

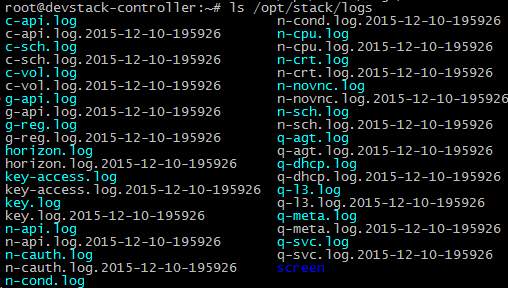
instance 从创建到删除的整个生命周期都是由 Nova 管理的。

后面各小节我们以 instance 生命周期中的不同操作场景为例，详细分析 Nova 不同组件如何协调工作，并通过日志分析加深大家对 Nova 的理解。

在研究 Nova 各个操作之前，我们先来学习一个重要的内容：OpenStack 日志。  
OpenStack 的日志记录了非常详细的细节信息，是我们学习和 troubleshoting 的利器。

**日志的位置**

我们实验环境使用的是 devstack，日志都统一放在 /opt/stack/logs 目录下，每个服务有自己的日志文件，从命名上很容易区分。



比如 nova-\* 各个子服务的日志都以 “n-” 开头：

n-api.log 是 nova-api 的日志

n-cpu.log 是 nova-compute 的日志。

Glance 的日志文件都是 “g-” 开头：

g-api.log 是 glance-api 的日志

g-reg.log 是 glance-registry 的日志。

Cinder、Neutron 的日志分别以 “c-” 和 “q-” 开头。

对于非 devstack 安装的 OpenStack，日志一般放在 /var/log/xxx/ 目录下。

比如 Nova 放在 /var/log/nova/ 下，Glance 放在/var/log/glance下……

各个子服务的日志文件也是单独保存，命名也很规范，容易区分。

比如 nova-api 的日志一般就命名为 /var/log/nova/api.log，其他日志类似。

**日志的格式**

OpenStack 的日志格式都是统一的，如下

<时间戳><日志等级><代码模块><Request ID><日志内容><源代码位置>

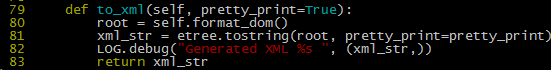
简单说明一下  
时间戳 日志记录的时间，包括 年 月 日 时 分 秒 毫秒  
日志等级 有INFO WARNING ERROR DEBUG等  
代码模块 当前运行的模块Request ID 日志会记录连续不同的操作，为了便于区分和增加可读性，每个操作都被分配唯一的Request ID,便于查找  
日志内容 这是日志的主体，记录当前正在执行的操作和结果等重要信息  
源代码位置 日志代码的位置，包括方法名称，源代码文件的目录位置和行号。这一项不是所有日志都有

下面举例说明

2015-12-10 20:46:49.566 DEBUG nova.virt.libvirt.config [req-5c973fff-e9ba-4317-bfd9-76678cc96584 None None] Generated XML ('<cpu>\n  <arch>x86\_64</arch>\n  <model>Westmere</model>\n  <vendor>Intel</vendor>\n  <topology sockets="2" cores="3" threads="1"/>\n  <feature name="avx"/>\n  <feature name="ds"/>\n  <feature name="ht"/>\n  <feature name="hypervisor"/>\n  <feature name="osxsave"/>\n  <feature name="pclmuldq"/>\n  <feature name="rdtscp"/>\n  <feature name="ss"/>\n  <feature name="vme"/>\n  <feature name="xsave"/>\n</cpu>\n',) to\_xml /opt/stack/nova/nova/virt/libvirt/config.py:82

这条日志我们可以得知：

1. 代码模块是 nova.virt.libvirt.config，由此可知应该是 Hypervisor Libvirt 相关的操作
2. 日志内容是生成 XML
3. 如果要跟踪源代码，可以到 /opt/stack/nova/nova/virt/libvirt/config.py 的 82 行，方法是 to\_xml



又例如下面这条日志：

2015-12-10 20:46:49.671 ERROR nova.compute.manager [req-5c973fff-e9ba-4317-bfd9-76678cc96584 None None] No compute node record for host devstack-controller

这条日志我们可以得知：

1. 这是一个 ERROR 日志
2. 具体内容是 “No compute node record for host devstack-controller”
3. 该日志没有指明源代码位置

**关于日志的几点说明**

1. 学习 OpenStack 需要看日志吗？这个问题的答案取决于你是谁。
2. 如果你只是 OpenStack 的最终用户，那么日志对你不重要。你只需要在 GUI上 操作，如果出问题直接找管理员就可以了。
3. 但如果你是 OpenStack 的运维和管理人员，日志对你就非常重要了。因为 OpenStack 操作如果出错，GUI 上给出的错误信息是非常笼统和简要的，日志则提供了大量的线索，特别是当 debug 选项打开之后。
4. 如果你正处于 OpenStack 的学习阶段，正如我们现在的状态，那么也强烈建议你多看日志。日志能够帮助你更加深入理解 OpenStack 的运行机制。
5. 日志能够帮助我们深入学习 OpenStack 和排查问题。但要想高效的使用日志还得有个前提：
6. 必须先掌握 OpenStack 的运行机制，然后针对性的查看日志。
7. 就拿 Instance Launch 操作来说，如果之前不了解 nova-\* 各子服务在操作中的协作关系，如果没有理解流程图，面对如此多和分散的日志文件，我们也很难下手不是。
8. 对于 OpenStack 的运维和管理员来说，在大部分情况下，我们都不需要看源代码。
9. 因为 OpenStack 的日志记录得很详细了，足以帮助我们分析和定位问题。
10. 但还是有一些细节日志没有记录，必要时可以通过查看源代码理解得更清楚。
11. 即便如此，日志也会为我们提供源代码查看的线索，不需要我们大海捞针。
12. 这一点我们会在后面的操作分析中看到。