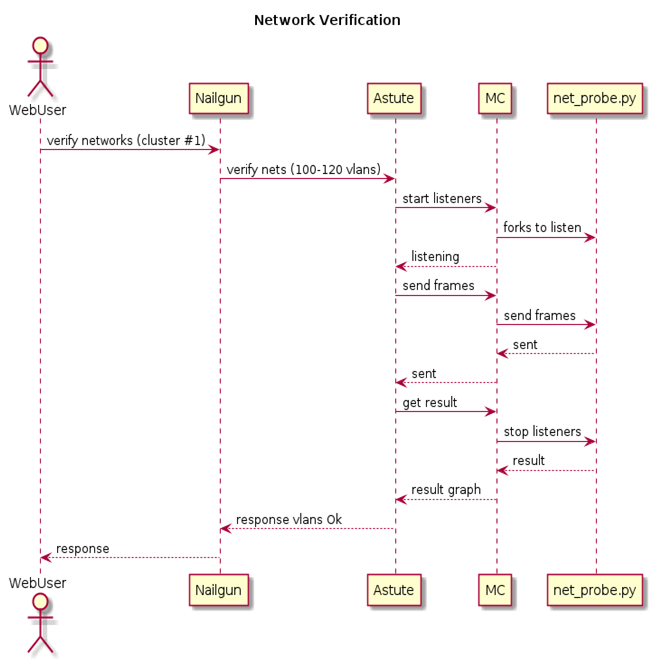
Nailgun docker

Astute docker

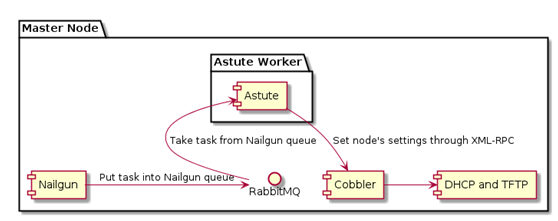
Cobbler docker

Fuel-web/nailgun/static/js/views/cluster\_page.js,rest api with nailgun

* Get request in in nailgun/api/v1/ handlers/cluster.py,
* Call task manager to save setting to postgres docker
* Call task manager to serialize a json to MQ with rpc.cast in nailgun/task/manager.py

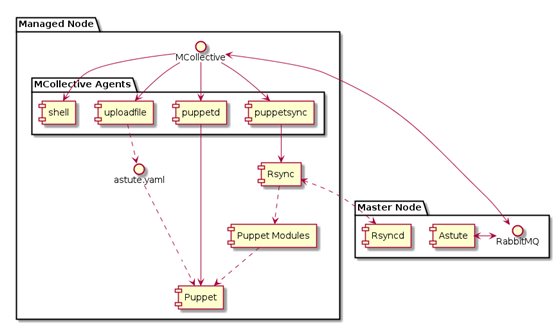


Astute workers会去主动监听rabbitmq队列并接受消息。首先，一旦有提供环境的节点，astute会使用XML-RPC去给cobbler设置那些节点的配置并使用mcollective重启那些节点让cobbler去安装基础操作系统。Cobbler是一个部署系统能控制DHCP和TFTP服务并使用他们去网络启动那些节点以及部署系统根据用户的配置内容。



## 平台部署

Astute发送一个特殊的信息给rabbitmq队列包含着所有需要在托管节点的操作。在已经被引导启动的节点上mcollectvie服务端会去监听刚才发送消息，一旦有消息过来他们就会运行所需的代理的操作与给定的参数。Mcollective客户端只是一组ruby脚本程序。这些程序会去运行mcollective所需要的执行的操作。



当节点的操作系统被安装完成后，Astute会去启动部署openstack的服务。首先它会上传节点的配置给/etc/astute.yaml这个文件，通过uploadfile agent这个文件包好了所有部署需要的变量与配置。

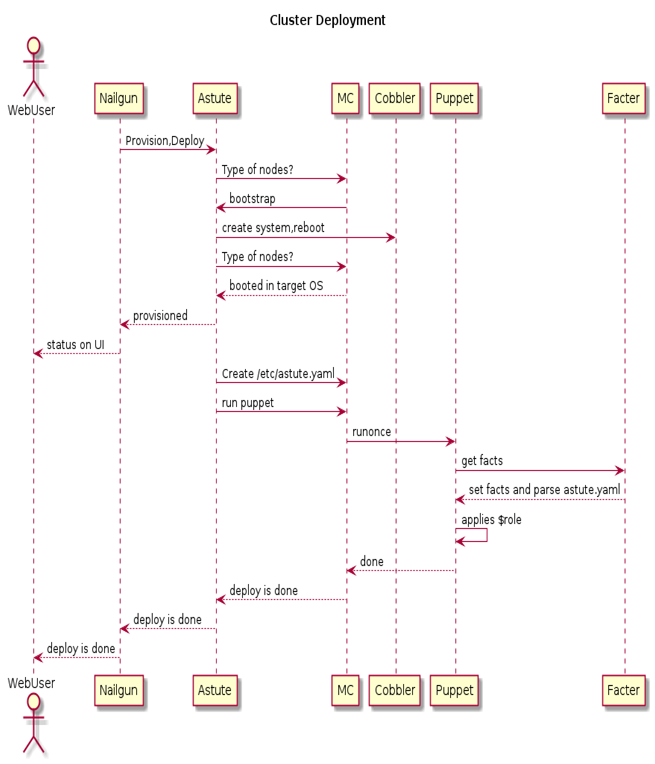
接下来，astute会去使用puppetsync agent去同步puppet模块和脚本。这个agent运行了一个rsync进程——它会去同步master节点并下载最新版本的puppet模块和脚本。

当模块同步完成后，astute就会去运行部署脚本通过puppet主要清单文件site.pp。mcolletve agent使用daemonize工具在后台运行puppet脚本进程：

daemonize puppet apply /etc/puppet/manifests/site.pp"

部署开始的时候，puppet会去读取astute.yaml文件内容通过fact并解析它到$fuel\_settings结构，这样一来就可以获取所有部署配置。

Astute会定期地调用agent去检查部署是否完成并汇报给nailgun，当然这一过程也是通过rabbitmq队列实现的。当puppet进程退出来之后无论成功还是返回一个错误，astute都会从节点得到一个报告文件并发送一个结果给nailgun。然后用户可以通过监控看到这个日志



# Carrier源码及build过程

* fuel-web: [git@git.tclab.lenovo.com:fuel-dev/fuel-web.git](mailto:git@git.tclab.lenovo.com:fuel-dev/fuel-web.git)，branch: sds/6.0

包含Fuel UI code和nailgun服务（容器）

* fuel-library: [git@git.tclab.lenovo.com:fuel-dev/fuel-library.git](mailto:git@git.tclab.lenovo.com:fuel-dev/fuel-library.git)，branch: sds/6.0

含fuel 中的puppet module

* dev-fuel60-buildfiles: [git@git.tclab.lenovo.com:fuel-dev/dev-fuel60-buildfiles.git](mailto:git@git.tclab.lenovo.com:fuel-dev/dev-fuel60-buildfiles.git)，branch: sds/6.0

用于build astute/nailgun/cobbler的项目文件

* rebuild\_iso\_cicd: [git@git.tclab.lenovo.com:fuel-dev/rebuild\_iso\_cicd.git](mailto:git@git.tclab.lenovo.com:fuel-dev/rebuild_iso_cicd.git)，branch: sds/6.0

用于build sds iso的项目文件

Local build:

下载rebuild\_iso\_cicd, 直接执行其下的rebuild.sh可自动build

Official build:

1. 先local build以生成新的fuel-images.tar.lrz
2. 上传fuel-images.tar.lrz到 [git@git.tclab.lenovo.com:nfv\_sds/base\_os.git](mailto:git@git.tclab.lenovo.com:nfv_sds/base_os.git) 下iso\_rebuild\docker\images