

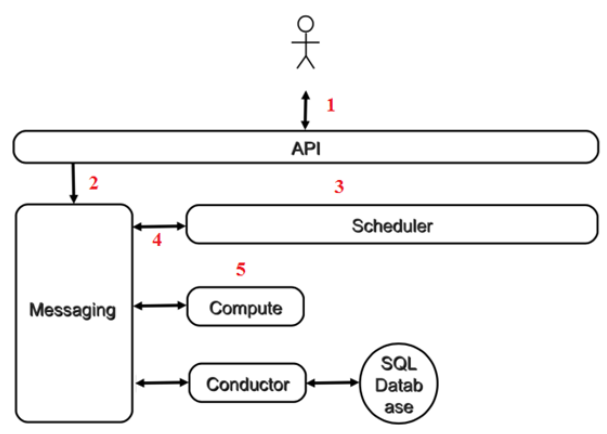
本节详细分析 instance launch 和 shut off 操作，以及如何在日志中快速定位有用信息的技巧。

**Launch**

Launch instance 应该算 Nova 最重要的操作。

仔细研究 lanuch 操作能够帮助我们充分理解 Nova 各个子服务的协调配合和运行机制。

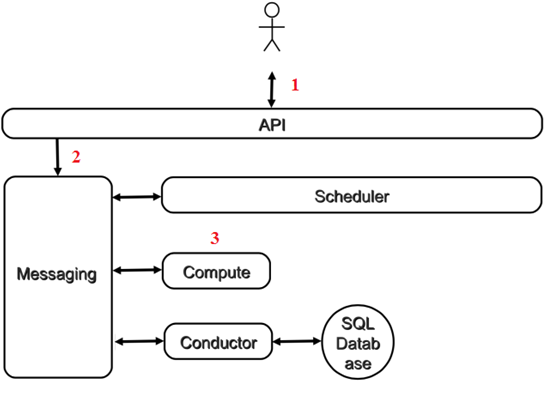
前面我们已经以 launch 操作为例详细讨论了各个 nova-\* 子服务。 这里不再赘述，只是再回顾一下流程。



1. 客户（可以是 OpenStack 最终用户，也可以是其他程序）向 API（nova-api）发送请求：“帮我创建一个 Instance”
2. API对请求做一些必要处理后，向 Messaging（RabbitMQ）发送了一条消息：“让 Scheduler 创建一个 Instance”
3. Scheduler（nova-scheduler）从 Messaging 获取到 API 发给它的消息，然后执行调度算法，从若干计算节点中选出节点 A
4. Scheduler 向 Messaging 发送了一条消息：“在计算节点 A 上创建这个 Instance”
5. 计算节点 A 的 Compute（nova-compute）从 Messaging 中获取到 Scheduler 发给它的消息，然后通过本节点的 Hypervisor Driver 创建 Instance。
6. 在 Instance 创建的过程中，Compute 如果需要查询或更新数据库信息，会通过 Messaging 向 Conductor（nova-conductor）发送消息，Conductor 负责数据库访问。

**Shut Off**

下面是 shut off instance 的流程图

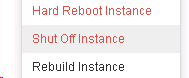


1. 向 nova-api 发送请求
2. nova-api 发送消息
3. nova-compute 执行操作

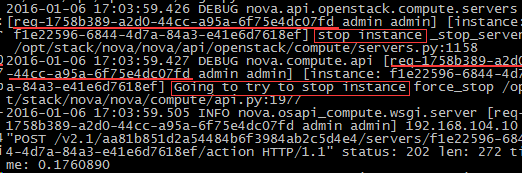
下面我们详细讨论每一个步骤。

**向 nova-api 发送请求**

客户（可以是 OpenStack 最终用户，也可以是其他程序）向 API（nova-api）发送请求：“帮我关闭这个 Instance”



查看日志 /opt/stack/logs/n-api.log



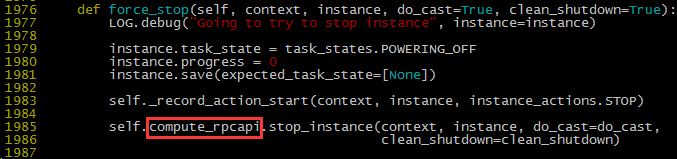
对于如何在日志文件中快速查找到有用的信息这里多聊几句。 对于初学者，这不是一件容易的事情，因为日志里条目和内容很多，特别是 debug 选项打开之后，容易让人眼花缭乱，无从下手。

这里给大家几个小窍门：

1. 先确定大的范围，比如在操作之前用 tail -f 打印日志文件，这样需要查看的日志肯定在操作之后的打印输出的这些内容里。 另外也可以通过时间戳来确定需要的日志范围。
2. 利用 “代码模块” 快速定位有用的信息。 nova-\* 子服务都有自己特定的代码模块：  
   **nova-api**   
   nova.api.openstack.compute.servers   
   nova.compute.api   
   nova.api.openstack.wsgi  
     
   **nova-compute**nova.compute.manager   
   nova.virt.libvirt.\*  
     
   **nova-scheduler**nova.scheduler.\*
3. 利用 Request ID 查找相关的日志信息。 在上面的日志中个，我们可以利用 “req-1758b389-a2d0-44cc-a95a-6f75e4dc07fd” 这个 Request ID 快速定位 n-api.log 中相与 shut off 操作的其他日志条目。 需要补充说明的是，Request ID 是跨日志文件的，这一个特性能帮助我们在其他子服务的日志文件中找到相关信息，我们后面马上将会看到这个技巧的应用。

**nova-api 发送消息**

nova-api 向 Messaging（RabbitMQ）发送了一条消息：“关闭这个 Instance” nova-api 没有将发送消息的操作记录到日志中，不过我们可以通过查看源代码来验证。 一提到源代码，大家可能以为要大海捞针了。其实很简单，上面日志已经清楚地告诉我们需要查看的源代码在 /opt/stack/nova/nova/compute/api.py 的 1977 行，方法是 force\_stop。



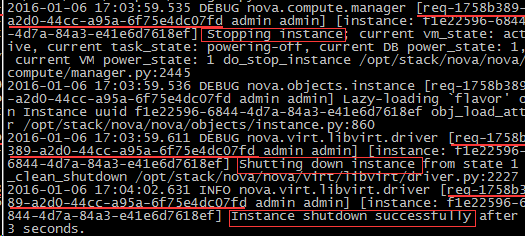
force\_stop 方法最后调用的是对象 self.compute\_rpcapi 的 stop\_instance 方法。 在 OpenStack 源码中，以 xxx\_rpcapi 命名的对象，表示的就是 xxx 的消息队列。 xxx\_rpcapi.yyy() 方法则表示向 xxx 的消息队列发送 yyy 操作的消息。

所以 self.compute\_rpcapi.stop\_instance() 的作用就是向 RabbitMQ 上 nova-compute 的消息队列里发送一条 stop instance 的消息。

这里补充说明一下： 关闭 instance 的前提是 instance 当前已经在某个计算节点上运行，所以这里不需要 nova-scheduler 再帮我们挑选合适的节点，这个跟 launch 操作不同。

**nova-compute 执行操作**

查看计算节点上的日志 /opt/stack/logs/n-cpu.log



这里我们利用了 Request ID “req-1758b389-a2d0-44cc-a95a-6f75e4dc07fd” 在 n-cpu.log 中快速定位到 nova-compute 关闭 instance 的日志条目。

**小结**

分析某个操作时，我们首先要理清该操作的内部流程，然后再到相应的节点上去查看日志。 例如shut off 的流程为：

1. 向 nova-api 发送请求
2. nova-api 发送消息
3. nova-compute 执行操作

1，2 两个步骤是在控制节点上执行的，查看 nova-api 的日志。 第 3 步是在计算节点上执行的，查看 nova-compute 的日志。