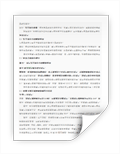
openstack中Heat说明文档

**导读：**Heat说明文档，2.2在配置文件中说明Orchestration服务存储的数据在数据库的位置，3.2命令使用说明，Heat说明文档1项目介绍1.1什么是Heat？Heat是一套业务流程平台，旨在帮助用户更轻松地配置以OpenStack为基础的云体系。利用Heat应用程序，开发人员能够在程序中使用模板以实现资源的自动化部署。Heat能够启动应用、创建虚拟机并自动处理整个流程。它还拥有出色的跨平



Heat说明文档

1 项目介绍

1.1 什么是Heat？

Heat是一套业务流程平台，旨在帮助用户更轻松地配置以OpenStack为基础的云体系。利用Heat应用程序，开发人员能够在程序中使用模板以实现资源的自动化部署。Heat能够启动应用、创建虚拟机并自动处理整个流程。它还拥有出色的跨平台兼容性，能够与Amazon Web Services业务流程平台CloudFormation相对接——这意味着用户完全可以将AWS模板引入OpenStack环境当中。

2 安装和配置

2.1 Install the Orchestration service

Install the Orchestration module on the controller node:

# yum install openstack-heat-api openstack-heat-engine

openstack-heat-api- cfn

2.2 在配置文件中说明Orchestration服务存储的数据在数据库的位置，下面

的例子使用MySQL数据库。

# openstack-config --set /etc/heat/heat.conf \

DEFAULT sql\_connection mysql://heat:HEAT\_DBPASS@controller/heat 以root身份和之前设置的密码登录，再创建一个名为heat的数据库。 # mysql -u root -p

mysql> CREATE DATABASE heat;

mysql> GRANT ALL PRIVILEGES ON heat.\* TO 'heat'@'localhost' \ IDENTIFIED BY 'HEAT\_DBPASS';

mysql> GRANT ALL PRIVILEGES ON heat.\* TO 'heat'@'%' \

IDENTIFIED BY 'HEAT\_DBPASS';

2.3 创建heat服务表

# heat-manage db\_sync

2.4 创建一个heat用户，Orachestration服务能够对其进行身份服务授权，使

用service tenant同时赋予该用户管理员权限。

# keystone user-create --name=heat --pass=HEAT\_PASS

--email=heat@example.

com

# keystone user-role-add --user=heat --tenant=service –role=admin

2.5 编辑/etc/heat/heat.conf 文件，在[keystone\_authtoken] 和

[ec2\_authtoken] 部分添加认证：

[keystone\_authtoken]

auth\_host = controller

auth\_port = 35357

auth\_protocol = http

auth\_uri = http://controller:5000/v2.0

admin\_tenant\_name = service

admin\_user = heat

admin\_password = HEAT\_PASS

[ec2\_authtoken]

auth\_uri = http://controller:5000/v2.0

keystone\_ec2\_uri = http://controller:5000/v2.0/ec2tokens

2.6 用身份服务去注册Heat和CloudFormation APIs以便其他openstack服务

能找到这些APIs，注册这些服务并指点端点：

# keystone service-create --name=heat --type=orchestration \ --description="Heat Orchestration API"

2.6.1 使用返回给服务的id属性去创建端点：

# keystone endpoint-create \

--service-id=the\_service\_id\_above \

--publicurl=http://controller:8004/v1/%\(tenant\_id\)s \

--internalurl=http://controller:8004/v1/%\(tenant\_id\)s \

--adminurl=http://controller:8004/v1/%\(tenant\_id\)s

# keystone service-create --name=heat-cfn --type=cloudformation \ --description="Heat CloudFormation API"

2.6.2 使用返回给服务的id去创建端点：

# keystone endpoint-create \

--service-id=the\_service\_id\_above \

--publicurl=http://controller:8000/v1 \

--internalurl=http://controller:8000/v1 \

--adminurl=http://controller:8000/v1

2.7 启动heat-api, heat-api-cfn 和 heat-engine 服务，同时配置他们使得

在机器启动的时候这些服务也会启动。

# service openstack-heat-api start

# service openstack-heat-api-cfn start

# service openstack-heat-engine start

# chkconfig openstack-heat-api on

# chkconfig openstack-heat-api-cfn on

# chkconfig openstack-heat-engine on

2.8 验证Orchestration服务的安装

为了验证Orchestration服务被正确安装和配置，确保所做的配置在

openrc.sh是正确的，使用下面的命令：

$ source openrc.sh

3 操作命令

3.1 命令用法

usage: heat [--version] [-d] [-v] [-k] [--os-cacert <ca-certificate>]

[--cert-file CERT\_FILE] [--key-file KEY\_FILE] [--ca-file CA\_FILE]

[--api-timeout API\_TIMEOUT] [--os-username OS\_USERNAME]

[--os-password OS\_PASSWORD] [--os-tenant-id OS\_TENANT\_ID]

[--os-tenant-name OS\_TENANT\_NAME] [--os-auth-url OS\_AUTH\_URL]

[--os-region-name OS\_REGION\_NAME] [--os-auth-token OS\_AUTH\_TOKEN]

[--os-no-client-auth] [--heat-url HEAT\_URL]

[--heat-api-version HEAT\_API\_VERSION]

[--os-service-type OS\_SERVICE\_TYPE]

[--os-endpoint-type OS\_ENDPOINT\_TYPE] [--include-password]

<subcommand> ...

3.2 命令使用说明

heat action-resume <NAME or ID>

重启或恢复栈

heat action-suspend <NAME or ID>

挂起栈

heat build-info

获取建栈的信息

heat event-list [-r <RESOURCE>] <NAME or ID>

stack的事件列表

heat event-show <NAME or ID> <RESOURCE> <EVENT>

描述stack的事件

usage: heat output-list <NAME or ID>

显示可用的输出

heat output-show <NAME or ID> <OUTPUT NAME>

显示可用输出的值

heat resource-list <NAME or ID>

显示某个stack资源的列表

usage: heat resource-metadata <NAME or ID> <RESOURCE>

列出元数据资源

heat resource-show <NAME or ID> <RESOURCE>

描述资源（例如一个wordperss的资源信息）

heat resource-signal [-D <DATA>] [-f <FILE>] <NAME or ID> <RESOURCE>

给资源发送信号

heat resource-template [-F <FORMAT>] <RESOURCE>

基于资源产生一个模板

heat resource-type-list

列出可用的资源类型

heat resource-type-show <RESOURCE\_TYPE>

显示资源类型

heat stack-abandon <NAME or ID>

放弃栈

heat stack-adopt [-f <FILE>] [-e <FILE or URL>] [-u <URL>] [-o <URL>]

[-c <TIMEOUT>] [-t <TIMEOUT>] [-a <FILE or URL>] [-r]

[-P <KEY1=VALUE1;KEY2=VALUE2...>]

<STACK\_NAME>

使用栈

heat stack-create [-f <FILE>] [-e <FILE or URL>] [-u <URL>] [-o <URL>]

[-c <TIMEOUT>] [-t <TIMEOUT>] [-r]

[-P <KEY1=VALUE1;KEY2=VALUE2...>]

<STACK\_NAME>

创建一个栈

heat stack-delete <NAME or ID> [<NAME or ID> ...]

删除一个栈

heat stack-list [-s] [-f <KEY1=VALUE1;KEY2=VALUE2...>] [-l <LIMIT>]

[-m <ID>] [-g]

列出用户所有的栈

heat stack-preview [-f <FILE>] [-e <FILE or URL>] [-u <URL>] [-o <URL>]

[-P <KEY1=VALUE1;KEY2=VALUE2...>]

<STACK\_NAME>

预览栈

heat stack-show <NAME or ID>

描述栈

heat stack-update [-f <FILE>] [-e <FILE or URL>] [-u <URL>] [-o <URL>]

[-t <TIMEOUT>] [-r] [-P <KEY1=VALUE1;KEY2=VALUE2...>] <NAME or ID>

更新栈

heat template-show <NAME or ID>

得到某个具体的栈的模板

heat template-validate [-u <URL>] [-f <FILE>] [-e <FILE or URL>]

[-o <URL>]

3.3 验证模板参数

命令可选参数

--version

显示版本信息

-d, --debug

Defaults to env[HEATCLIENT\_DEBUG].

-v, --verbose

打印更详细的输出

-k, --insecure

Explicitly allow the client to perform "insecure" SSL (https) requests. The server's certificate will not be verified against any certificate authorities. This option should be used with caution.

--os-cacert <ca-certificate>

Specify a CA bundle file to use in verifying a TLS (https) server certificate. Defaults to env[OS\_CACERT]

--cert-file CERT\_FILE

Path of certificate file to use in SSL connection. This file can optionally be prepended with the private key.

--key-file KEY\_FILE

Path of client key to use in SSL connection.This option is not necessary if your key is prepended to your cert file. --ca-file CA\_FILE

Path of CA SSL certificate(s) used to verify the remote server's certificate. Without this option the client looks for the default system CA certificates.

--api-timeout API\_TIMEOUT

Number of seconds to wait for an API response, defaults to system socket timeout

--os-username OS\_USERNAME

Defaults to env[OS\_USERNAME].

--os-password OS\_PASSWORD

Defaults to env[OS\_PASSWORD].

--os-tenant-id OS\_TENANT\_ID

Defaults to env[OS\_TENANT\_ID].

--os-tenant-name OS\_TENANT\_NAME

Defaults to env[OS\_TENANT\_NAME].

--os-auth-url OS\_AUTH\_URL

Defaults to env[OS\_AUTH\_URL].

--os-region-name OS\_REGION\_NAME

Defaults to env[OS\_REGION\_NAME].

--os-auth-token OS\_AUTH\_TOKEN

Defaults to env[OS\_AUTH\_TOKEN].

--os-no-client-auth

Do not contact keystone for a token. Defaults to

env[OS\_NO\_CLIENT\_AUTH].

--heat-url HEAT\_URL

Defaults to env[HEAT\_URL].

--heat-api-version HEAT\_API\_VERSION

Defaults to env[HEAT\_API\_VERSION] or 1.

--os-service-type OS\_SERVICE\_TYPE

Defaults to env[OS\_SERVICE\_TYPE].

--os-endpoint-type OS\_ENDPOINT\_TYPE

Defaults to env[OS\_ENDPOINT\_TYPE].

--include-password

Send os-username and os-password to heat

4 创建和管理stacks（Dashboard中的使用）

4.1 创建stack的前提是已经安装好Devstack，这里就忽略Devstack的安装过

程，在安装好的Devstack的Dashboard中可以看到图1所示界面。

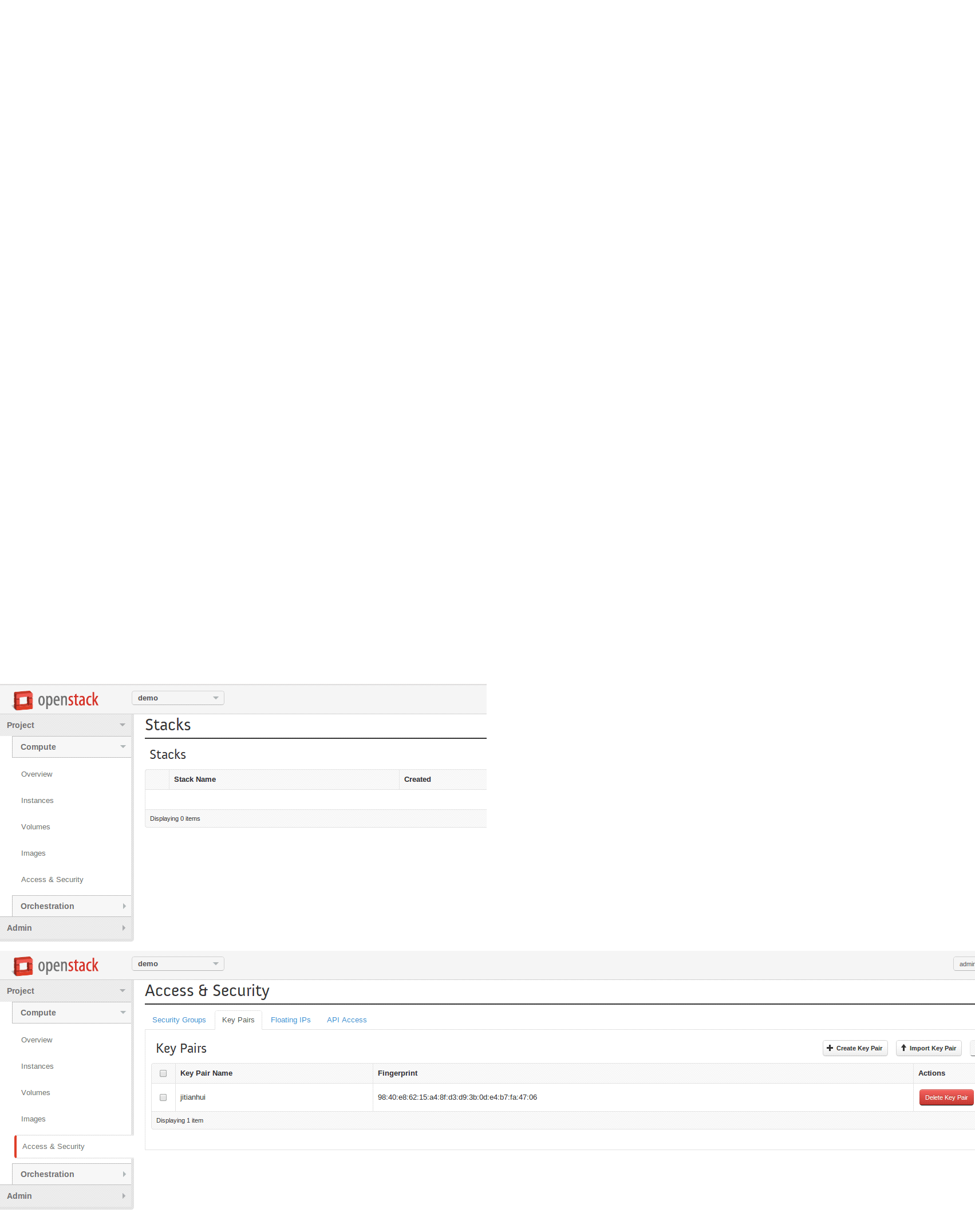


图1（Dashboard界面）

4.2 创建stack需要模板

可以自己定义，也可以直接利用模板的URL，也可以从网站上下载模板并加载到stack中，模板可以创建不同的应用，这里就以wordpress应用并利用在线的模板为例创建stack，因为有的模板需要keypair有的不许要，这里的例子是需要的，那么首先需要创建一个keypai。

4.2.1 进入Access&Security选项中，可以看到如图2所示界面：

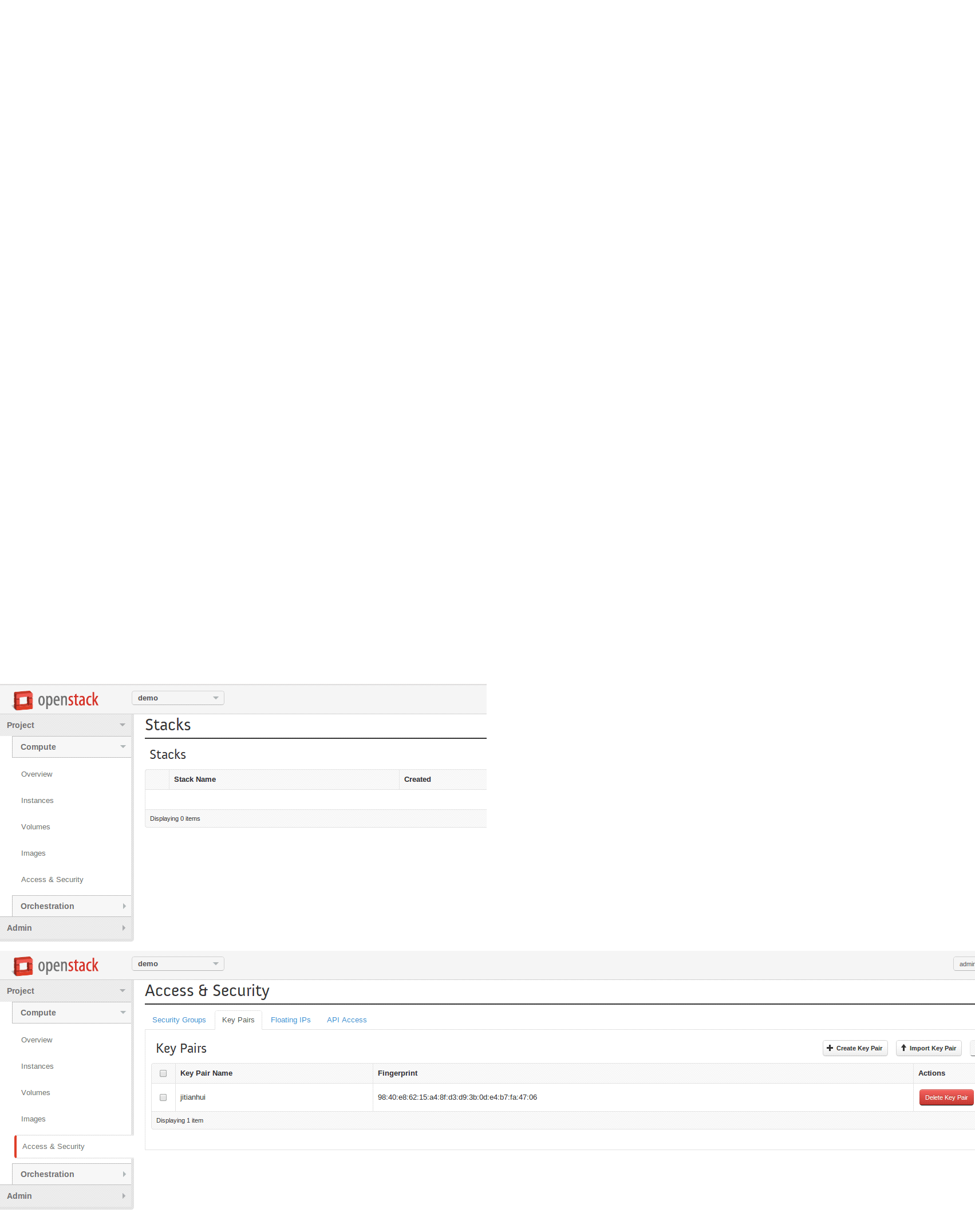
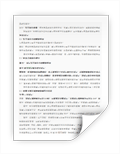


图2（Access&Security界面）

4.2.2 单击右上角的Create Key Pair，如图3所示，输入Key Pair Name，单击Create

Kye Pair。

**导读：**这里是参考官网文档中的模板，图3（CreateKeyPair）4.2.3结果如图4所示图4（结果）4.3Stack的创建4.3.1进入Orchestration列表下的Stacks选项，如图5所示。图5（Stacks界面）4.3.2单击右上角的LaunchStacks，会出现一个创建stack的界面并输入相应的信息，如图6所示，其中TemplateSource选择URL，在Templa



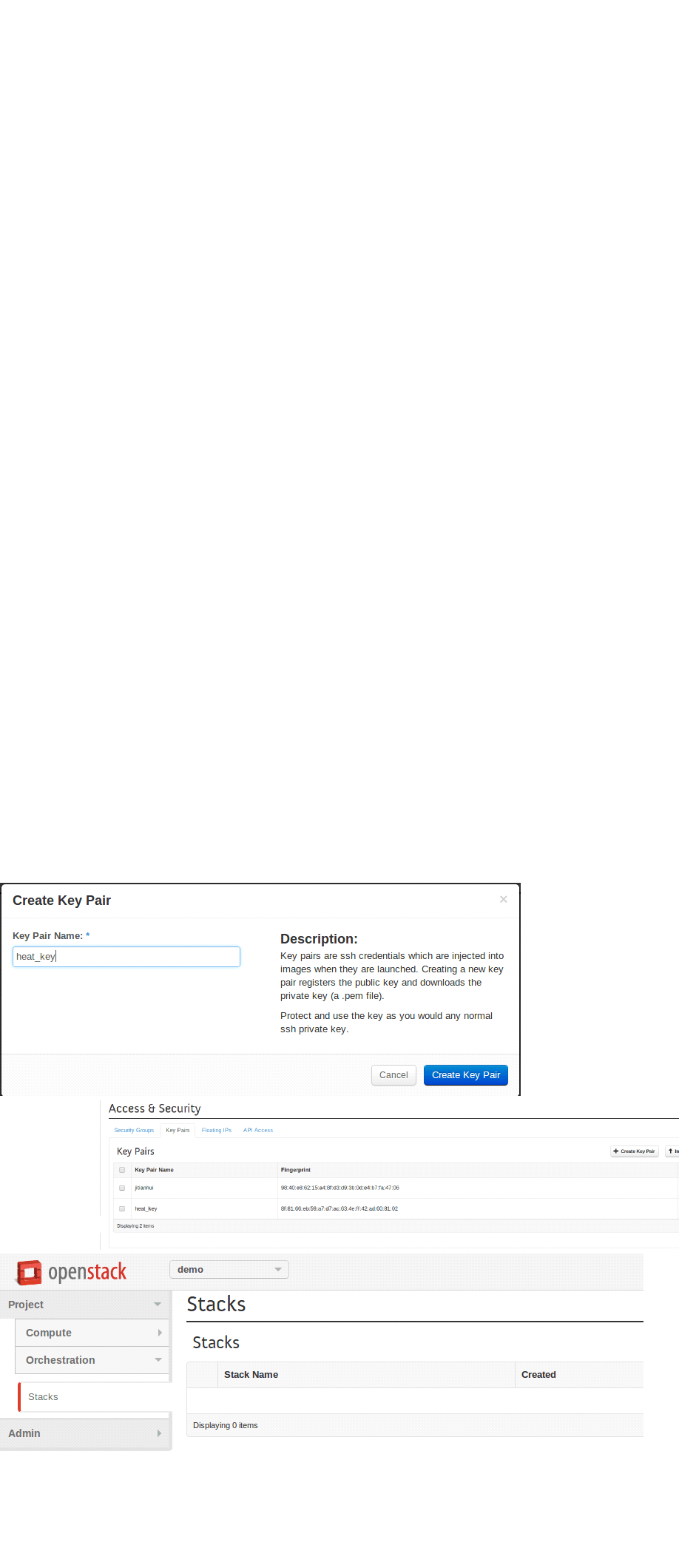


图3（Create Key Pair）

4.2.3 结果如图4所示

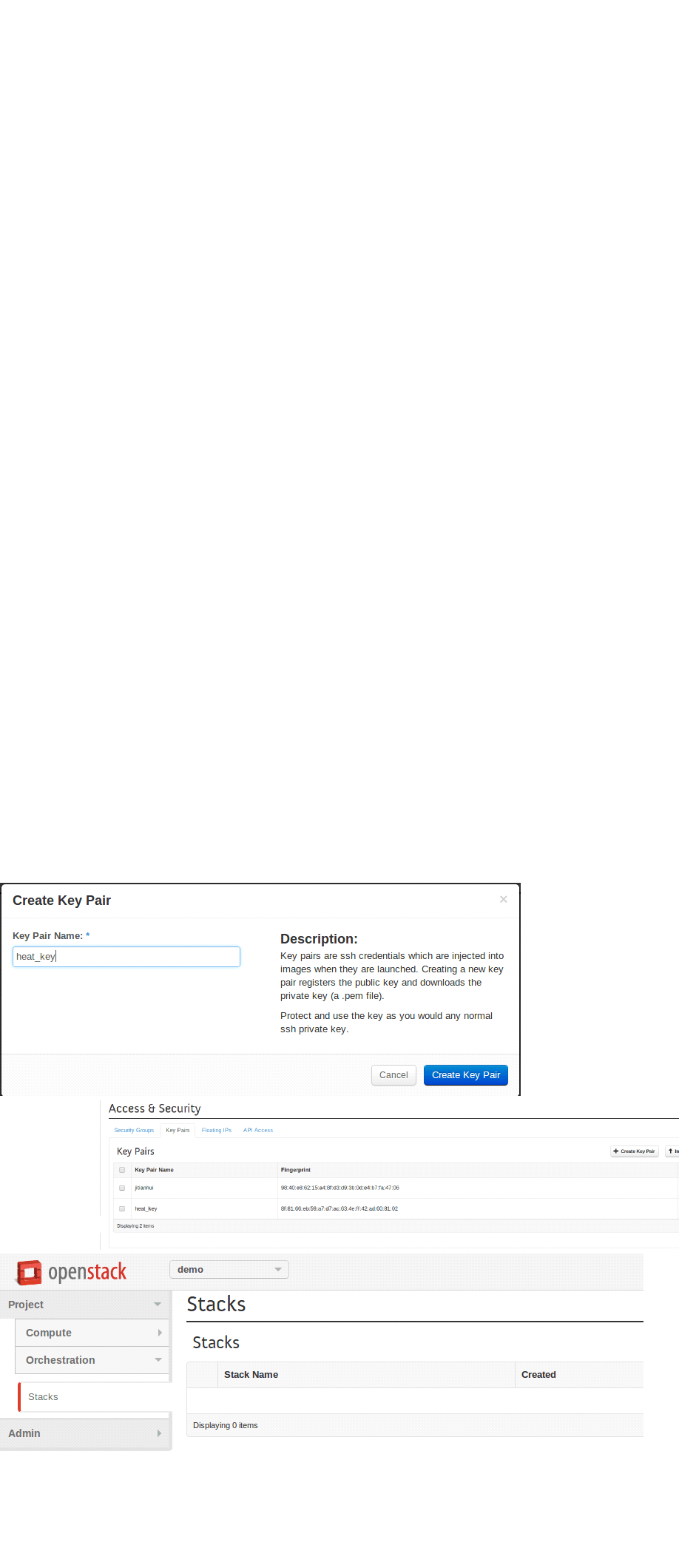


图4（结果）

4.3 Stack的创建

4.3.1 进入Orchestration列表下的Stacks选项，如图5所示。

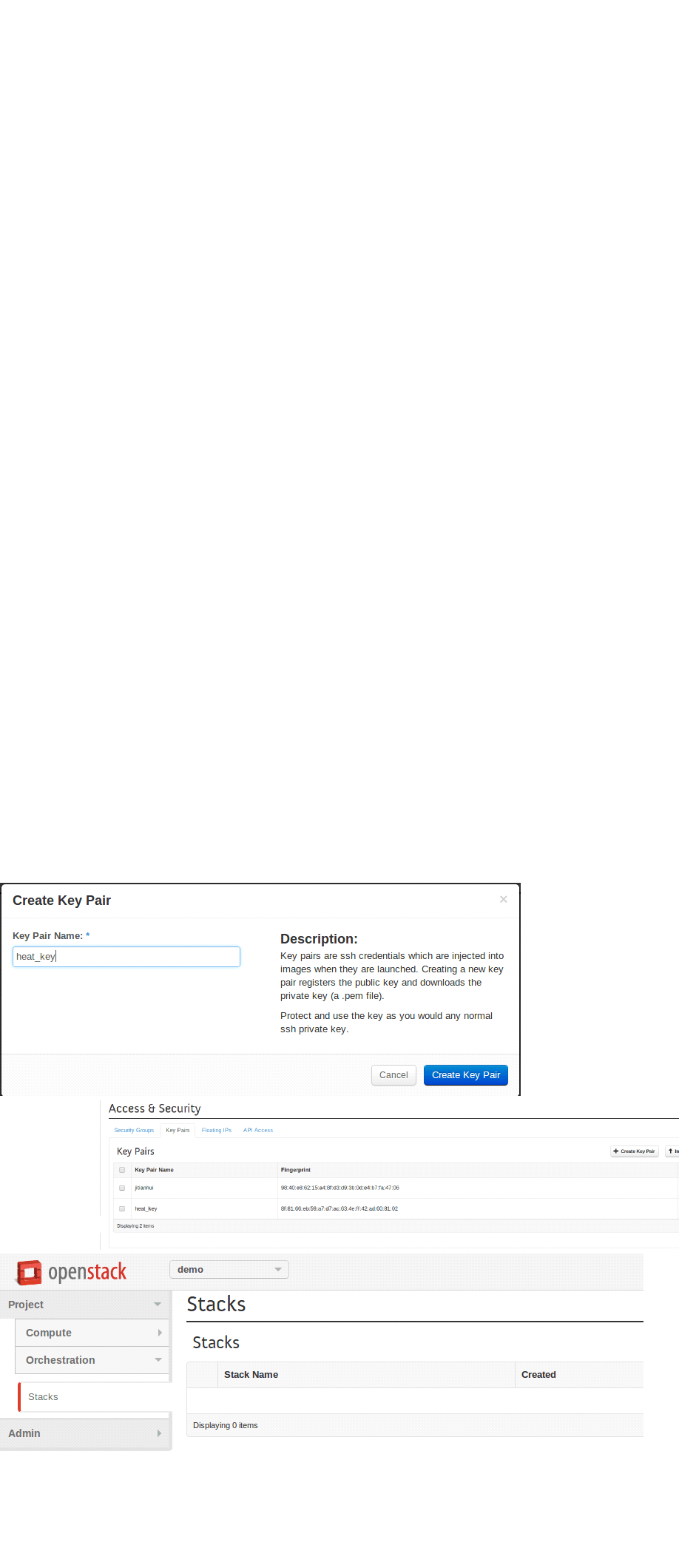


图5（Stacks界面）

4.3.2 单击右上角的Launch Stacks，会出现一个创建stack的界面并输入相应的信

息，如图6所示，其中Template Source选择URL，在Template URL中输入模板的地址，这里是参考官网文档中的模板，输入后单击Next。

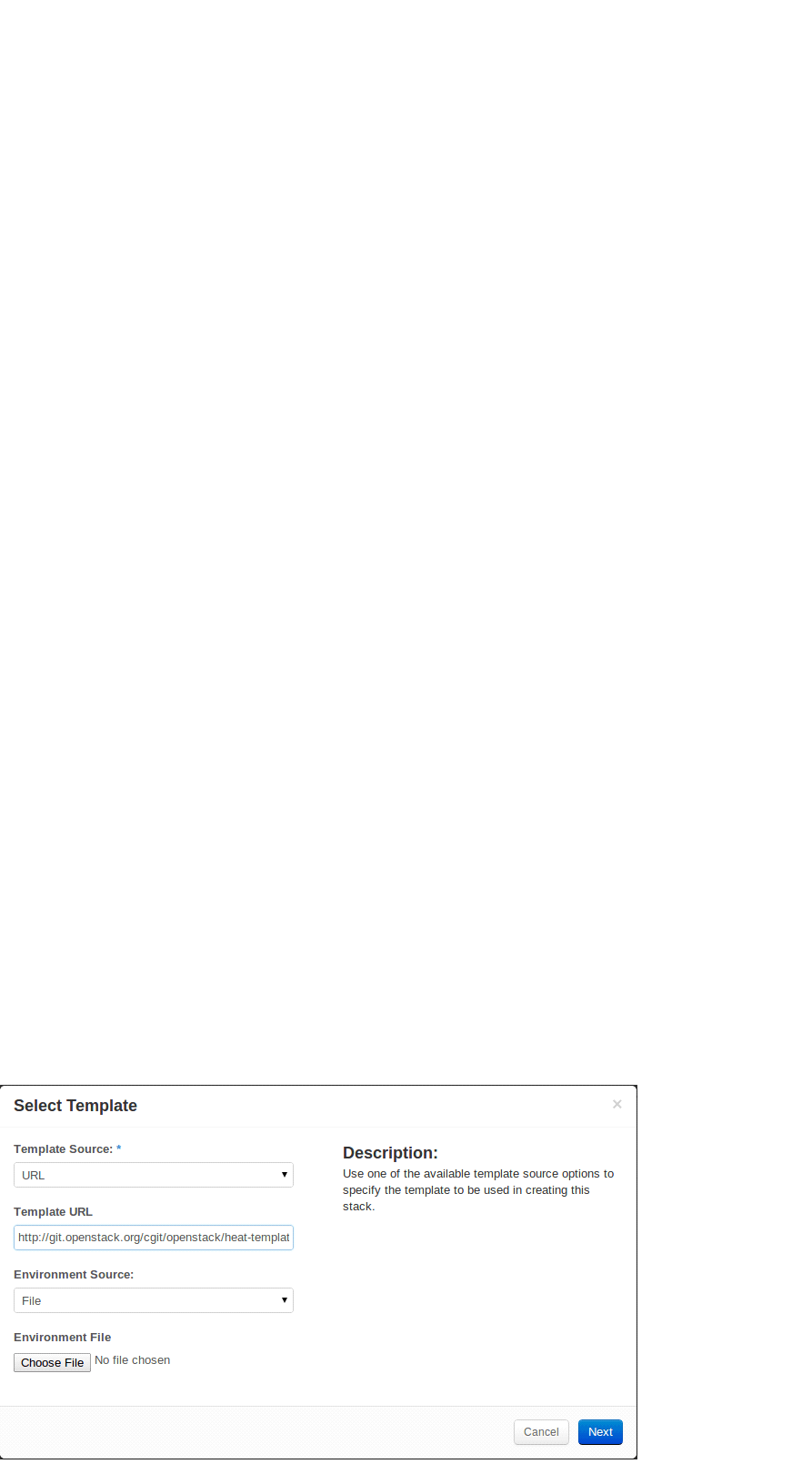


图6（Stack信息输入界面）

4.3.3 单击Next后会出现填入模板中规定字段内容的界面

具体如图7所示，其中Stack Name为必填字段，可以任意取值；Password for usr “admin”字段内容所填的是登录dashboard界面的密码；由于官网中所显示的image版本已经发生变化，这里就用Fedora-x86\_64-20-20140618-sda镜像；key\_name字段所填入内容为步骤1中所创建key\_pair的名字，为heat\_key。其余字段在模板文件中已经写好，不需要改动，单击Launch。

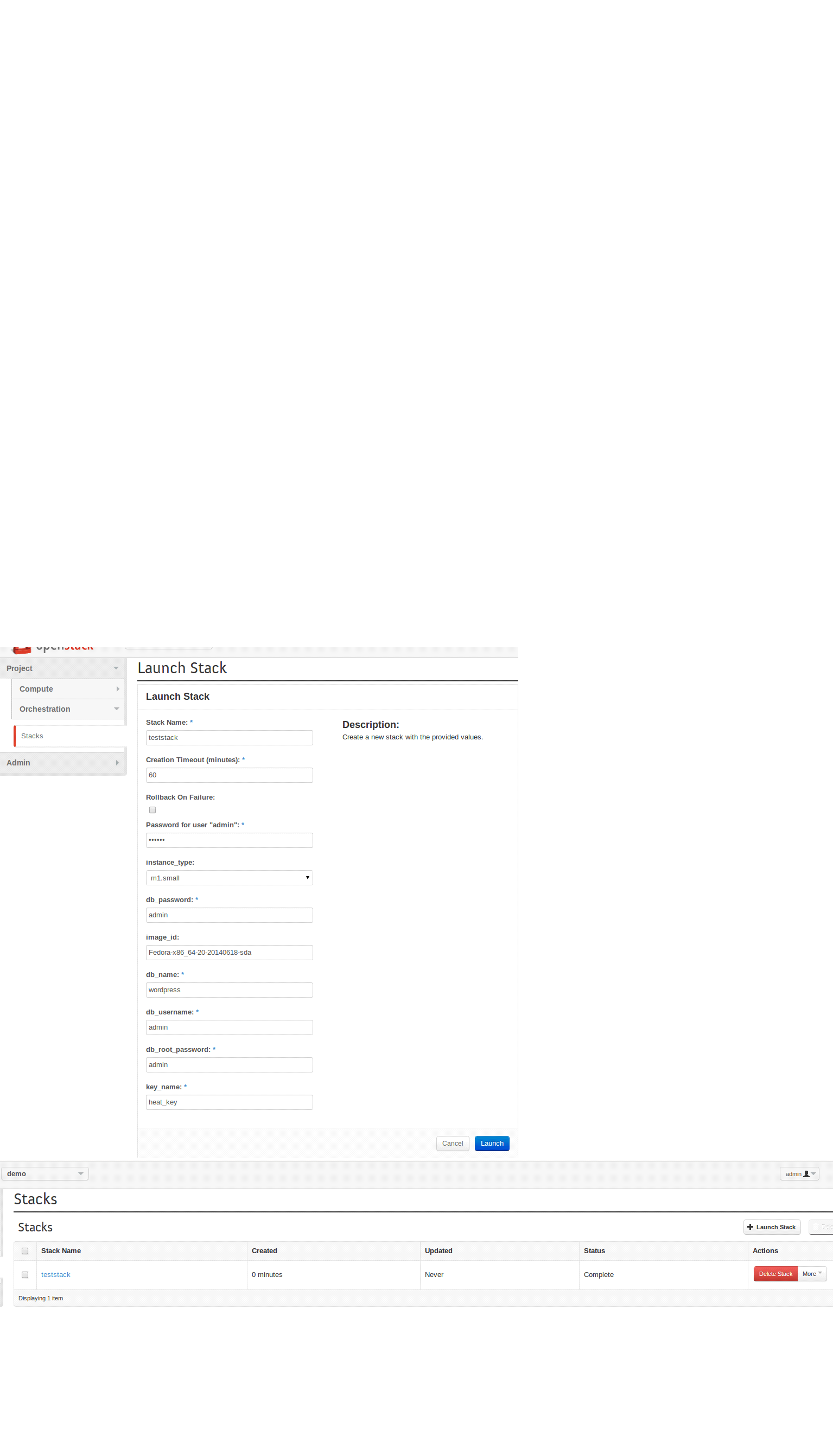


图7（模板信息内容界面）

4.3.4 完成之后可以看到如图8所示的界面，可以看到创建已经完毕。

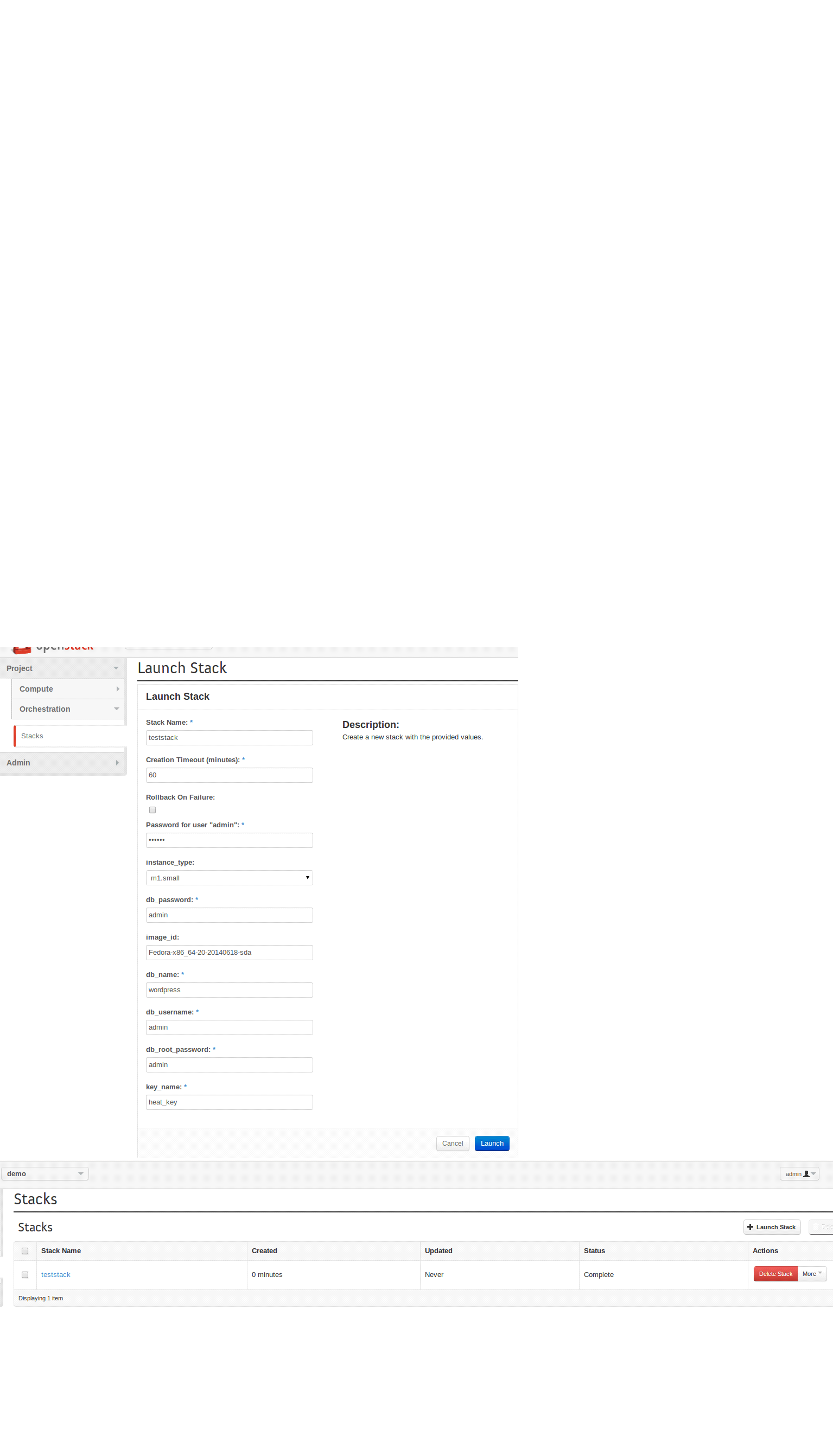


图8（stack创建结果）

4.3.5 单击stack的名字teststack之后可以看到该stack具体信息

进入event选项后可以看到如图9所示界面，可以看到stack的创建有两个状态，一个是Create In Progress，这个状态持续十几秒之后就会出现第二个状态（一般来说每个stack的创建都会有这两个状态，但是不同模板的stack的create in progress时间不一样），即Create Complate状态。

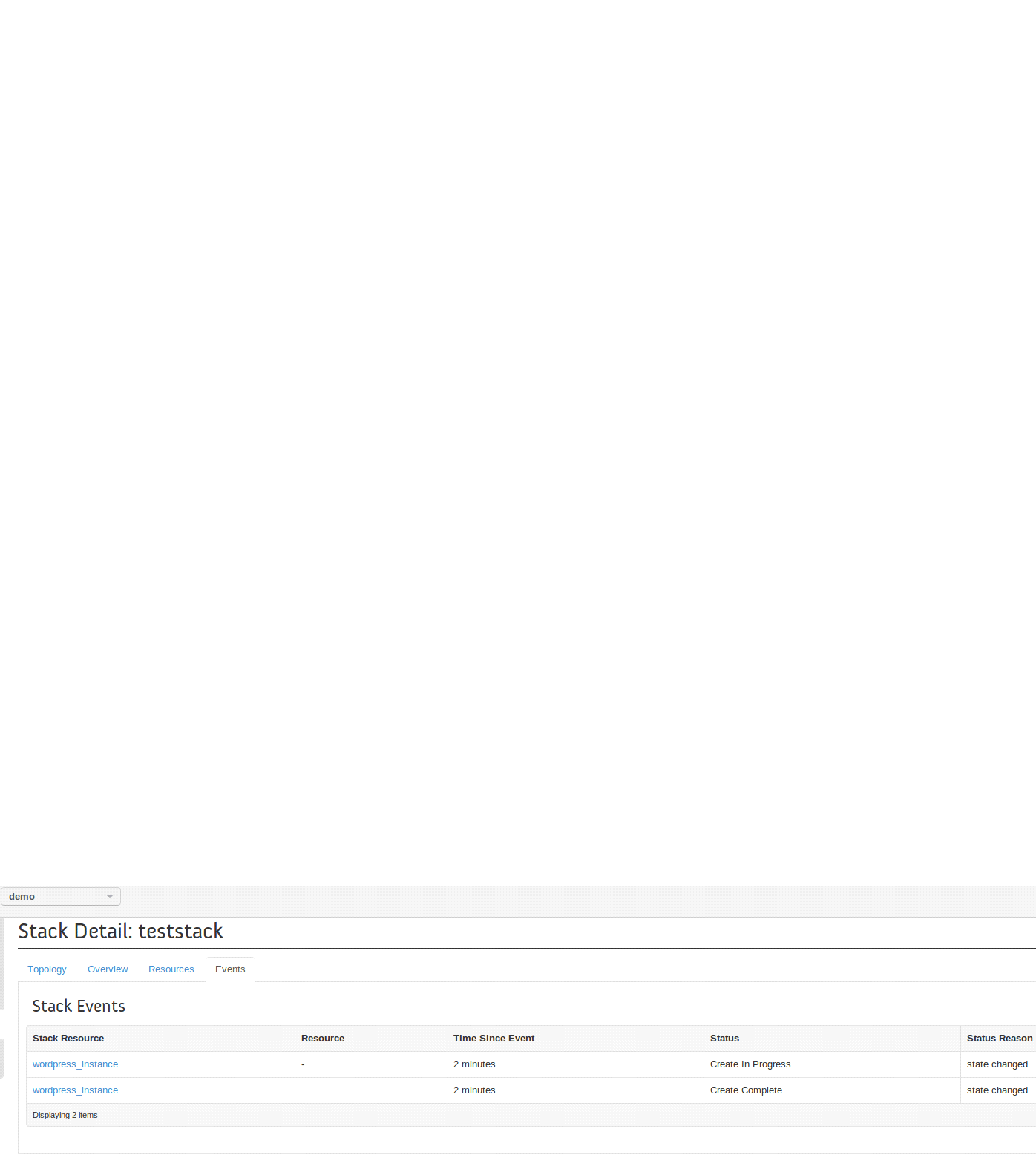


图9（stack实例Events信息）

4.3.6 再单击Oerview选项后可以看到模板的详细信息

如图10所示。可以看到返回的wordpress链接地址信息以及其他一些基本信息（模板中output部分）。等云主机创建好并部署好wordpress应用之后就可以通过这个链接地址访问。

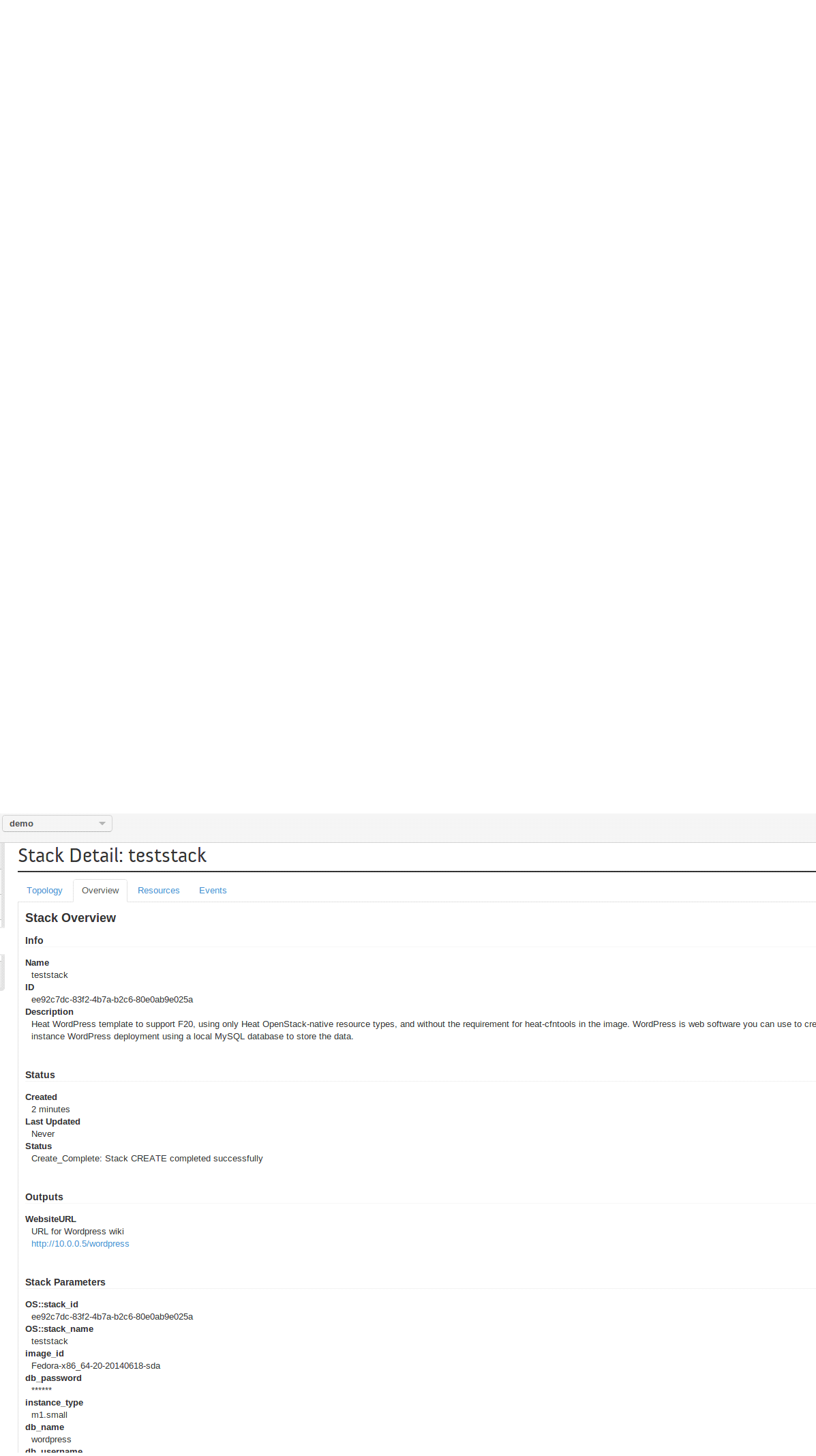


图10(stack详细信息)

4.3.7 接着进入Compute下面的Instances选项，可以看到之前创建的stack中创建出

来的虚拟机，如图11所示。

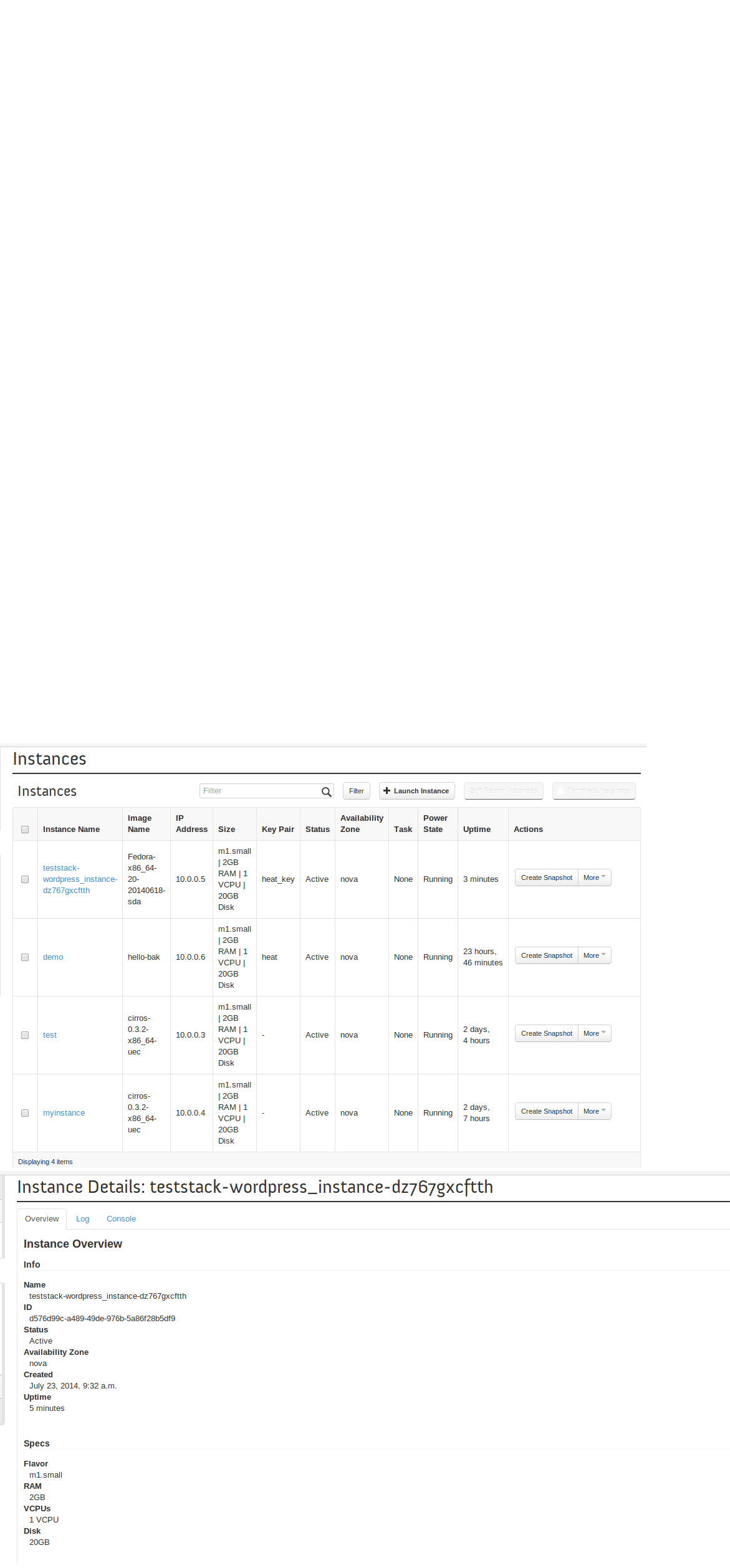


图11（instance信息）

4.3.8 单击该虚拟机的名字进入该虚拟机的详细信息界面，如图12所示。

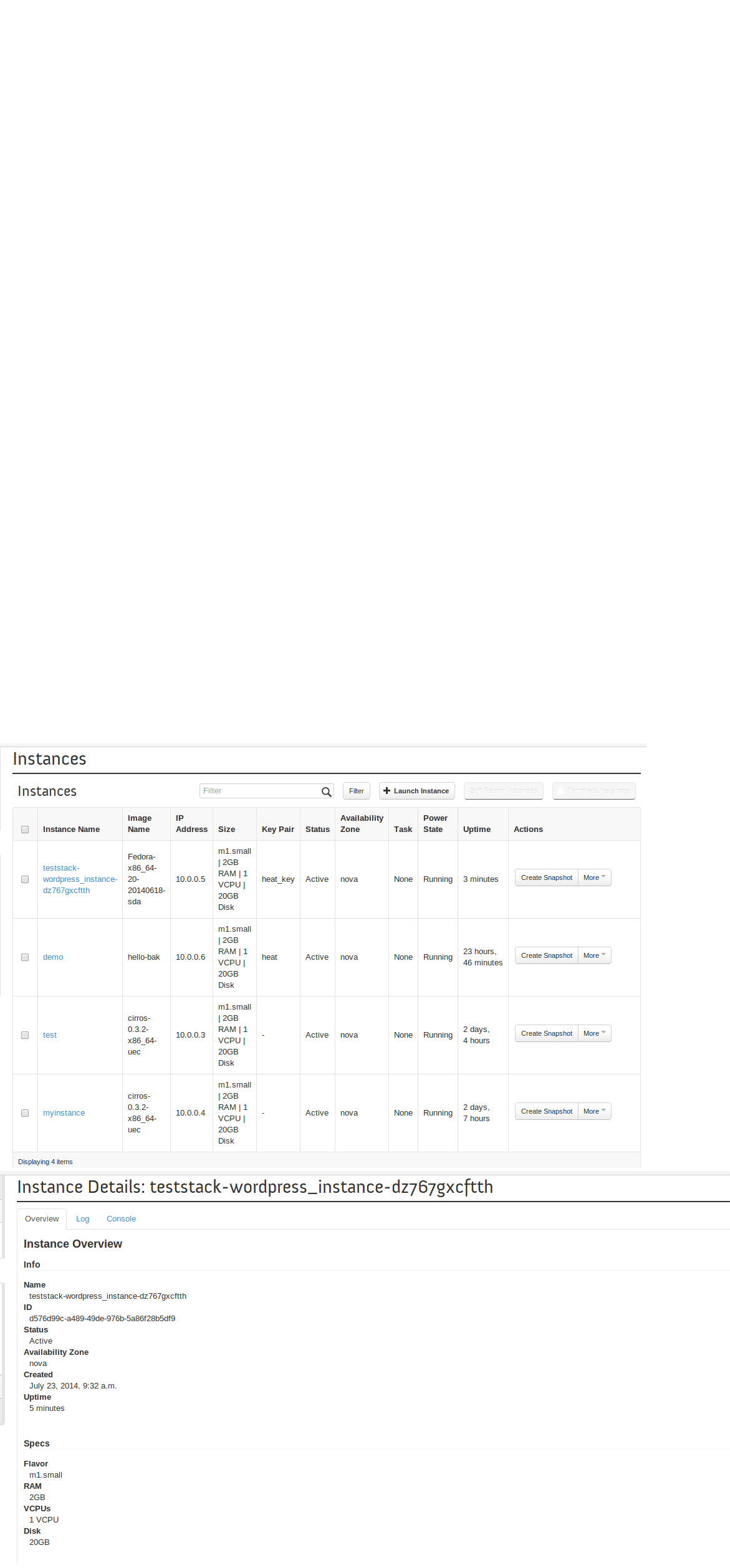
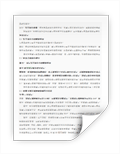


图12（instance详细信息）

4.3.9 单击Log选项进入创建过程日志，可以看到整个创建过程的日志，如图13所示。

**导读：**图13（日志界面）4.4在等待创建完毕的过程中可以通过终端去wget该wordpress应用地址，以便于我们知道该应用的地址是否能够被访问。结果如图14所示：图14（终端测试）4.5最后可以通过浏览器去访问该wordpress应用。4.5.1在浏览器中输入创建stack过程中返回的链接地址，在本例中是http://10.0.0.5/wordpress，结果如



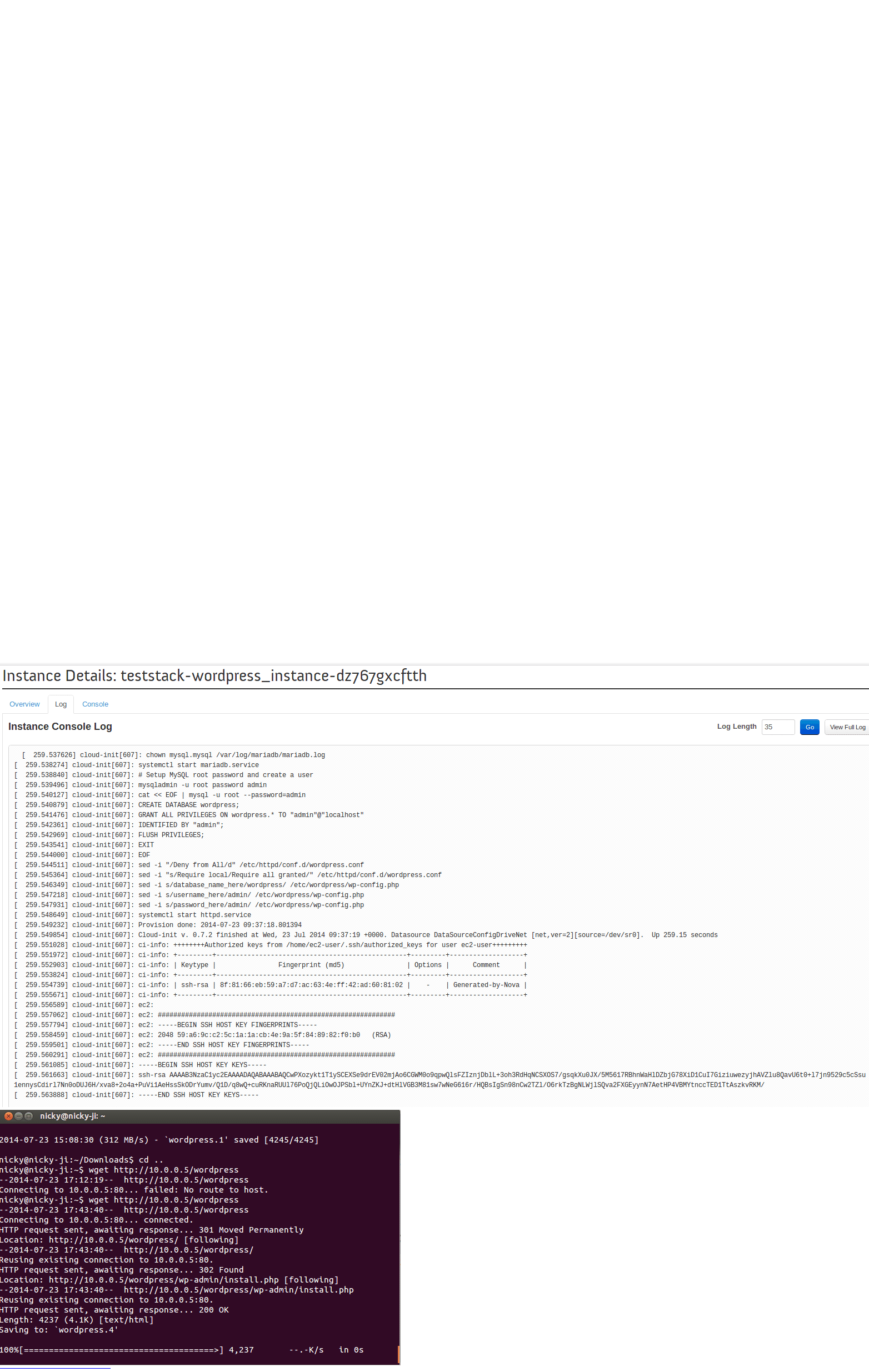


图13（日志界面）

4.4 在等待创建完毕的过程中可以通过终端去wget该wordpress应用地址，以

便于我们知道该应用的地址是否能够被访问。结果如图14所示：

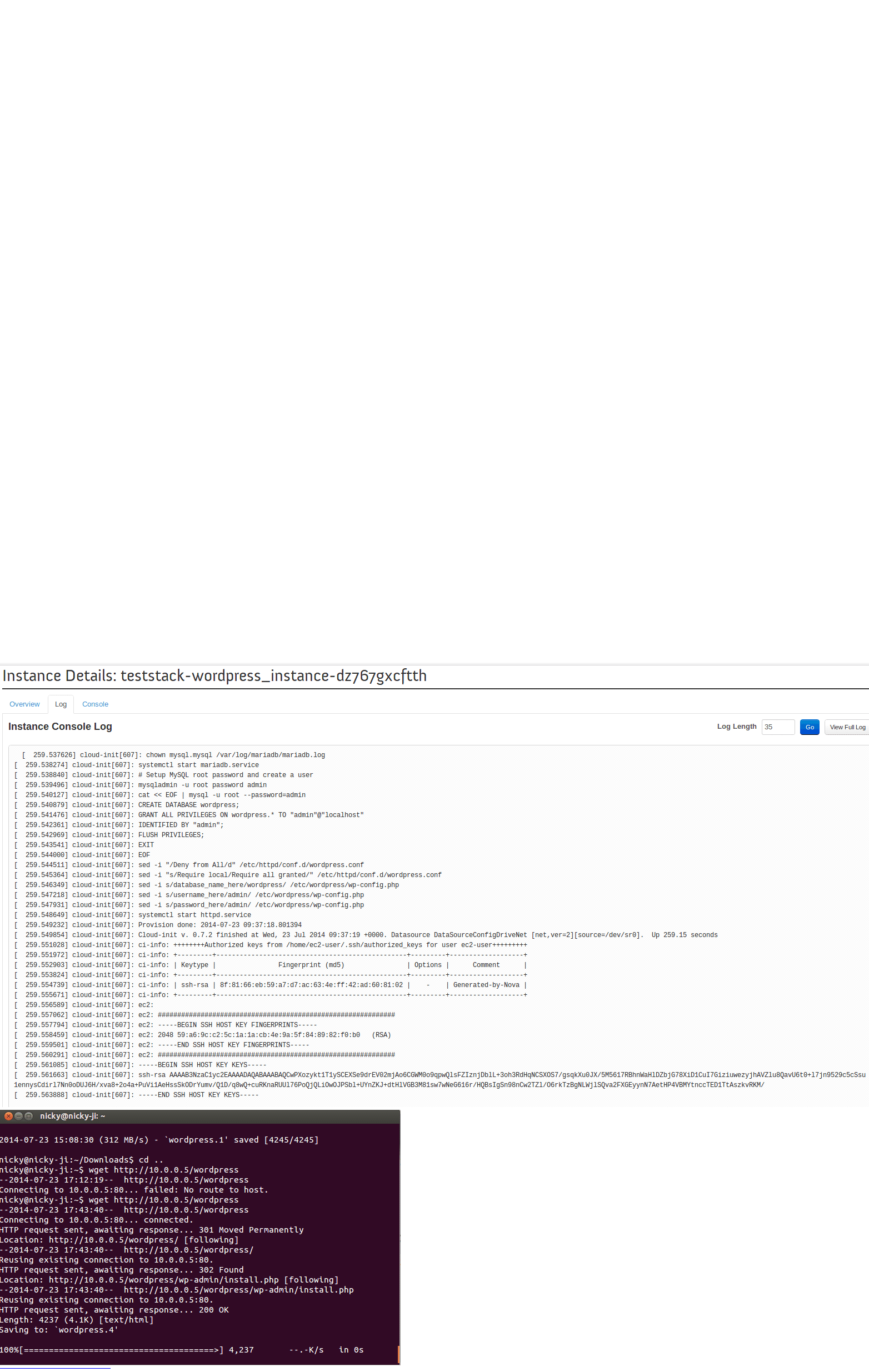


图14（终端测试）

4.5 最后可以通过浏览器去访问该wordpress应用。

4.5.1 在浏览器中输入创建stack过程中返回的链接地址，在本例中是http://10.0.0.5/wordpress，结果如图15所示，可以看到正在加载的界面。加

载完毕的界面如图16所示。

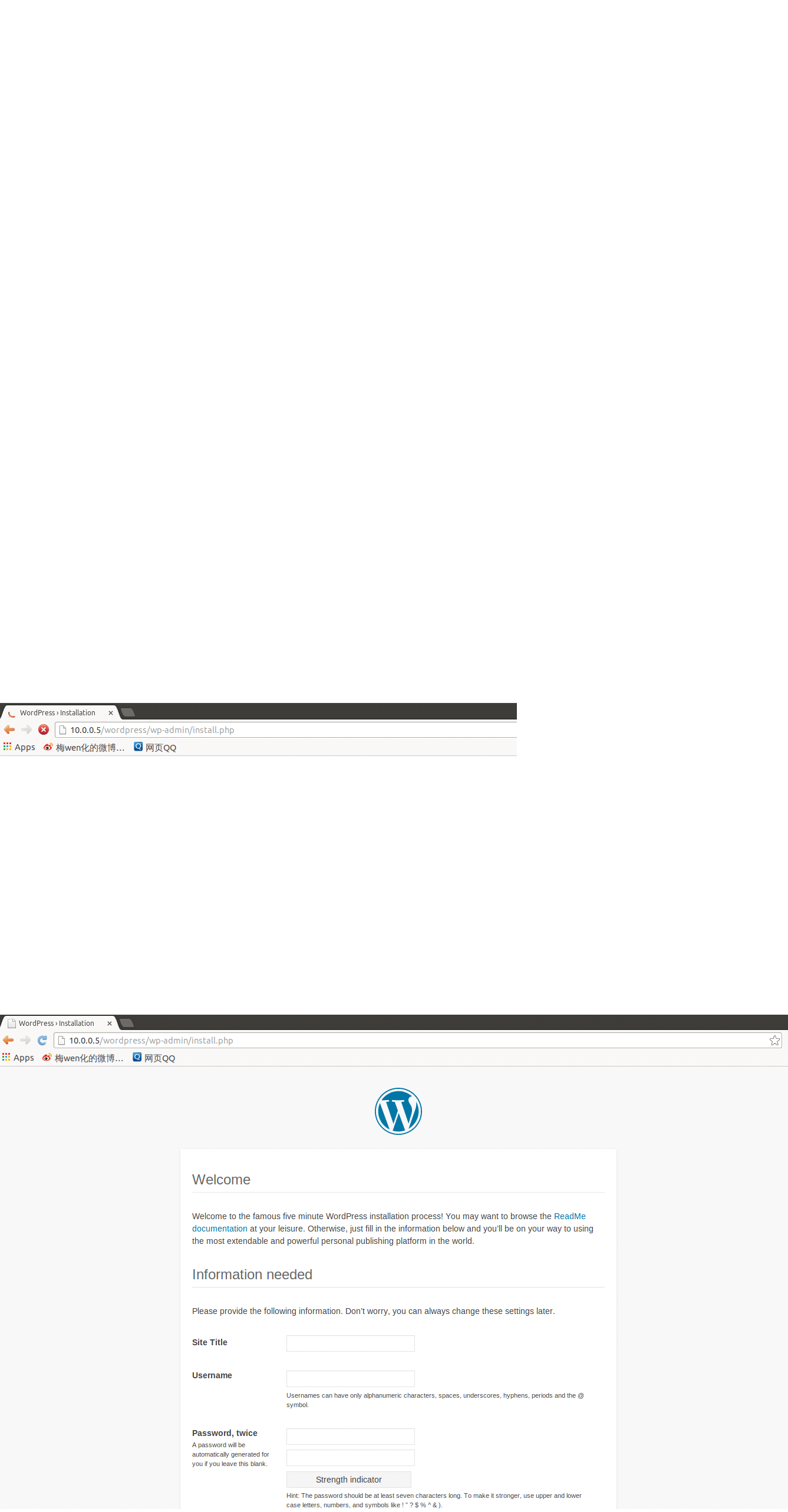


图15（wordpress界面加载）

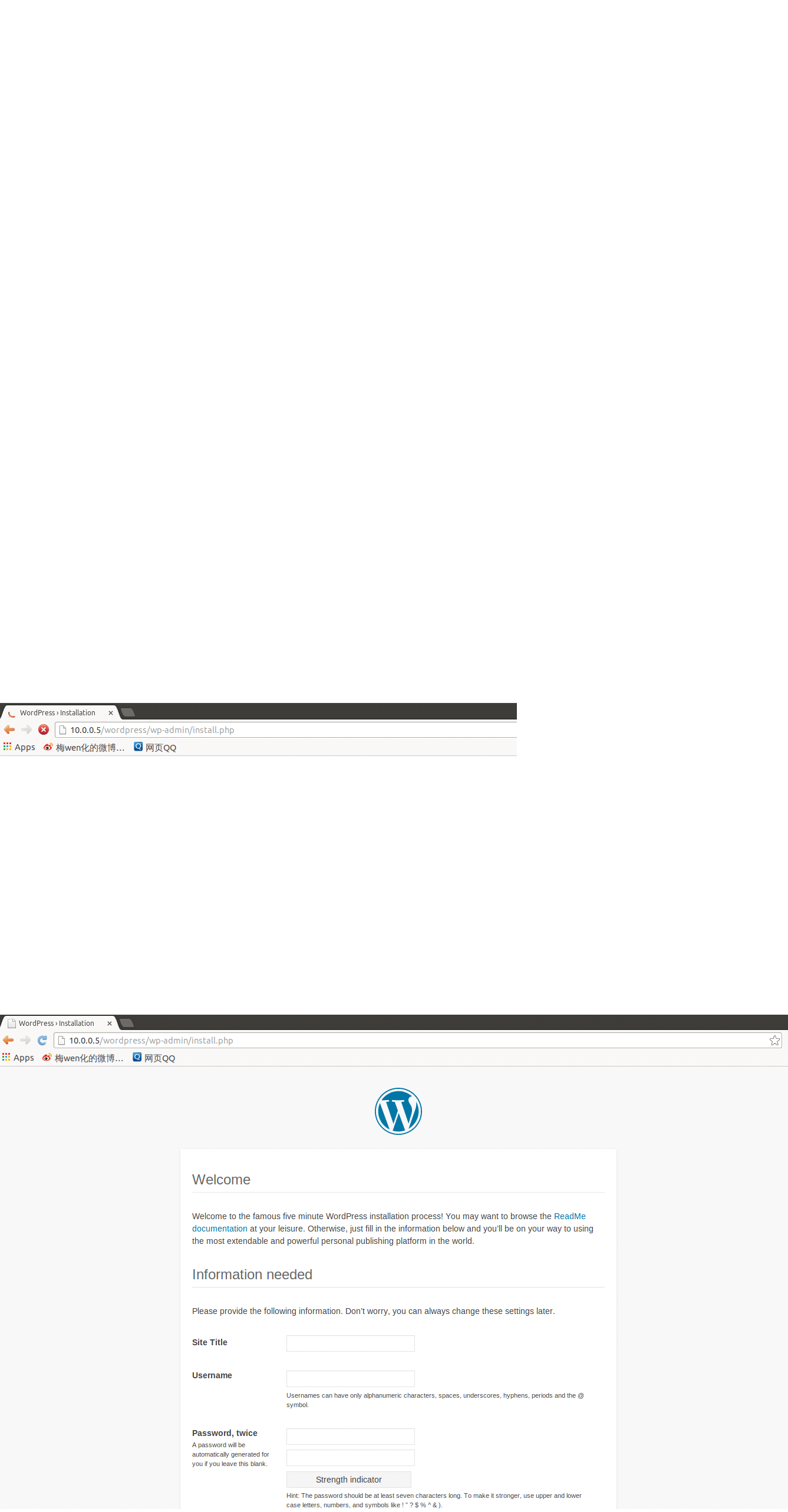


图16（加载完毕）

4.5.2 在加载完毕的界面中输入相应的信息（标题、用户名、密码以及邮箱），如图

17所示，然后单击Install WordPress就可以安装这个应用了。安装完毕的界面

如图18所示

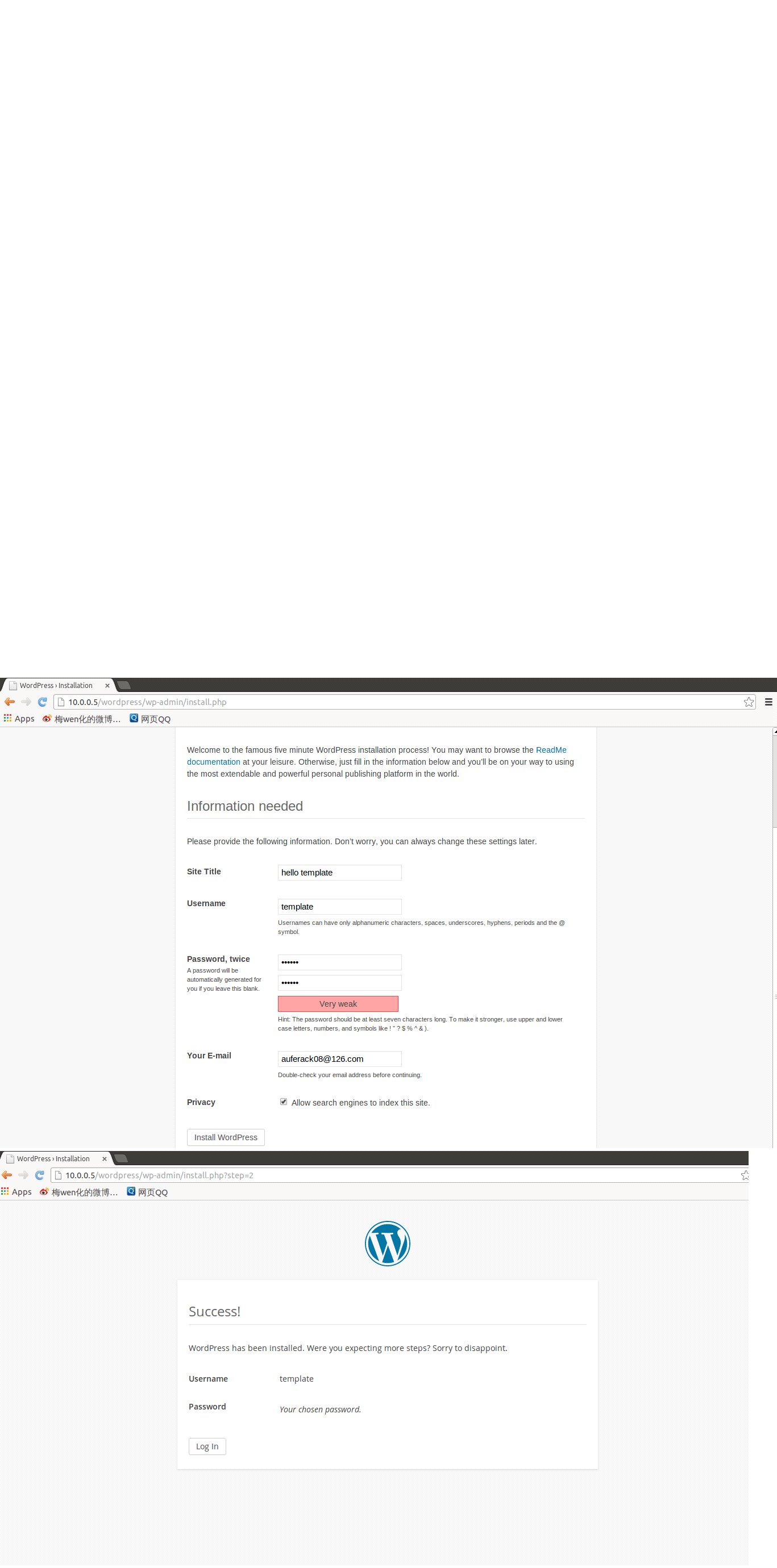


图17（输入信息）

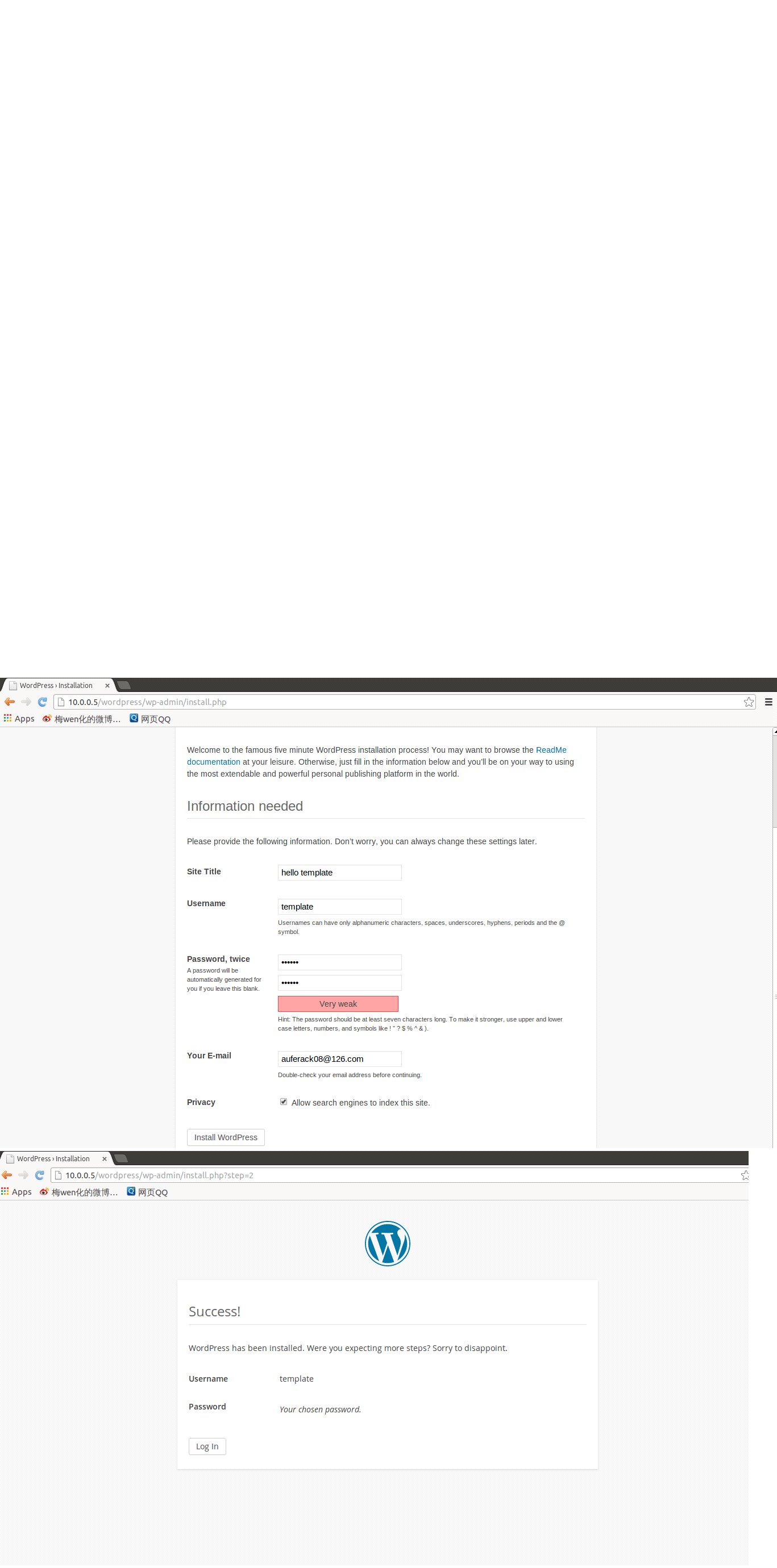


图18（安装完毕）

4.5.3 接着单击Log In登录界面如图19所示，输入相应的用户名和密码（步骤5.2中

的信息）登录进去。

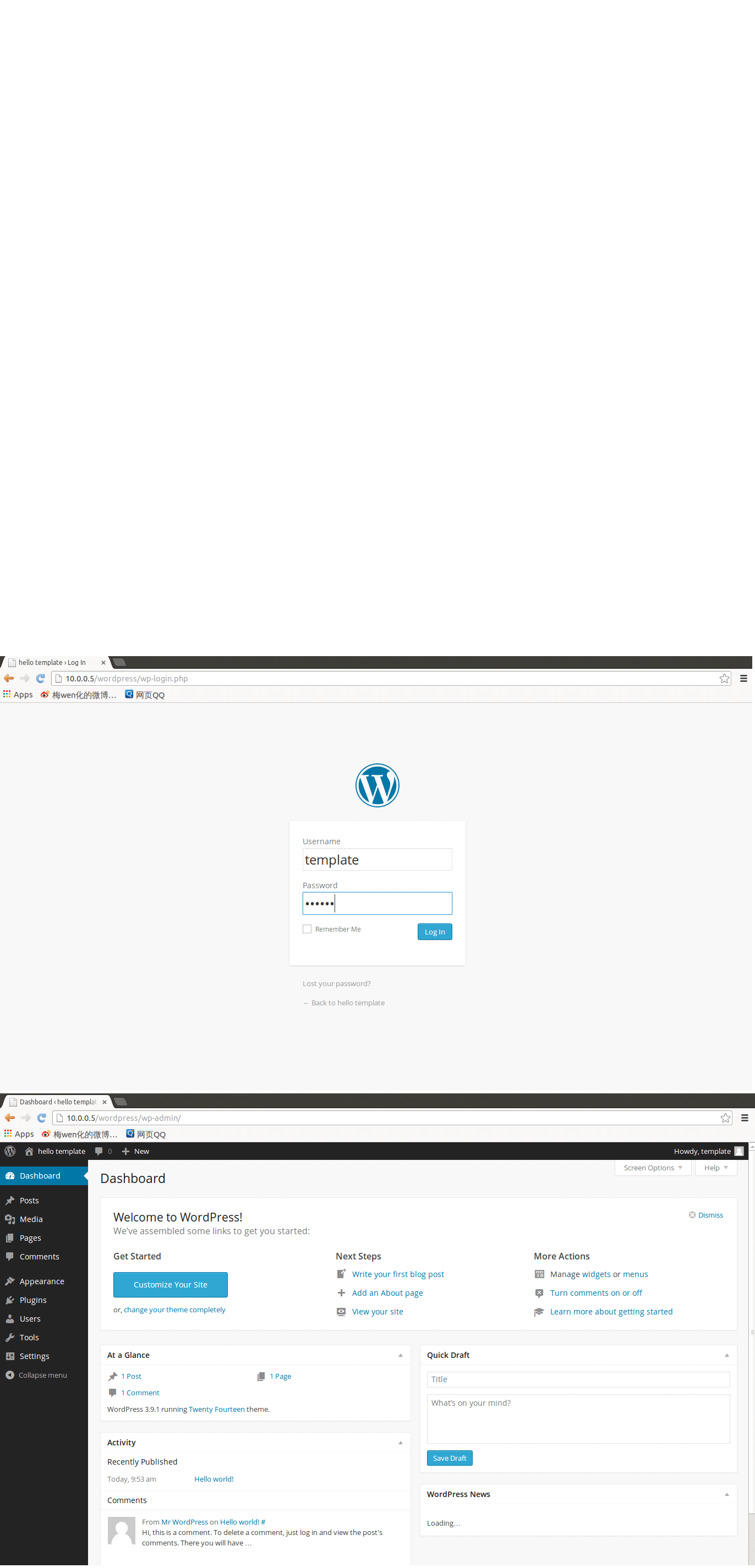


图19（登录界面）

4.5.4 登录进去之后可以看到整个wordpress的界面，如图20所示，至此，整个stack

的创建和应用的安装结束。

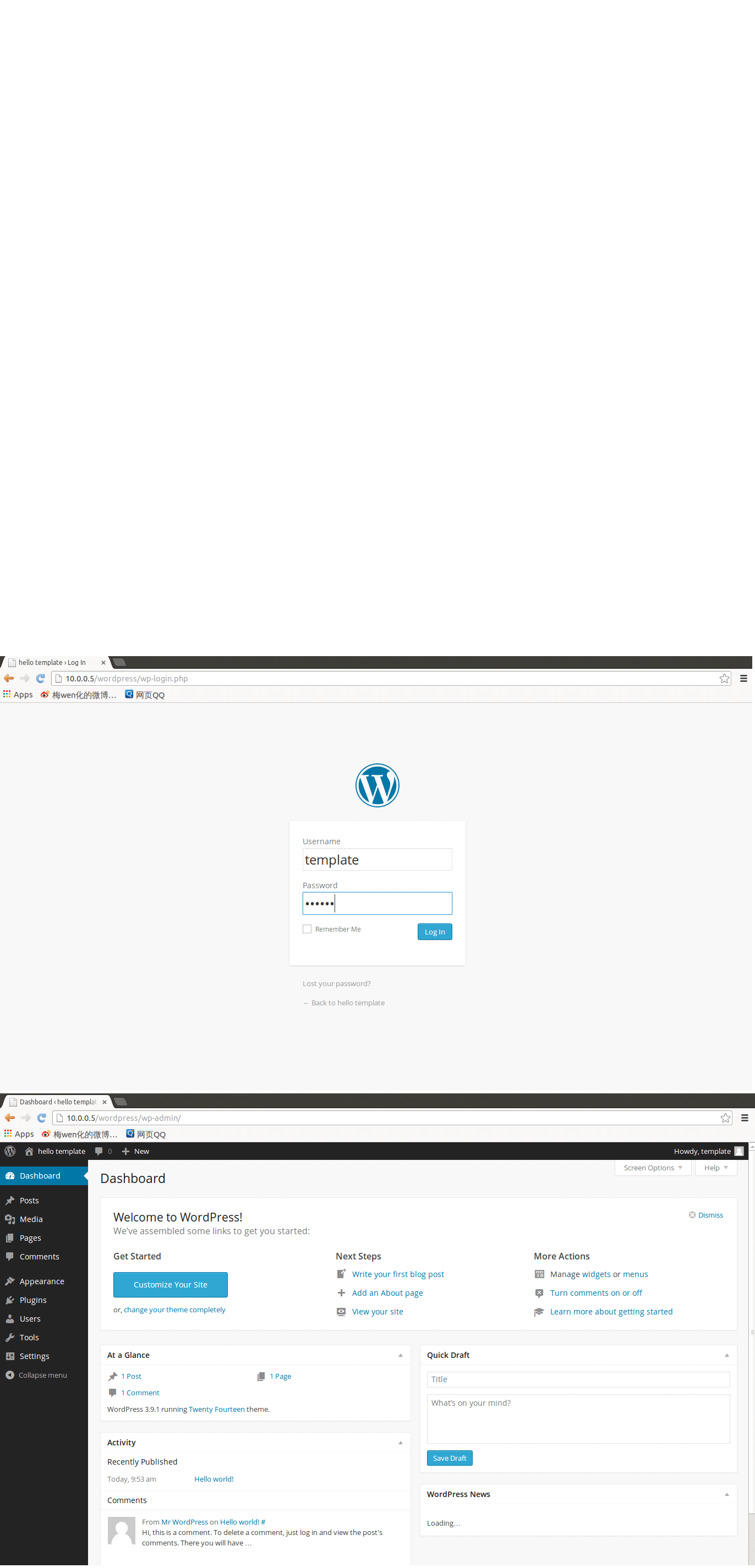


图20（wordpress应用界面）

5 数据库展示

数据库的展示就用命令从终端查看，截图如图21：

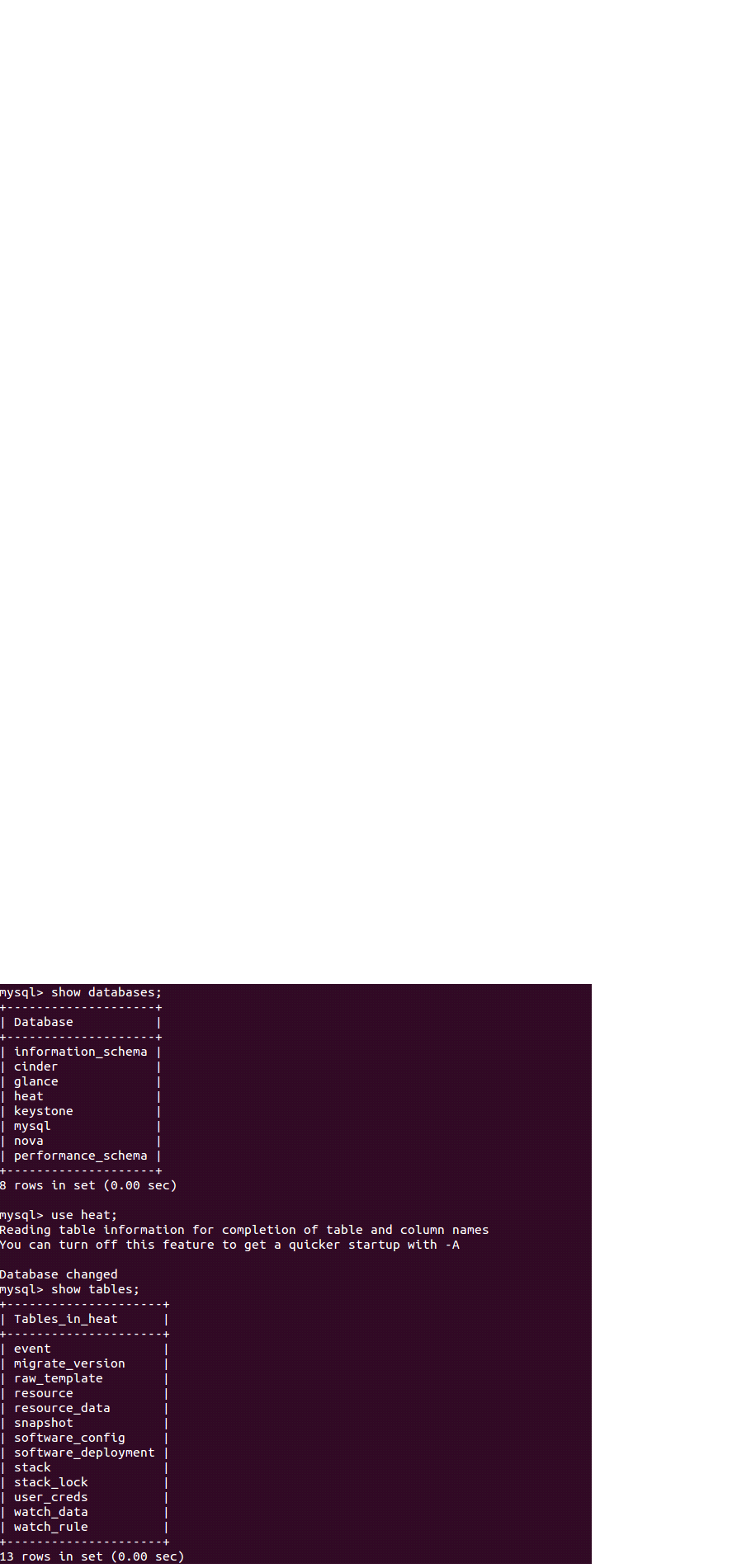


图21（数据库展示）

6 如何进行debug

6.1 首先在ubuntu中安装eclipse，并配置好PyDev，然后新建一个PyDev项目，

如图22所示：

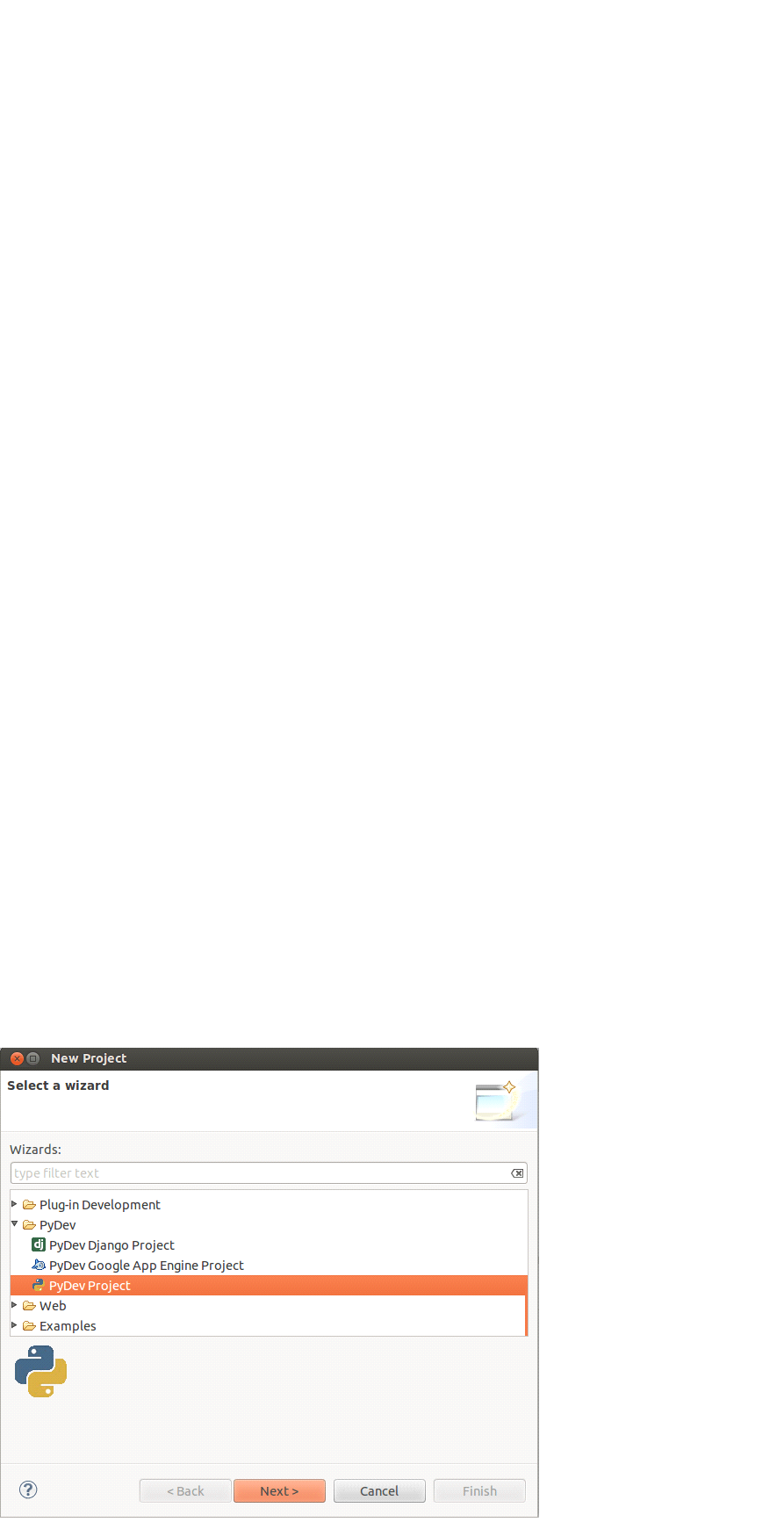
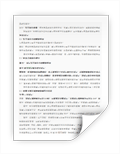


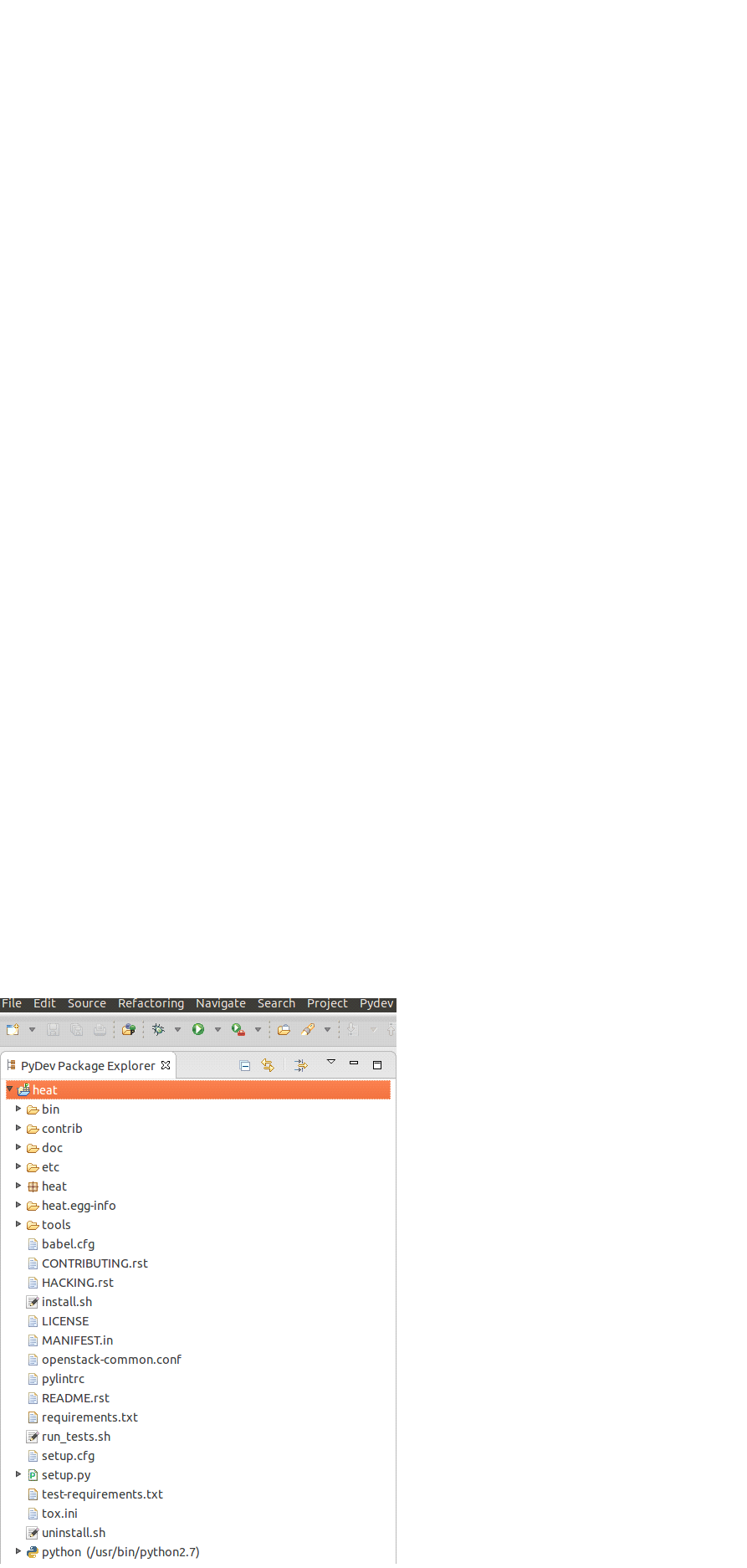
图22（创建PyDev项目）

6.2 创建好一个项目后将stack目录下的heat目录中的所有文件拷贝到heat项

目中，如图23所示：

**导读：**#用来对初始化模板时必须要提供的一些参数的说明，heat\_template\_version是版本的说明信息，（heat项目代码框架）6.3从终端查看heat服务，ps-ef|grepheat，如图24所示：图23图23（终端查看服务）6.4然后将heat-api服务对应的进程kill，然后从dashboard中查看stack-list，可以看到无法查看，证明服务已经被关闭，如图24所示：图24





（heat项目代码框架）

6.3 从终端查看heat服务，ps -ef|grep heat，如图24所示：图23

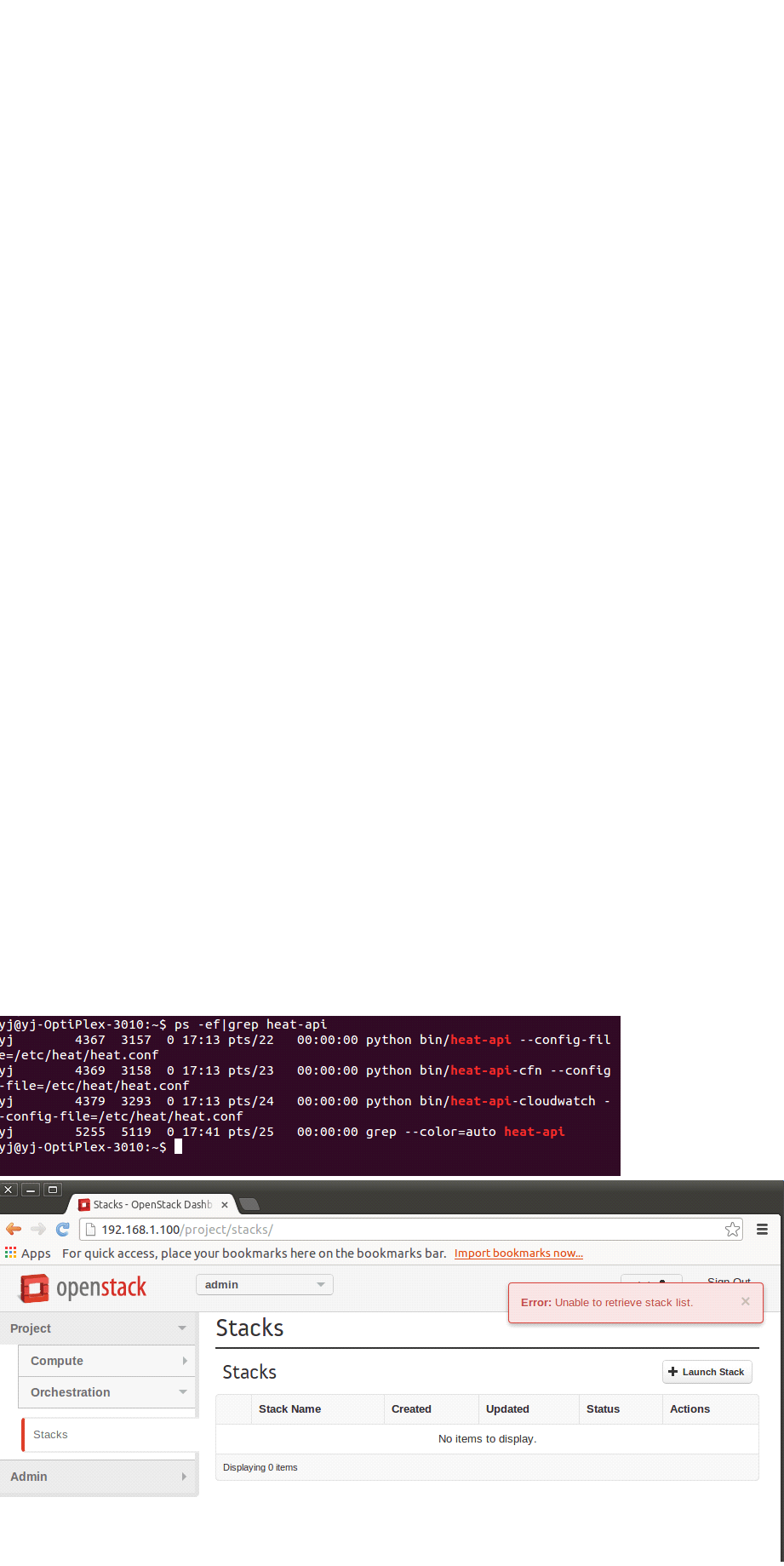


图23（终端查看服务）

6.4 然后将heat-api服务对应的进程kill，然后从dashboard中查看

stack-list，可以看到无法查看 ，证明服务已经被关闭，如图24所示：

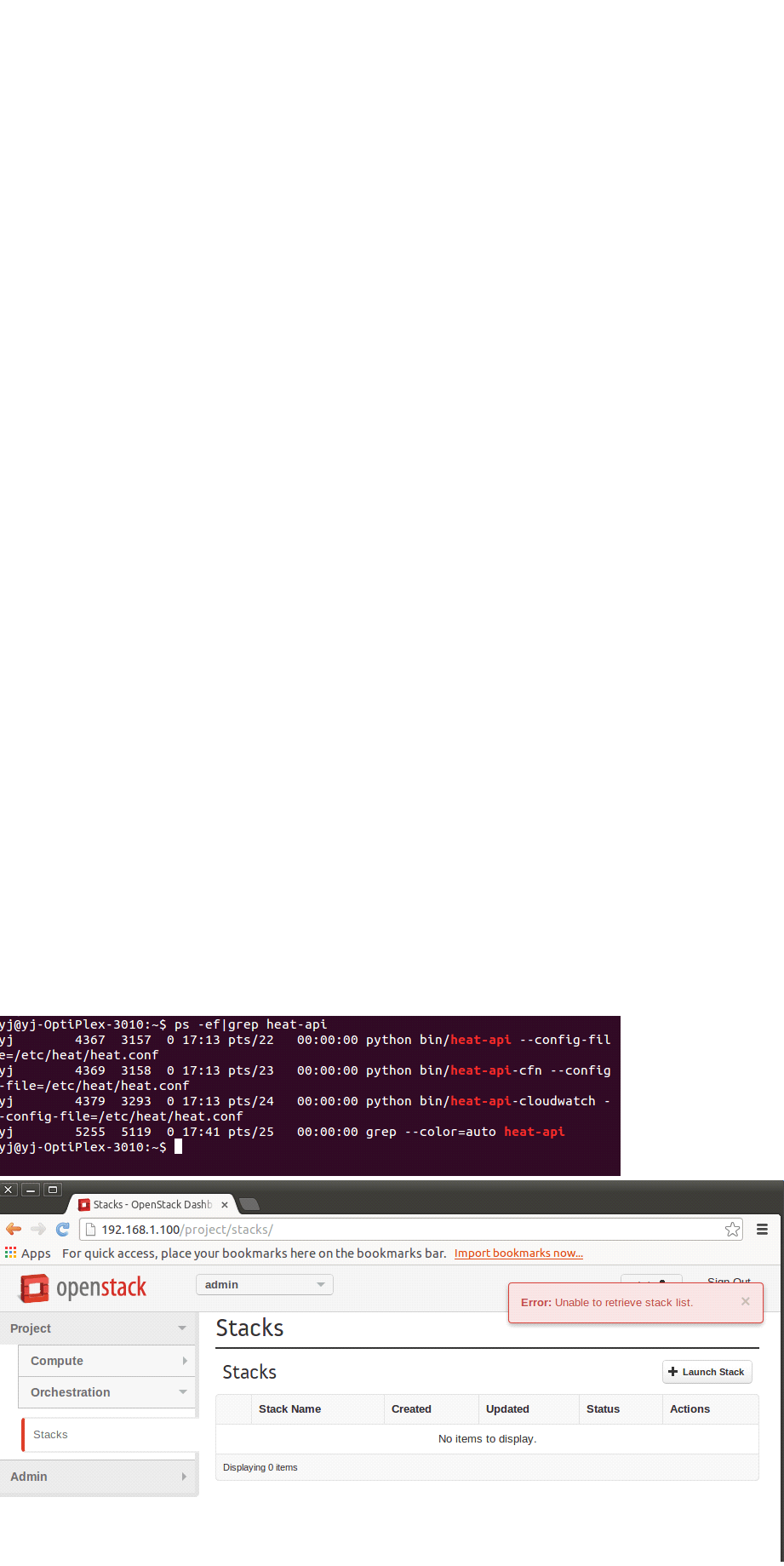


图24（查看stack list）

6.5 然后在eclipse中启动heat-api，如图25所示：

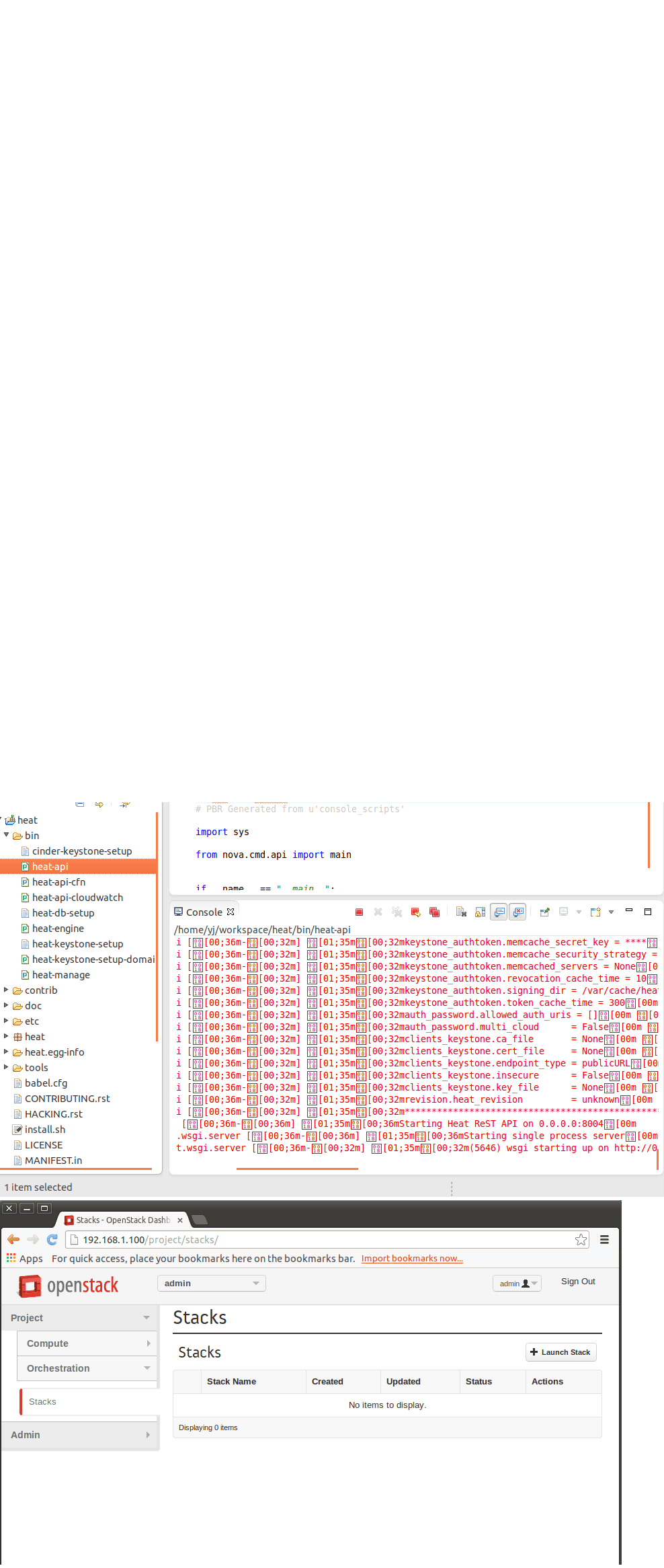


图25（eclipse启动heat服务）

6.6 然后在dashboard中继续查看heat服务是否启动成功，如图26所示：

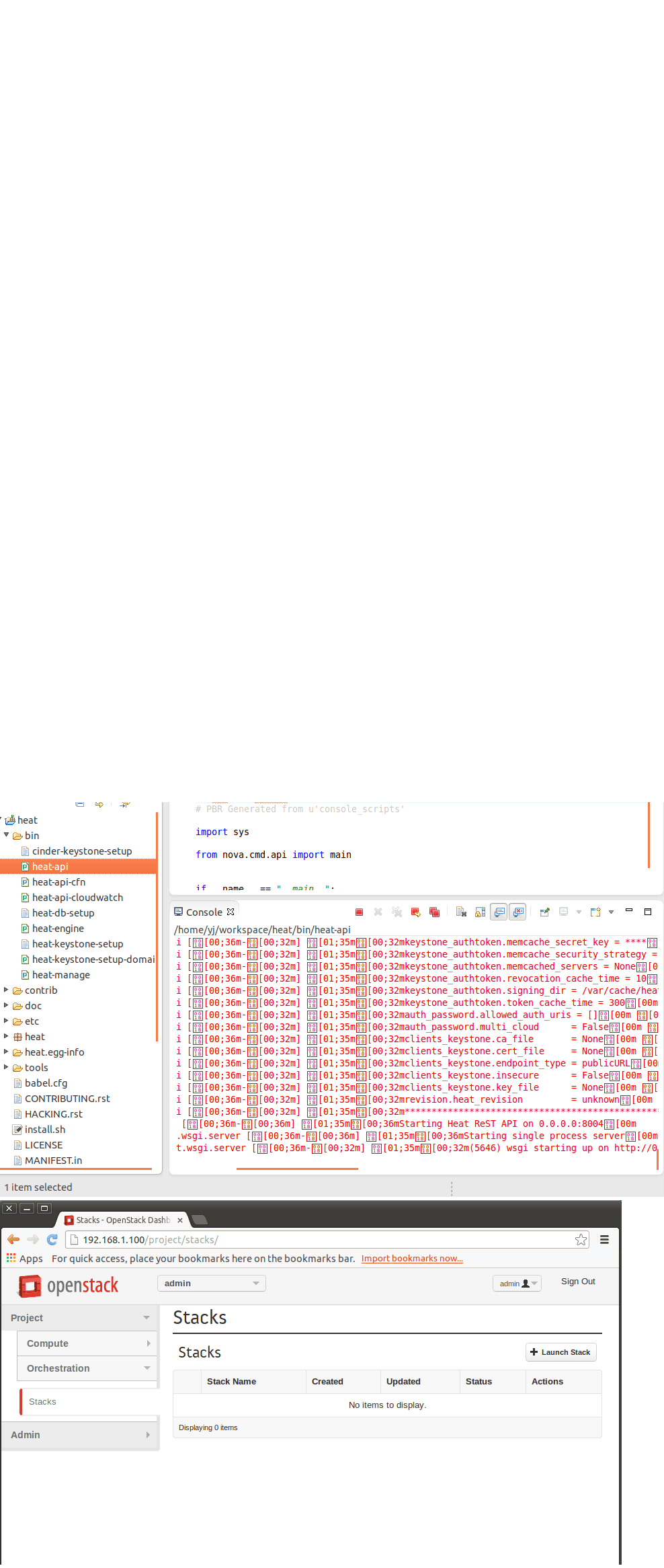


图26（dashboard界面查看heat服务是否启动）

7 Heat中template介绍

7.1 template的formatversion介绍

template 中有三个formatversion，第一种是aws的格式，表示这是一个json

文件；第二种是heat为了扩展aws并且更好的加入注释，引入的 yaml格式文件；第三种是openstack的heat特有的hot格式的头部，也是基于yaml格式的。 "AWSTemplateFormatVersion": "2010-09-09"

"HeatTemplateFormatVersion": "2012-12-12"

"heat\_template\_version": "2013-05-23 "

7.2 template的formatversion字段介绍

AWSTemplateFormatVersion和HeatTemplateFormatVersion支持的字段 ‘AWSTemplateFormatVersion’,

格式版本，指定了编写模板时依据的AWS CloudFormation模板版本

‘HeatTemplateFormatVersion’,

格式版本，指定了编写模板时依据的HeatTemplateFormation模板版本 ‘Description’,

可选描述，记录该模板的描述。

‘Mappings’,

可选映像，通过映像部分，可以声明条件值，可在Resources和Outputs部分使用内建函数Fn::FindInMap将与键对应的值返回到 Mappings 部分声明的双层映射中。 。

‘Parameters’,

可选条件，可以定义用于控制是否创建某些资源或者是否在堆栈创建或更新过程中为某些资

源属性分配值的条件。

‘Resources’,

资源部分将列出堆栈的成员资源。每项资源将予以分别列明,并指定创建此特定资源所必需的资源属性。可在资源和输出部分取消资源参考。

‘Outputs’

可选输出，在 Outputs 部分,您可以选择对响应 aws cloudformation

describe-stacks 命令而返回的自定义 值进行定义。这些输出值将包括基于文本、资源、参数、虚拟参数和内部函数的信息。

7.3 heat\_template\_version支持的字段

heat\_template\_version: 2013-05-23

# 2013-05-23或者是在这个日期之后的日期表明YAML被定义为是HOT模板

description:

# 用来描述模板的一些信息，通常可以在创建完模板之后看到。

parameter\_groups:

# 用来表明一些参数应该被怎样分组以及提供参数的方式，这部分是可选的，当没有输出值时可以省略。Parameters:

# 用来对初始化模板时必须要提供的一些参数的说明，是可选的，当没有输出值时可以省略。 Resources:

# 这部分包含模板单一资源的声明，必须包含至少一个应该在HOT模板中被定义的资源，否则模板不会做任何实例化的事。

Outputs:

# 这部分用于在模板被实例化后输出给用户可以使用参数信息，这部分是可选的，当没有输出值时可以省略。

7.4 heat\_template\_version详细介绍

原本是想把所有不同格式的模板都介绍一遍，但是目前openstack官网上只有对heat\_template\_version的详细介绍，加上不同的模板大致是相通的，所以这里就对绍heat\_template\_version模板详细的分析下。

heat\_template\_version是版本的说明信息，description是描述模板的信息，没有其他作用，这里就重点介绍后面的几个部分。

7.4.1 parameters\_groups

这些组被定义为一个列表，列表中每个组包含一个相关的参数列表。这些列表用于定义参数的期望顺序。每个参数应该跟只使用一次参数名称同时将这个参数绑定到parameters section中的参数定义部分的指定的组相关联。下面看下parameter\_groups具体内容：

parameter\_groups:

- label: <human-readable label of parameter group>

description: <description of the parameter group>

parameters:

- <param name>

- <param name>

label标签

一个定义与参数相关联的组的可读的标签。

description

这个属性允许给参数组一个可读的描述。

Parameters

与参数组相关联的参数列表。

Param name

在后面参数部分被定义的参数的名字。

7.4.2 Parameters 部分

参数部分允许为在实例化模板的时候指定输入参数，这些参数通常被用于自定义每个部署或者应用，例如设定自定义的用户名和密码。每个参数会在一个单独的小模块（嵌套块）中被定义，模块中第一行是参数的名称，其他的属性比如类型或者默认值将作为模块中其他的的元素。下面看下parameter的具体内容：

parameters:

<param name>:

type: <string | number | json | comma\_delimited\_list | boolean>

label: <human-readable name of the parameter>

description: <description of the parameter>

default: <default value for parameter>

hidden: <true | false>

constraints:

<parameter constraints>

param name

每个参数模块中所定义的参数的名字。

Type

这个属性是参数的类型。目前所支持的类型有

string,number,comma\_delimited\_list,json,和boolean。

Label

这个可选属性允许为参数提供一个可读的名字，通常省略。

Description

这个可选属性允许为参数提供一个可读的描述。

Default

这个可选属性是为了在部署或者应用中没有指定参数的值 时定义一个默认的值。

Hidden

这个可选属性允许当从模板运行时创建一个栈的时候隐藏参数的值，如果没有指明，默认值是false。

Constraints

这个可选属性允许指明其他的参数限制条件，比如数字参数的最大值或者最小值。

下面是一个关于两个参数定义的简单的例子，可以看出description和label实际上是可选的，但是为每个参数提供description和label是很好的做法。 Parameters:

user\_name:

type: string

label: User Name

description: User name to be configured for the application port\_number:

type: number

label: Port Number

description: Port number to be configured for the web server parameter constraints

参数定义的限制模块允许定义额外的参数值的限制条件，在模板实例化的时候，用户所提供的参数值会被这些条件去验证以确保用户所提供的值是符合用户的期望的，限制条件被定义在一个列表中，如下所示：

constraints:

- <constraint type>: <constraint definition>

description: <constraint description>

constraint type

限制条件的类型表明所定义的限制条件的种类

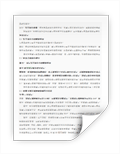
constraint definition

这个是根据限制类型所定义的具体的限制条件

description

这个可选属性允许指定当前限制条件的具体描述，例如，当所提供的输入值不符合限制条件，这个描述内容将会被呈现给用户（其实就是创建过程中如果出现错误，错误的具体描述），如果省略，一个默认的值将会呈现给用户。下面的例子是定义了一个带有两个限制条件的一个字符串参数，表明当尽管每个限制条件的描述是可选的，为每个应用部署时提供限制条件的描述是非

**导读：**常重要的。Parameters:user\_name:type:stringlabel:UserNamedescription:Usernametobeconfiguredfortheapplicationconstraints:-length:{min:6,max:8}description:Usernamemustbebetween6and8charact



常重要的。

Parameters:

user\_name:

type: string

label: User Name

description: User name to be configured for the application

constraints:

- length: { min: 6, max: 8 }

description: User name must be between 6 and 8 characters

- allowed\_pattern: "[A-Z]+[a-zA-Z0-9]\*"

description: User name must start with an uppercase character

length

长度的限制适用于字符串类型的参数，允许定义字符串参数值的最大长度和最小长度，语法格式如下：

length: { min: <lower limit>, max: <upper limit> }

range

范围的限制适用于数值类型的参数，允许定义数值类型的参数的最大值和最小值，语法格式如下：

range: { min: 0, max: 10 }

allowed\_values

allowed\_values限制条件适用于字符串或者数字类型的参数，指明一组可用的参数值，在部署的时候，用户提供的参数值必须在这一组参数值中。语法格式如下：

allowed\_values: [ <value>, <value>, ... ]或者可以使用下面的格式：

allowed\_values:

- <value>

- <value>

- ...

allowed\_pattern

允许模式限制条件适用于字符串参数，定义一个具体的正则表达式，用于匹配用户提供的参数值，语法格式如下：

allowed\_pattern: <regular expression>

custom\_constraint

custom\_constraint限制条件提供一个额外的验证步骤，通常用于检验后台中资源是否存在，custom\_constraint限制条件是通过插件的方式实现的，能够提供更高级的限制条件验证逻辑。语法格式如下：

custom\_constraint: <name>

Pseudo Parameters

模板中除了用户自定义的参数外，Heat也为每个stack创建了两个参数，一个是

“OS::stack\_name”，是stack的名字，另一个是“OS::stack\_id”，是stack的id，这两个值可以像用户自定义的参数一样通过“get\_param”内建函数被访问，

7.4.3 Resources Section

在资源部分，实际资源的模板将从被定义的HOT模板中构成一个stack的部署，每个资源在资源部分被定义成一个单独的块，语法格式如下： resources:

<resource ID>:

type: <resource type>

properties:

<property name>: <property value>

metadata:

<resource specific metadata>

depends\_on: <resource ID or list of ID>

update\_policy: <update policy>

deletion\_policy: <deletion policy>

resource ID

一个资源块是由资源ID所标识，是模板中资源部分独一无二的。 type

这个属性表明资源的类型，比如OS::Nova::Server.

properties

这个可选部分包含一个资源指定属性的列表，属性值可以在适当的位置提供或者通过函数提供。

metadata

这个可选部分包含资源类型的特定metadata。

depends\_on

这个可选属性允许指明当前资源跟其他资源的依赖关系。

update\_policy

该可选属性允许以嵌套字典的形式指明资源更新的策略，支持更新或者不更新策略，同时确切的语义依赖于当前的资源。

deletion\_policy

该可选属性允许为资源指明删除策略，具体哪种删除策略依赖于当前资源的类型。根据资源的类型，资源块可以包含更多的资源具体数据，基本上能被用于CFN模板的所有资源类型同样可以被用于HOT模板中，适用于上文所述YMAL结构。下面是一个简单的带有固定属性值的计算资源定义的例子： resources:

my\_instance:

type: OS::Nova::Server

properties:

flavor: m1.small

image: F18-x86\_64-cfntools

Resource Dependencies

通过在一个属性部分的depends\_on属性，Resource Dependencies能够定义资源与资源之间的依赖关系，如果一个资源仅仅依赖另外一个资源，那么另外一个资源的ID将作为depends\_on属性的值，如下例所示：

resources:

server1:

type: OS::Nova::Server

depends\_on: server2

server2:

type: OS::Nova::Server

如果一个资源依赖的资源不止一个，depends\_on属性值将以一个资源IDs列表的形式呈现，如下例所示：

resources:

server1:

type: OS::Nova::Server

depends\_on: [ server2, server3 ]

server2:

type: OS::Nova::Server

server3:

type: OS::Nova::Server

7.4.4 Outputs Section

在输出部分，任何向用户于提供可用输出参数将被定义，通常，参数可以是所部署的实例的IP地址或者所部署的栈的web应用实例的URLs，每个输出参数将在输出部分作为一个单独的块被定义，语法格式如下：

outputs:

<parameter name>:

description: <description>

value: <parameter value>

parameter name

一个输出参数块中主要的是输出参数的名字，这个输出参数名字在模板中的输出部分是独一无二的。

Desceiption

这个元素提供一个关于输出参数的简短的描述。

Parameter value

这部分是输出参数的值，通常是通过一个函数来解析去获得值，例如去获得一个stack的资源属性值。下面的例子表明一个计算资源的IP地址是如何作为一个输出参数： outputs:

instance\_ip:

description: IP address of the deployed compute instance

value: { get\_attr: [my\_instance, first\_address] }