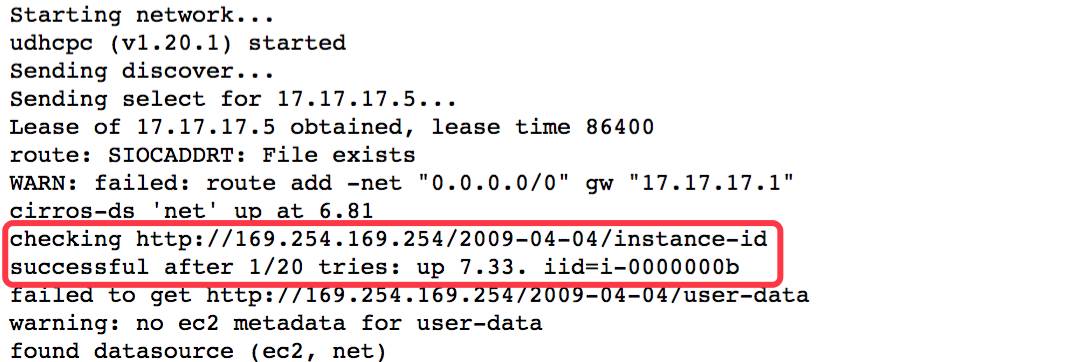
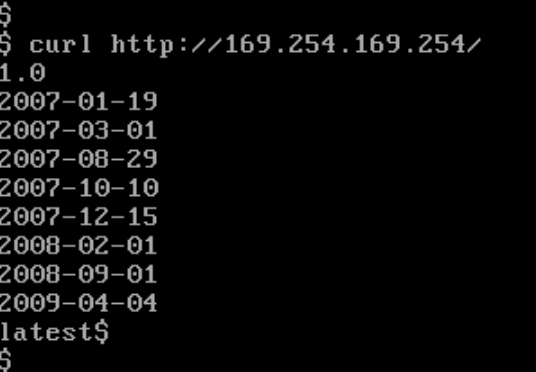
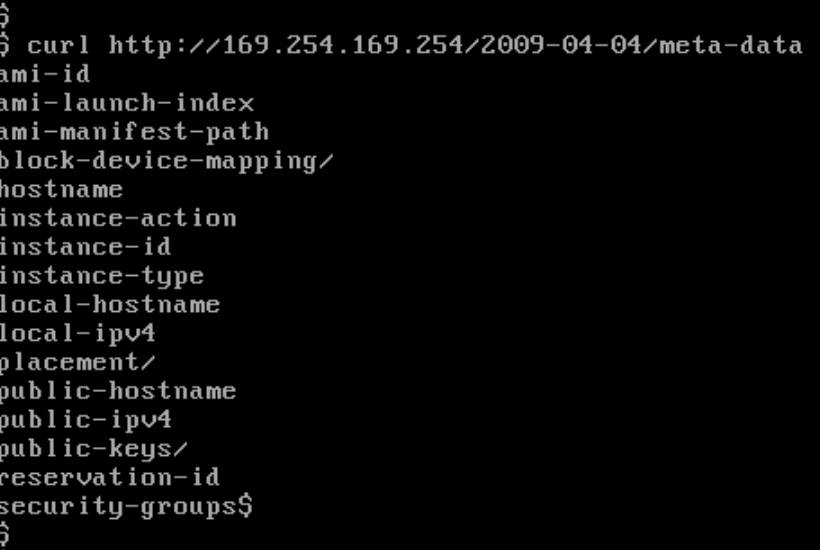
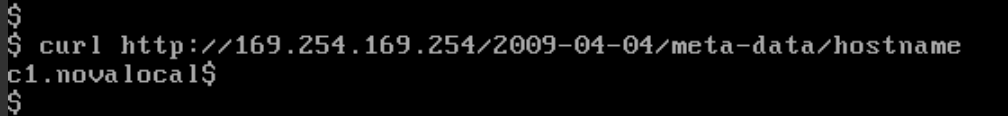
接上节，启动 neutron router 后 instance c1 终于拿到了 metadata, 从下面 c1 的启动日志可知：



c1 所认为的 metadata 服务地址是 169.254.169.254，端口为 80。我们在 c1 中尝试访问一下 metadata。

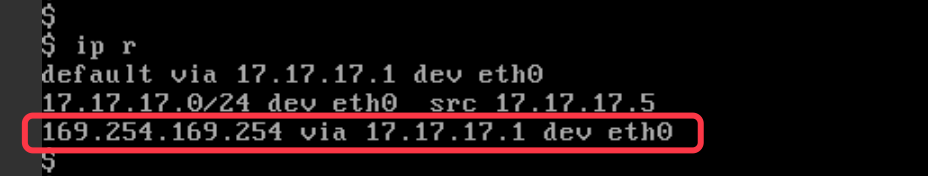




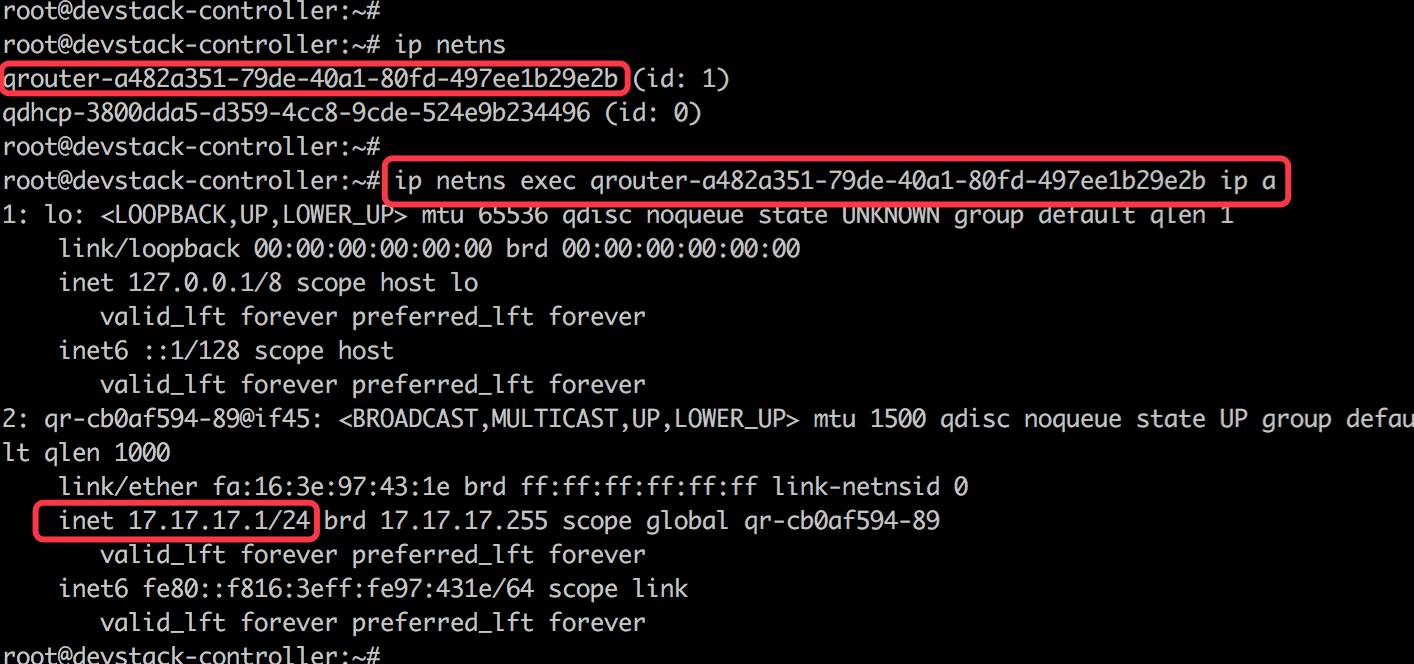


确实能够拿到 metadata。但我们知道 nova-api-metadata 是运行在控制节点上的，IP并不是 169.254.169.254，这是怎么实现的呢？下面我们分析一下这个过程。

从 c1 的路由表得访问 169.254.169.254 的请求会走 17.17.17.1。

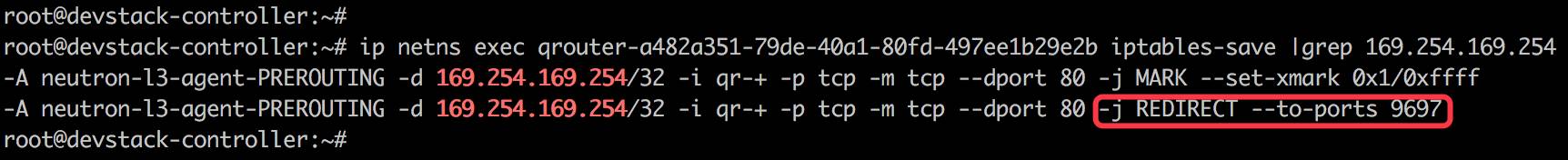


17.17.17.1 实际上就是 test\_router 在 test\_net 上的 interface IP。这条路由是 OpenStack 自动添加到 instance 中的，这样就将 metadata 的请求转发到 neutron router。

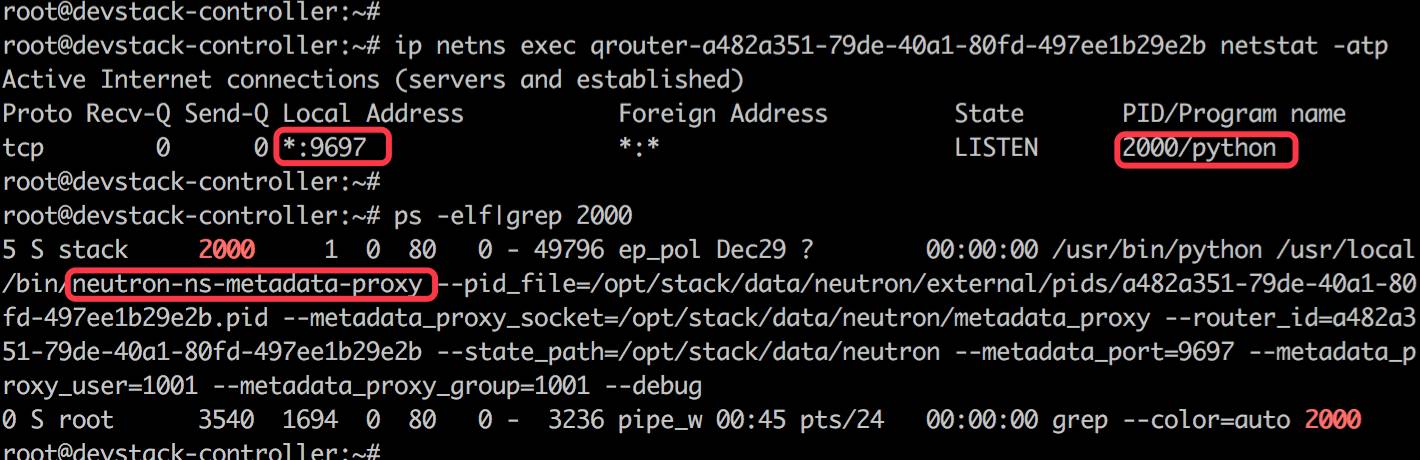


ip netns 是管理 linux network namespace 的命令，如果对 namespace 不熟悉，可参考教程前面相关章节。

test\_router 接收到 c1 的请求，会通过 iptable 规则转发到 9697 端口。



9697 端口是干嘛的？这是 neutron-ns-metadata-proxy 的监听端口。



到这里我们可以把思路重新理一下了：

1. instance 通过预定义的 169.254.169.254 请求 metadata。
2. 请求被转发到 neutron router。
3. router 将请求转发给 neutron-ns-metadata-proxy。
4. 再后面就简单了：neutron-ns-metadata-proxy 将请求通过 unix domain socket 发给 neutron-metadata-agent，后者再通过管理网络发给 nova-api-metadata。

OpenStack 默认通过 l3-agent 创建和管理 neutron-ns-metadata-proxy。但不是所有环境都有 l3-agent，比如直接用物理 router 的场景。这时就需要让 dhcp-agent 来管理 neutron-ns-metadata-proxy。  
  
下一节我们分析 dhcp-agent 如何处理 metadata 请求。