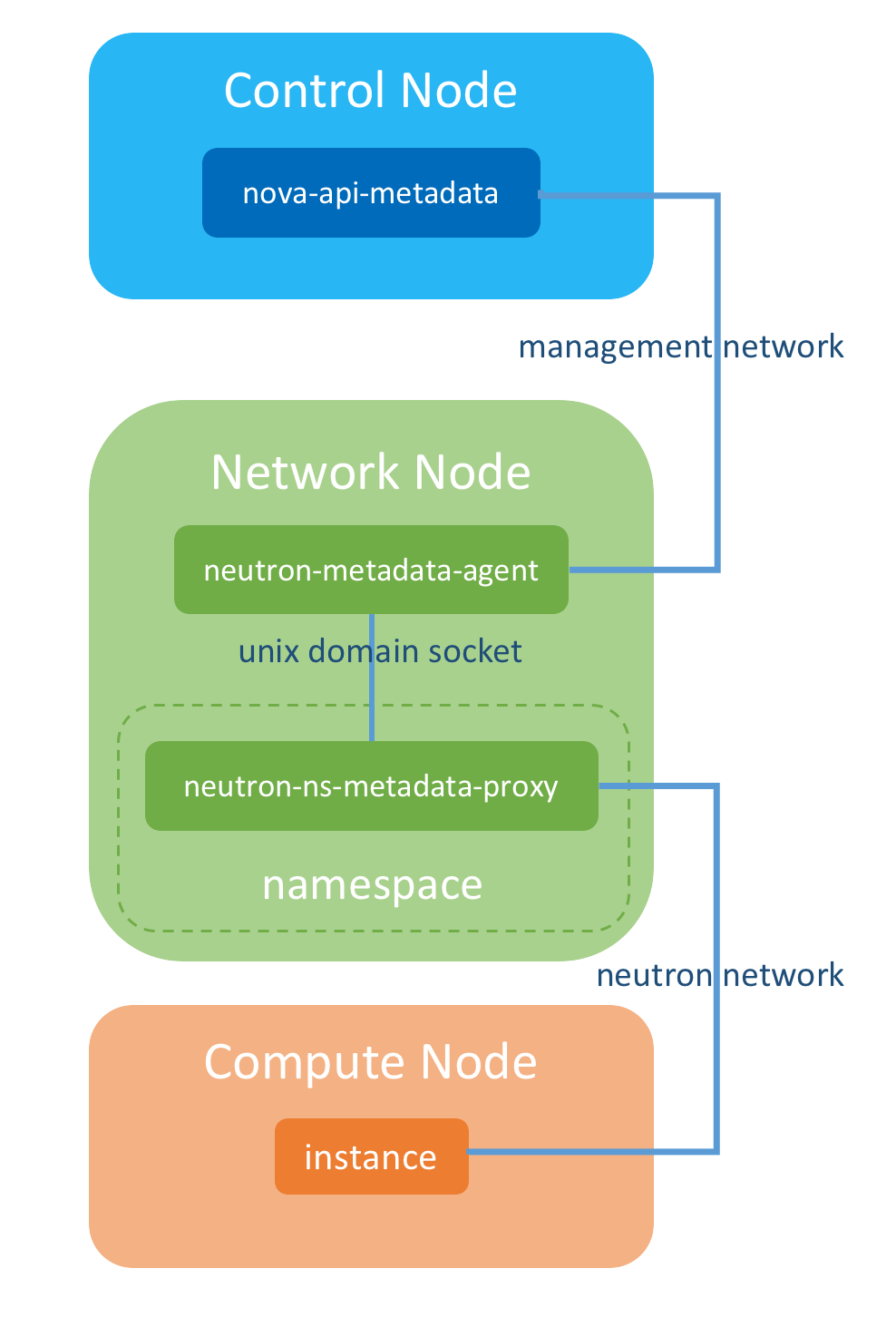
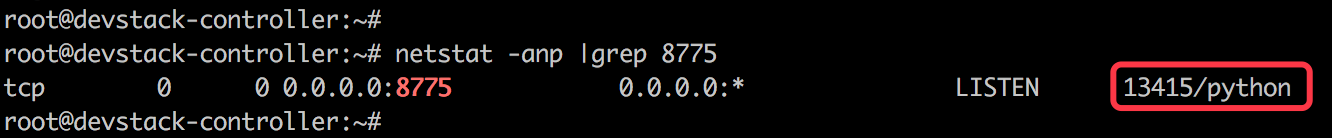
下面是 Metadata Service 的架构图，本节我们详细讨论各个组件以及它们之间的关系。

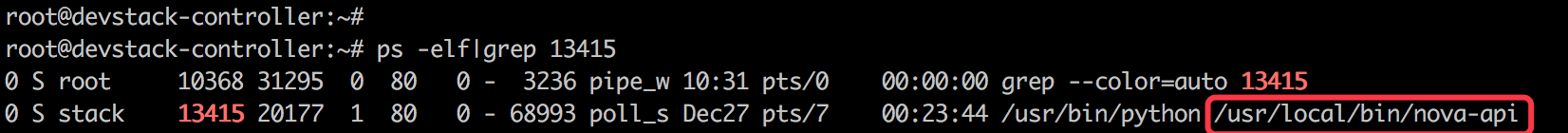


**nova-api-metadata**

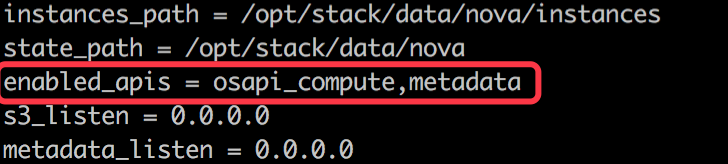
nova-api-metadata 是 nova-api 的一个子服务，它是 metadata 的提供者，instance 可以通过 nova-api-metadata 的 REST API 来获取 metadata 信息。  
  
nova-api-metadata 运行在控制节点上，服务端口是 8775。



通过进程 ID 13415 查看该启动程序。



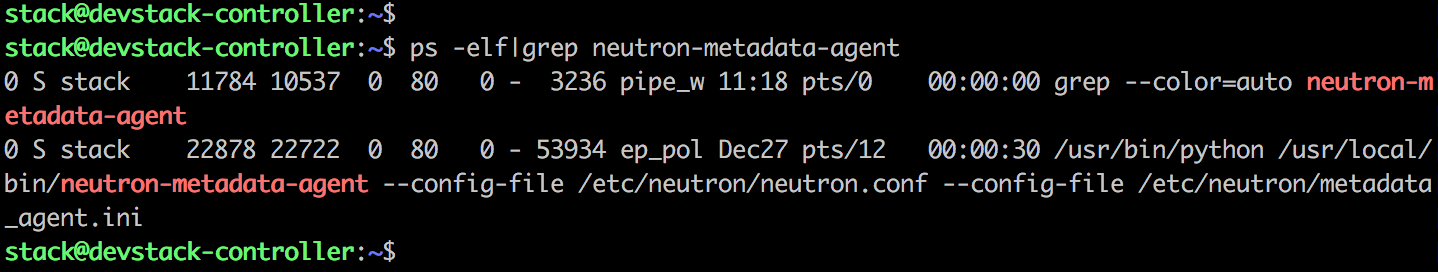
我们这个环境是 devstack，nova-api-metadata 的程序名称就是 nova-api，nova-api-metadata 与常规的 nova-api 服务是合并在一起的。不过在 OpenStack 的其他发行版中可能有单独的 nova-api-metadata 进程存在。  
  
nova.conf 通过参数 enabled\_apis 指定是否启用 nova-api-metadata。



osapi\_compute 是常规的 nova-api 服务，metadata 就是 nova-api-metadata 服务。

**neutron-metadata-agent**

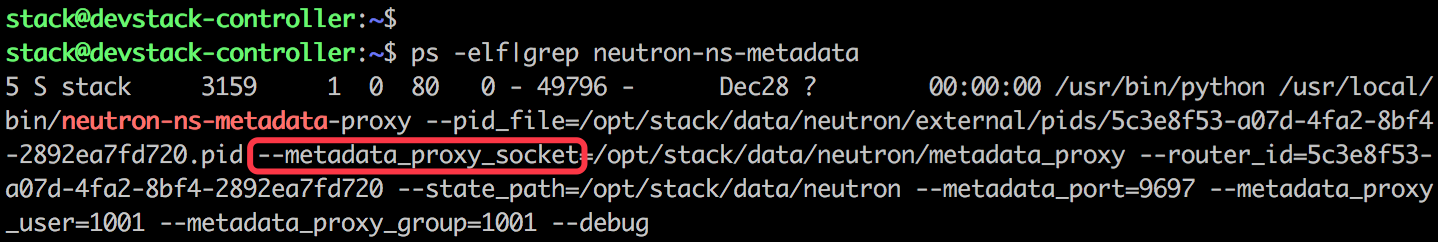
nova-api-metadata 在控制节点上，走 OpenStack 内部管理网络，instance 是无法通过 http://controller\_ip:8775 直接访问 metadata service 的，因为网络不通。  
  
那怎么办呢？  
  
答案是：借助 neutron-metadata-agent。  
  
neutron-metadata-agent 运行在网络节点上。instance 先将 metadata 请求发给 neutron-metadata-agent，neutron-metadata-agent 再将请求转发到 nova-api-metadata。



这里还有个问题需要解释清楚：instance 如何将请求发送到 neutron-metadata-agent？  
  
实际上 instance 是不能直接与 neutron-metadata-agent 通信的，因为 neutron-metadata-agent 也是在 OpenStack 内部管理网络上的。不过好在网络节点上有另外两个组件，dhcp agent 和 l3 agent，它们两兄弟与 instance 可以位于同一 OpenStack network 中，这样就引出了下一个组件： neutron-ns-metadata-proxy。

**neutron-ns-metadata-proxy**

neutron-ns-metadata-proxy 是由 dhcp-agent 或者 l3-agent 创建的，也运行在网络节点。更精确的说法是：运行在网络节点的 namespace 中。  
  
如果由 dhcp-agent 创建，neutron-ns-metadata-proxy 就运行在 dhcp-agent 所在的 namespace 中；如果由 l3-agent 创建，neutron-ns-metadata-proxy 就运行在 neutron router 所在的 namespace 中。“neutron-ns-metadata-proxy” 中间的 ns 就是 namespace 的意思。neutron-ns-metadata-proxy 与 neutron-metadata-agent 通过 unix domain socket 直接相连。



这样整个链路就打通了：  
  
1. instance 通过 neutron network（Project 网络）将 metadata 请求发送到 neutron-ns-metadata-proxy。  
  
2. neutron-ns-metadata-proxy 通过 unix domain socket 将请求发给 neutron-metadata-agent。  
  
3. neutron-metadata-agent 通过内部管理网络将请求发送给 nova-api-metadata。  
  
可能大家对于 neutron-ns-metadata-proxy 还会有些疑虑：既然 dhcp-agent 和 l3-agent 都可以创建和管理 neutron-ns-metadata-proxy，使用的时候该如何选择呢？  
  
简单的说：各有各的使用场景，并且两种方案可以共存。大家不用担心，后面我们会通过例子详细讨论。  
  
Metadata Service 的架构已经讨论清楚了，下一节将通过实践加深理解