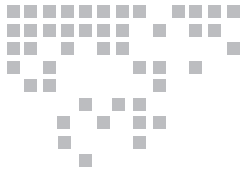

Chapter 3 第 3 章

OpenStack 集群——云控制器和公共服务

“独行欲速，同行致远。”

——非洲谚语

在前面的章节中，我们了解了如何通过自动化方式部署一个大型 OpenStack 基础架构。现在，我们来更深入地学习 OpenStack 中的各种概念性设计。在一个大型基础架构中，如果想要所有服务都一直运行着，我们必须确保 OpenStack 基础架构是可靠的，是能够保证业务连续性的。我们已经在第 1 章中讨论了几个与设计相关的方面，并重点介绍了 OpenStack 中可扩展架构模型的一些最佳实践。然后，我们还讨论了自动化的效率问题，还使用 OpenStack Ansible 部署工具在一个一体化（all-in-one）环境中配置了云控制节点和计算节点。

本章一开始会介绍一些集群知识。它将会引导大家更深入地学习 OpenStack 设计模式，这些模式在聚焦于云控制器的同时，还将功能分解为不同的服务。请注意，本章不会详细介绍高可用性，也不会涵盖所有 OpenStack 服务层；相反，本章将概述 OpenStack 集群设计的几种可能性。在强调标准化和持续 IT 构建的 OpenStack 环境中，集群是一项非常关键的基础性技术。

本章主要包括以下内容：

- ❑ 简要介绍集群的概念。
- ❑ 介绍 OpenStack 环境中的云控制器和公共服务。
- ❑ 基于控制节点上的各服务的功能来介绍其他 OpenStack 集群模型。
- ❑ 理解公共基础架构服务。

- 学习 OpenStack 控制节点中的各种服务。
- 掌握使用 Ansible 来自动化 OpenStack 基础架构部署。

3.1　集群核心概念

您不要害怕声称在一个给定的基础架构中，实际上是集群（clustering）在提供高可用性。两台或多台服务器的能力聚合（capacity aggregation）就是服务器集群。通过堆积机器这种方式，就能实现这种聚合。

不要混淆了向上扩展（scale up，也称为垂直扩展（vertical scaling））和向下扩展（scaling down，也称为水平扩展（horizonal scaling））这两个概念。水平扩展是指增加更多的标准商用服务器，而垂直扩展是指向服务器添加更多的 CPU 和内存。

我们还需要区分高可用性（high availability）、负载均衡（load balancing）和故障切换（failing over）这几个术语。这些概念将在第 9 章进行详细介绍。请记住这一点，对于前面提到的所有术语，它们的配置往往是从集群概念开始的。我们将在下一节中了解如何区分它们。

3.1.1　非对称集群

非对称集群（asymmetric clustering）通常用于高可用目的，比如扩展数据库、消息系统或者文件系统的读写能力。在这种系统中，备服务器（standby server）只有在主服务器（master server）发生故障时才会接管系统。备服务器是个沉睡中的观察者（sleepy watcher），它包含了故障切换配置。

3.1.2　对称集群

在对称集群（symmetric clustering）中，所有节点都是活动的（active），都参与请求处理。因为所有节点都服务于应用和用户，这种配置往往具备成本效益。故障节点会被集群丢弃，其他节点将接管其工作负载并继续处理事务。可以将对称集群视为类似于负载均衡集群。在云基础架构中，负载均衡集群的所有节点都会共享工作负载，以增强性能和服务的扩展性。

3.1.3　集群分而治之

OpenStack 被设计成可实现水平扩展。它的设计采取了分而治之（Divide and conquer）的策略，其功能被广泛分布到多个服务中。这些服务本身由 API 服务、调度器（scheduler）、工作程序（worker）和代理（agent）组成。OpenStack 控制器运行 API 服务，计算、存储和网络节点运行各种代理和工作程序。

3.2 云控制器及其服务

云控制器（cloud controller）的概念旨在为您的 OpenStack 部署环境提供集中管理和控制。例如，我们可以认为云控制器管理着所有 API 调用和消息传递事务。

设想一个中型甚至大型基础架构，毫无疑问，我们需要的肯定不仅仅是单个节点。从 OpecStack 操作人员的角度来看，云控制器可以被视为服务聚合体，其中运行着 OpenStack 所需的大多数管理服务。

我们来看看云控制器主要做什么：

- 作为访问云管理和服务的网关（gateway）。
- 提供 API 服务，使得各个 OpenStack 组件能相互通信，并为最终用户提供服务接口。
- 通过各种集群和负载均衡工具实现高可用集成服务机制。
- 提供关键的基础架构服务，例如数据库和消息队列。
- 提供持久存储，其后端可能在多个分开的存储节点上。

您很可能在第 1 章中就已经注意到了云控制器中的主要服务，但那时我们并没有深入探讨为什么这些服务应该运行在控制节点中。在本章中，我们将详细研究云控制节点。控制节点聚合了 OpenStack 的多个最关键服务。下面来看看控制节点上的服务，如图 3-1 所示。

图 3-1 OpenStack 控制节点服务

3.2.1　Keystone 服务

Keystone 服务在 OpenStack 中提供身份（identity）认证和服务目录（service catalog）功能。OpenStack 中的所有其他服务必须向 Keystone 注册其 API 端点（API endpoint）。因此，Keystone 中保存着在 OpenStack 云中运行的各种服务的目录。可以使用 Keystone REST API 查询该目录。

Keystone 还维护着一个策略引擎（policy engine），该引擎提供基于规则的访问和服务授权。Keystone 服务自身由多个提供者（provider）组成，这些提供者彼此协同工作。每个提供者实现了 Keystone 架构中的一个概念（如图 3-2 所示）：

- ❑ Identity（身份）
- ❑ Resource（资源）
- ❑ Authorizaiton（认证）
- ❑ Token（令牌）
- ❑ Catalog（目录）
- ❑ Policy（策略）

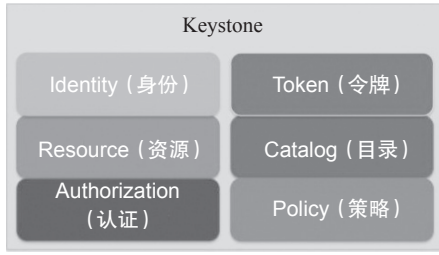

图 3-2　Keystone 功能集

1. 身份提供者

身份提供者（identity provider）负责校验用户和用户组的身份凭据（credential）。OpenStack Keystone 包含一个内置的身份提供者，它可以用于创建和管理用户和用户组的身份凭据。Keystone 还能整合外部身份提供者，比如 LDAP。OpenStack Ansible 项目提供了将 LDAP 服务作为外部身份提供者整合进 Keystone 的 playbook。

为了管理 OpenStack 服务的访问级别（access level），Keystone 支持多种用户类型。用户可以是以下几种类型之一：

- ❑ 服务用户（service user），它关联到 OpenStack 中运行的一个服务。
- ❑ 管理员用户（administrative user），它拥有资源和服务的管理权限。
- ❑ 终端用户（end user），它只是 OpenStack 资源的消费者，没有别的访问权限。

2. 资源提供者

资源提供者（resource provider）实现了 Keystone 中的项目（project）和域（domain）这两个概念。域的概念为 Keystone 中的实体（entity）提供了一个容器，包括用户、用户组和项目，可以将域看成一个企业或者一个服务提供商。

3. 认证提供者

从概念上来说，认证（authorization）是指 OpenStack 用户和用户组与他们的角色（role）之间的关系。角色被用于管理对 OpenStack 中服务的访问。Keystone 策略提供者（policy provider）根据用户所属的组和角色来强制执行规则（rule）。

4. 令牌提供者

要访问 OpenStack 中的服务，用户必须通过身份提供者（identity provider）的认证。一

旦校验完成，令牌提供者（token provider）就会产生一个令牌（token），用于该用户访问 OpenStack 服务的身份认证。令牌只在一段时间内有效。访问 OpenStack 服务需要一个有效的令牌。

5. 目录提供者

前面提到过，所有 OpenStack 服务都必须在 Keystone 中注册。目录提供者（catalog provider）维护服务和相关联端点的目录。这样，Keystone 就提供了发现 OpenStack 集群中所有服务的入口。

6. 策略提供者

策略提供者（policy provider）与允许访问 OpenStack 资源的规则（rule）相关联。策略（policy）由多条规则组成，每条规则定义了哪些用户和角色被允许访问哪些资源。比如，只允许具有管理员角色的用户向组添加用户。

前面我们学习了身份服务的主要概念。现在我们来学习 Keystone 服务的一些高级特性，这些特性包括：

- 联邦 Keystone（Federated Keystone）
- Fernet 令牌

7. 联邦 Keystone

联邦身份（federated identity）是一种使用**外部身份提供者**（external Identity Provider，IdP）提供的身份服务来访问**服务提供商**（Service Provider，SP）可用资源的机制。在 OpenStack 中，身份提供者可以是第三方，而 OpenStack 则充当服务提供商，拥有诸如虚拟存储、网络和计算实例资源。

使用联邦身份比在 OpenStack 中维护身份验证系统有一些优势：

- 一个组织可以使用现有的身份源（如 LDAP 或 Active Directory）在 OpenStack 中提供用户身份验证。它消除了为 OpenStack 用户管理单独身份源的必要性。
- 使用身份服务有助于与不同的云服务集成。
- 它提供了单一的事实和用户管理源。因此，如果用户是身份服务的一部分，那就会非常容易地给予他访问 OpenStack 云基础架构的权限。
- 因为不需要为 OpenStack 用户运行另一个身份系统，从而降低了安全风险。
- 它有助于为 OpenStack 用户提供更好的安全性和单点登录。

因此，一个显而易见的问题是：Federated 身份是如何运作的呢？为了理解这一点，让我们看一下用户如何与 SP 和 IdP 服务进行交互。简而言之，使用 Federated 身份验证用户将包含以下步骤：

- 用户尝试去访问 SP 的某个可用资源。
- SP 检查用户是否已有经过认证了的会话（session）。如果没有，他将被重定向至身份提供者的身份认证 URL。
- 身份提供者提示用户输入身份凭证，例如用户名和密码，然后验证其身份，并发放

无作用域令牌（unscoped token）。无作用域令牌包含通过了身份验证的用户所属的组的列表。

- 然后，用户使用无作用域令牌来确定服务提供商的云中的可访问的域和项目列表。
- 然后，用户使用无作用域令牌为他感兴趣的项目和域获取有作用域令牌（scoped token），并可以开始使用云上的资源。

联邦身份的一个有趣用例是使用来自两个不同云基础架构的 Keystone 服务，这两个云基础架构分别充当服务提供者和身份提供者（也称为 K2K 或 Keystone 到 Keystone 用例），从而允许来自作为身份提供者的第一个 Keystone 的用户访问作为服务提供者的另一个云基础架构上的资源。

我们在第 2 章中讨论过的 OpenStack Ansible 项目为 Keystone Federated 服务提供了丰富的配置选项。例如，您可在以下位置获得 Mitaka 版本中发布的配置选项的详细信息：`https://docs.openstack.org/developer/openstack-ansible/mitaka/install-guide/configure-federation.html`。

8. Fernet 令牌

过去，Keystone 向通过了身份验证的用户发放基于公钥基础结构（Public Key Infrastructure，PKI）的令牌。用户使用这些令牌访问各种 OpenStack 服务。PKI 令牌以 JSON 格式封装了用户身份和授权上下文（authorization context）。这使得 PKI 令牌很大。此外，这些令牌需要被存储在数据库中以允许令牌撤销。PKIZ 令牌是对 PKI 令牌的增强，因为它们通过压缩来减小令牌的大小，但因为要保存所有已发放的令牌，仍然在数据库上产生了相当大的压力。

为了克服 KPI 令牌太大以及数据库高负载这些缺点，Keystone 中引入了 Fernet 令牌，现在它已成为默认令牌格式。Fernet 令牌相对较小。使用 Fernet 令牌的主要优点是它们不需要存储在数据库中。由于不涉及数据库交互，因此创建这种令牌的速度会快很多。Fernet 令牌包含用户身份、项目授权范围和过期时间。使用密钥加密身份和授权会生成 Fernet 令牌。它们比 PKI 令牌小得多，速度也快很多。

要在 Keystone 中启用 Fernet 令牌，需要对令牌提供者做如下修改：

```
keystone.token.providers.fernet.Provider
```

OpenStack Ansible 项目默认使用 Fernet 令牌，但您可以使用用户变量 keystone_token_provider 来修改配置。由于 Fernet 使用加密密钥生成新令牌，为了维护系统安全性，这些密钥需要时不时地进行轮换（rotate）。使用 Fernet 提供者的 Keystone 服务将同时激活多个 Fernet 密钥。密钥轮换机制是可配置的，它确定了活动密钥的数量和轮换频率。需要注意那些用于配置 Fernet 密钥轮换的变量，它们是密钥存储库位置、活动密钥数目和轮换机制。

以下是 OpenStack Ansible 工具提供的默认值：

```
keystone_fernet_tokens_key_repository: "/etc/keystone/fernet-keys"
keystone_fernet_tokens_max_active_keys: 7
keystone_fernet_rotation: daily
keystone_fernet_auto_rotation_script: /opt/keystone-fernet-
rotate.sh
```

您可以在这个链接中找到 Ansible Keystone 剧本（playbook）所使用的默认配置设置：https://github.com/openstack/openstack-ansible-os_keystone/blob/master/defaults/main.yml。

3.2.2 nova-conductor 服务

如果您曾安装过 OpenStack Grizzly 版本，现在当您查看 OpenStack 节点中运行的 Nova 服务时，您可能会注意到一个名为 nova-conductor 的新服务。不用诧异！这项令人惊叹的新服务改变了 nova-compute 服务访问数据库的方式。增加该服务是为了解绑从计算节点上直接访问数据库而增强安全性。运行 nova-compute 服务的易受攻击的计算节点有可能受到各种攻击。您可以想象通过攻击虚拟机来将计算节点置于攻击者的控制之下。更糟糕的是，这可能危及数据库。然后，您会猜到结果：整个 OpenStack 集群都受到了攻击！ nova-conductor 的工作是代表计算节点执行数据库操作，并提供一层数据库访问隔离。

所以，可以认为 nova-conductor 是在 nova-compute 之上增加了一个新层。此外，不同于其他解决数据库访问瓶颈的方法，nova-conductor 将来自计算节点的请求并行化。

如果您在 OpenStack 环境中使用 nova-network 和多主机（multihost）网络，nova-compute 仍然需要直接访问数据库。但是对于每个 OpenStack 新版本，运营商可以迁移到 Neutron 以满足其网络需求。

3.2.3 nova-scheduler 服务

近些年来，在云计算领域中已经有了很多工作流调度方面的研究和实现，其目的都是为了确定资源创建的最佳放置位置。在 OpenStack 中，我们需要决定在哪个计算节点上创建虚拟机。值得注意的是，OpenStack 中已经有了一系列调度算法。

Nova-scheduler 还可能影响虚拟机的性能。因此，OpenStack 支持一组过滤器（filter），它们会检查计算节点上资源的可用性，然后通过加权机制（weighting mechanism）来过滤出计算节点列表，然后再确定启动虚拟机的最佳节点。调度程序允许您可以根据一定数量的度量标准和策略考量来配置其选项。此外，nova-scheduler 可以被视为云控制节点中的决策者（decision-maker box），它使用复杂的算法来有效地利用和放置虚拟机。

最后，正如前面所提到的，OpenStack 中的调度程序将在云控制节点中运行。试问一下：

高可用性环境中的调度程序有何不同？此时，我们可利用 OpenStack 体系结构的开放性，每个调度程序都运行多个实例，它们一同监视同一个调度请求队列。

顺便说一下，其他 OpenStack 服务也实现了调度程序。例如，cinder-scheduler 是 OpenStack 中块存储卷的调度服务。类似地，Neutron 实现了调度程序以在网络节点之间分配诸如虚拟机路由器（vRouter）和 DHCP 服务器之类的网络组件。

调度程序可以通过各种选项来进行配置，这些配置选项位于 /etc/nova/nova.conf 中。要阅读 OpenStack 调度的更多细节，请参阅以下链接：https://docs.openstack.org/mitaka/config-reference/compute/sched uler.html。

3.2.4　API 服务

我们已经在第 1 章中简要介绍了 nova-api 服务。现在，我们向前进一步，来学习作为云控制器中的编排引擎（orchestrator engine）的 nova-api 服务。nova-api 服务运行在控制节点中。

Nova-api 服务能将消息写入数据库和消息队列来向其他守护进程传递消息，这使得它能够处理复杂请求。该服务也是基于端点（endpoint）概念，端点是所有 API 查询的发起点。它提供两种不同的 API，OpenStack API 和 EC2 API。因此，在部署云控制节点之前，您需要决定使用哪种 API。如果两种都使用的话，可能会有一些问题。这是因为每种 API 信息呈现的异质性（heterogeneity）。例如，OpenStack API 使用名称和数字来引用实例，而 EC2 API 使用基于十六进制值的标识符。

此外，我们还将计算、身份、镜像、网络和存储 API 服务放在控制节点中，还可以运行其他 API 服务。例如，我们往往将大部分 API 服务运行在云控制节点上来满足部署要求。

应用程序编程接口（API）允许公共访问 OpenStack 服务，并提供与它们交互的方式。API 访问可以通过命令行或 Web 执行。要阅读有关 OpenStack 中 API 的更多信息，请参阅以下链接：http://developer.openstack.org/#api。

3.2.5　镜像管理

云控制器还将托管用于镜像管理的 Glance 服务，该服务使用 glance-api 和 glance-registry 来存储、列表和获取镜像。Glance API 提供外部 REST 接口，用于查询虚拟机镜像及相关元数据。Glance registry 将镜像元数据存储在数据库中，并利用存储后端来存储实际镜像。因此，在设计镜像服务时，需要决定采用哪种存储后端。

glance-api 支持几种后端存储来存储镜像。Swift 是一个很好的选择，它将镜像存储为对象，并且具有良好的扩展性。还有其他替代方案，例如文件系统后端、Amazon S3 和 HTTP。接下来的第 5 章会详细介绍 OpenStack 中的不同存储模型。

3.2.6 网络服务

就像 OpenStack 中的 Nova 服务向计算资源的动态请求提供 API 一样，网络服务也采用类似概念，通过驻留在云控制器中的 API 来提供服务。它还支持通过扩展（extension）来提供高级网络功能，例如防火墙、路由、负载均衡等。正如前面所讨论的，我们强烈建议将大多数网络工作进程（worker）分开。

另一方面，您还需考虑到云控制器运行多个服务时会遇到的大流量；因此，您应该牢记可能面临的性能挑战。在这种情况下，集群最佳实践可帮助您的部署更具可伸缩性并可提高其性能。前面提到的那些技术是必不可少的，但还不够。例如，还需要带有至少 10GB 的网卡绑定的服务器。

网卡绑定（NIC bonding）技术用于增加可用带宽。两块或者多块绑定在一起的网卡就像一块物理网卡一样工作。

您可以随时参考第 1 章中的内容，通过周密计算，以使您的云控制器能够顺利响应所有请求而不会出现瓶颈。在早期阶段过多考虑性能对于获得良好的拓扑弹性可能没有什么帮助。要实现弹性，您可以随时利用扩展性功能来改进部署环境。我们建议在需要的时候采用水平扩展方式。

3.2.7 Horizon 仪表板服务

OpenStack 仪表板（dashboard）运行在 Apache web 服务器后端，基于 Python Django web 应用框架。因此您可考虑将 Horizon 仪表板运行在一个可以访问 API 端点的单独节点上，以减轻云控制节点的负载。像大多数部署环境一样，您可以将 Horzion 运行在控制节点上，但建议对它密切监控，以便在需要时把它分离到一个单独节点上。

3.2.8 计量服务

在 Liberty 之前的 OpenStack 版本中，计量服务（telemetry service）最初只由 Ceilometer 这一个组件组成，来提供 OpenStack 中资源利用率的计量。在多用户共享的基础架构中，跟踪资源使用情况至关重要。资源利用率数据可用于多种目的，例如计费、容量规划、按需求和吞吐量的虚拟基础架构自动扩展等。

Ceilometer 最初被设计为 OpenStack 计费解决方案的一部分，但后来却发现了把它作为一个独立项目的许多用例。自 Liberty 版本以来，又有两个新服务从计量服务中分离出来成为独立服务。

- Aodh：一个子项目，负责在资源利用率超过预定义阈值时生成告警。
- Gnocchi：一个子项目，负责大规模存储度量（metric）和事件（event）数据。

Ceilometer 采用基于代理（agent）的架构。服务本身由 API 服务、多个数据收集代理和

数据存储组成。Ceilometer 通过 REST API 收集数据，监听通知（notification）并直接轮询资源。以下是数据收集代理及其角色。

- Ceilometer 轮询代理（polling agent）：其提供了一个插件框架来收集各种资源的利用率数据。它在控制节点上运行，并以插件方式实现了一个灵活的框架来方便添加各种数据收集代理。
- Ceilometer 中央代理（central agent）：其主要在控制节点上运行，并使用其他 OpenStack 服务公开的 REST API 来轮询租户创建的虚拟资源。中央代理可以轮询对象和块存储、网络以及使用 SNMP 轮询物理硬件。
- Ceilometer 计算代理（compute agent）：在计算节点上运行，其主要目的是收集虚拟机管理程序（hypervisor）的统计信息。
- Ceilometer IPMI 代理（IPMI agent）：在被监控的计算节点上运行。它通过服务器上的智能平台管理接口（IPMI）来收集物理资源利用率数据。被监视的服务器必须配备 IPMI 传感器并安装 ipmitool 程序。它使用消息总线发送收集到的数据。
- Ceilometer 通知代理（notification agent）：从 OpenStack 消息总线上获取各个组件的通知数据。它也在控制节点上运行。

1. 告警

除了各种数据收集代理外，Ceilometer 还包含告警（alarm）通知代理（notification agent）和评估代理（evaluation agent）。告警评估代理判断资源使用数值是否超过阈值，并使用通知代理发出相应的通知。告警系统可用于构建自动缩放的基础架构，或采取其他纠正措施。Aodh 子项目旨在将告警与 Ceilometer 分离。您可以在部署全新 OpenStack 环境时启用这种分离配置。

2. 事件

事件（Event）描述了 OpenStack 中资源的状态变化；例如，卷创建、虚拟机实例启动等。事件可用于向计费系统提供数据。Ceilometer 监听从各种 OpenStack 服务发出的通知，并将它们转换为事件。有关 OpenStack 中计量服务和监控的更多详细信息和更新，将在第 10 章中介绍。

3.2.9　基础架构服务

基础架构服务（Infrastructure service）不是 OpenStack 公共服务，但被多个 OpenStack 组件使用，例如数据库、消息队列和缓存等。我们来看看这些服务的性能和可靠性要求。

1. 规划消息队列

消息队列系统（message queue system）是另一个关键子系统，它必须是集群式的。如果消息队列出现故障，那么整个 OpenStack 集群将停止运行。OpenStack 支持多种消息队列方案，包括以下几种：

- RabbitMQ

- ZeroMQ
- Qpid

我们选择 RabbitMQ 作为消息队列系统，因为它原生支持集群模式。要在 OpenStack 环境中提供高可用消息队列，RabbitMQ 消息服务器必须以集群方式运行。请记住，默认情况下，消息队列服务器集群只会复制消息服务器运行在高可用模式下所需的状态数据，但不会复制队列。队列驻留在最初被创建时所在的单个节点上。为了提供可靠消息传递服务，我们还应该启用镜像队列（mirrored queue）。

应该注意的是，即使在采用镜像队列的集群中，客户端也会始终连接到主节点（master）以发送和使用消息，从节点（slave）只会复制消息。从节点会一直保留消息，直到主节点确认消息被删除了。要提供完全集群式的活动 – 活动（active-active）消息服务，队列还需要与 Pacemaker 和 DRBD 等集群方案集成。

另一方面还要考虑消息队列中通信的安全性。RabbitMQ 使用 TLS 提供传输安全性。使用 TLS，客户端可以获得受保护不会被篡改的消息。TLS 使用 SSL 证书加密和保护通信。SSL 证书可以是自签名的，也可以由 CA 提供。此外，RabbitMQ 还基于用户名和密码提供身份验证和授权。

一个好的做法是，当我们启动一个运行有 RabbitMQ 服务的简单云控制器时，也要考虑到这些复杂的挑战。可喜的是，我们的设计是富有弹性的，可以通过增加控制节点来扩展集群。我们还可以将 RabbitMQ 服务从控制节点分离出去，以简化控制节点。

参考 OpenStack Ansible 文档中配置 Mitaka 版本中的 RabbitMQ 部分，比如：`https://docs.openstack.org/develope r/openstack-ansible/mitaka/install-guide/configure-rabbitmq.html`。我们将在第 9 章中深入介绍 RabbitMQ 的高可用性。

2. 整合数据库

IT 基础架构中发生的大多数灾难都可能导致数据丢失，不仅是生产数据，还包括历史数据。数据丢失可能导致 OpenStack 环境不能运行甚至无法恢复。因此，我们需要在早期阶段就采用 MySQL 集群和高可用性方案。我们可在云控制器中运行 MySQL Galera。有关配置 MySQL 集群的更多详细信息将在第 9 章中介绍。

在 OpenStack 集群中启用计量服务后，我们还需要安装 No-SQL 数据库来存储计量信息。我们将使用 MongoDB 作为 No-SQL 数据库。

3.3 云控制器集群部署准备

作为物理云控制器的支持者，为实现机器集群所做的努力被认为是向实现高可用性在

正确方向上迈出的第一步。第 9 章中将讨论几种高可用（HA）拓扑。

正如我们已经看到几个服务用例，包括分离式和集群式，我们将扩展第 1 章中描述的云控制器的逻辑设计。请记住，OpenStack 是一个具有高度可配置性的平台，它能满足特定需求和特别的条件。下一步是确认第一个逻辑设计。可能会出现以下问题：它是否满足某些需求？所有服务都在安全的高可用区吗？

好吧，请注意我们采用了 MySQL Galera 集群来确保数据库高可用。这意味着云控制器集群中需要至少引入第三个控制器，以支撑 Galera 基于法定人数的一致系统（quorum-based consensus system）。

由此产生的问题是：我是否应该添加额外的云控制器来实现复制和数据库高可用？还需要第四个或第五个控制器吗？好！请一直保持这种心态吧。这个时候，您的设计方向已经是对的了，并且您已经知道必须做一些更改以应对某些物理约束。在早期阶段，我们需要确保我们的设计都是高可用的。请记住，设计的每一层都不能有故障单点！

冗余（redundeny）是通过虚拟 IP 和 Pacemaker 实现的。然后，利用 HAProxy 实现负载均衡。MySQL 使用 Galera 进行复制，RabbitMQ 被构建为集群模式，数据库和消息队列服务器被实现为 active-active 高可用模式。通过与 Corosync、Heartbeat 或 Keepalived 集成，我们的设计还有其他选择。第 9 章中将详细介绍负载均衡、高可用性和故障切换的相关解决方案。

预先准备好如何实现云控制器集群非常重要。您可以参考第 9 章的内容以查看更多详细信息和实际例子。例如，整个 OpenStack 云应能通过添加运行各种服务的节点轻松得以扩展。我们稍后会通过一种自动化方法来实现云的横向扩展。

3.3.1　OpenStack Ansible 安装部署

上面我们已经讨论了运行在控制节点上的各种服务。现在，我们来使用 OpenStack Ansible（OSA）开始部署控制节点。我们建议生产环境控制器的配置包括至少三个节点，这些节点运行 OpenStack API 服务并承载基础架构服务，例如 MySQL 服务、memcached 和 RabbitMQ 服务等。这符合我们的集群需求。

此外，除 OpenStack 控制节点之外，我们还需要一个节点来运行 OpenStack Ansible 工具，它被称为部署节点，如图 3-3 所示。

部署节点

部署节点（deployment node）上运行着 Ansible OpenStack 工具，它将在目标节点（target node）上编排 OpenStack 云节点。部署节点安装 Ubuntu 14.04 LTS 64 位操作系统。要在部署节点上运行 OpenStack Ansible，需执行以下步骤。

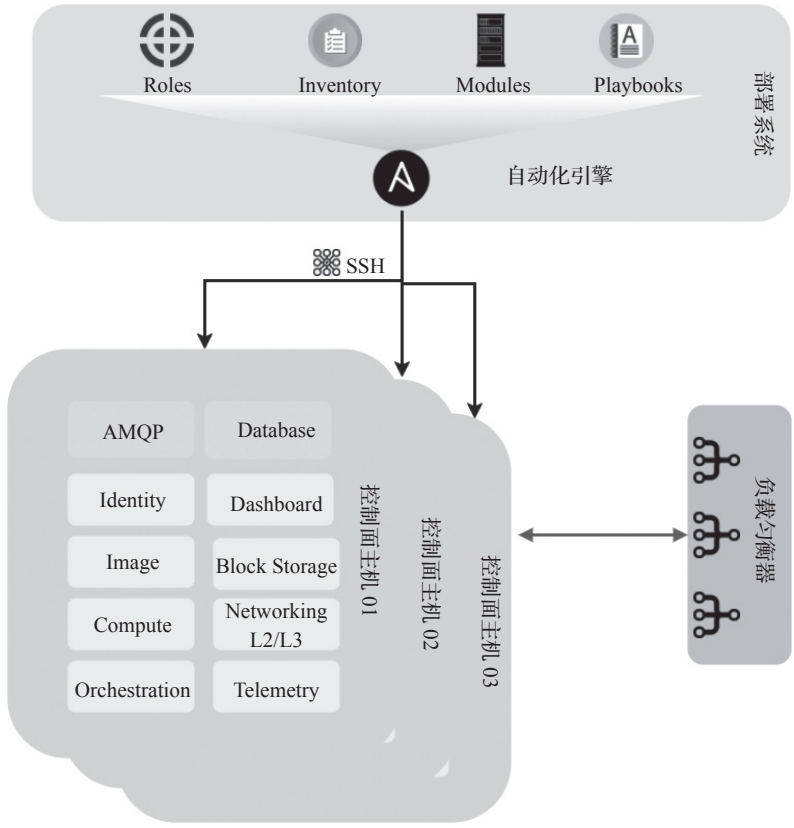

图 3-3　OpenStack 集群部署架构

（1）安装必要的软件包，包括 Git、ntp、sudo 和 openssh-server：

```
# apt-get install aptitude build-essential git ntp ntpdate
openssh-server python-dev sudo
```

（2）配置 NTP 来从网络同步时间。

（3）配置网络，使得目标主机可以从部署主机上访问到。

（4）克隆 OpenStack Ansible 代码仓库：

```
# git clone -b TAG https://github.com/openstack/openstack-
ansible.git /opt/openstack-ansible
```

（5）启动 OpenStack Ansible：

```
# cd /opt/openstack-ansible
# scripts/bootstrap-ansible.sh
```

（6）在部署节点上生成 SSH 密钥。

3.3.2　控制节点环境准备

现在我们来看一下控制节点。我们需要安装所有 API 服务，以及公共的和基础架构服务。本节中，我们会介绍用于部署控制节点的所有 OpenStack Ansible 剧本（playbook）。

1. 目标主机

控制节点是 OpenStack Ansible 的目标主机（target host）中的一部分。所有目标节点都安装 Ubuntu 14.04 LTS 64 位操作系统。

从 OpenStack Ocata 版本开始，OSA 已支持 Centos 7 了。

节点必须安装最新的系统软件包更新：

```
# apt-get dist-upgrade
```

目标节点上必须安装网卡绑定（NIC bonding）、VLAN、网桥（bridging）等软件包：

```
# apt-get install bridge-utils debootstrap ifenslave
ifenslave-2.6 lsof lvm2 ntp ntpdate openssh-server sudo
tcpdump vlan
```

配置目标节点与 NTP 服务器同步。在撰写本书时，OpenStack Ansible 最低需要内核版本 3.13.0-34-generic。

接下来，从 OpenStack Ansible 部署主机上向目标节点部署 SSH 密钥。这可以通过将部署主机的公钥添加到每个目标主机上的 `authorized_keys` 文件来实现。目标主机上能可选地配置名为 `lxc` 的 LVM 卷组（LVM volume group）。如果目标主机上存在 LVM 卷组 `lxc`，则 `lxc` 文件系统会被创建在卷组上；否则，它会被创建在 `/var/lib/lxc` 内。

2. 配置网络

目标节点的网络设置必须为以下服务提供通信路径：

- ❑ 管理网络
- ❑ 存储管理
- ❑ 租户网络

OpenStack Ansible 使用网桥（bridge）在目标节点上提供上述通信路径。租户网络既可以实现为基于 VLAN 的网段，也可以使用隧道网络（如 VXLAN）。可通过从单个网卡或多个绑定的网卡上创建子接口来提供到网桥的物理链接。管理网络由 br-mgmt 网桥提供，存储网络由 br-storage 网桥提供，租户网络由 br-vlan 和 br-vxlan 网桥提供。图 3-4 显示了采用网卡绑定（bonding）和多网卡的详细网桥配置：

尽管多网卡和绑定接口不是必需的，但我们建议在生产环境中采用这种配置。

要了解参考网络配置的详情，请访问：`https://docs.openstack.org/developer/openstack-ansible/`。

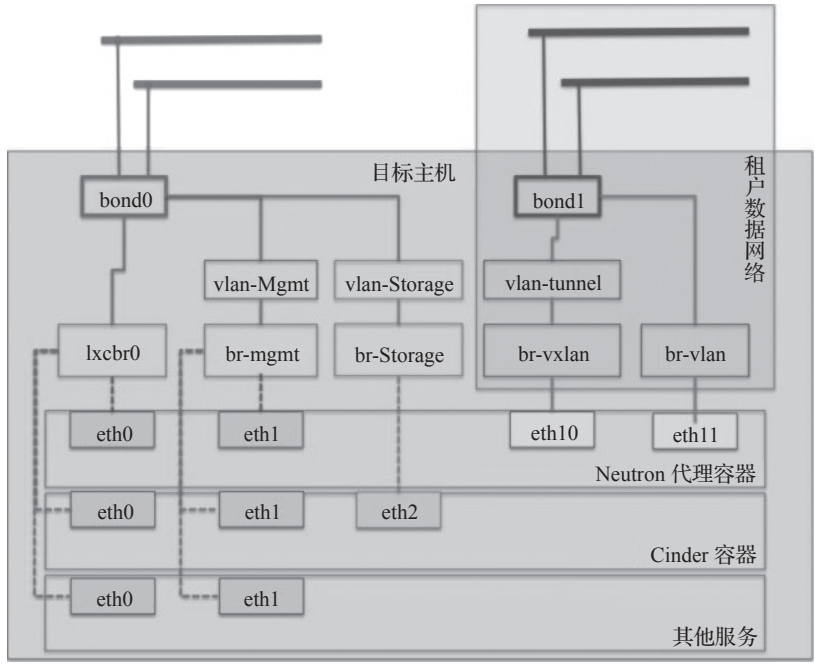

图 3-4　网卡绑定与多网桥

尽管多网卡和绑定接口不是必需的，但我们建议在生产环境中采用这种配置。要了解参考网络配置的详情，请访问：https://docs.openstack.org/developer/openstack-ansible/。

3.4　使用 OpenStack playbook 部署集群

OSA 提供 playbook（剧本）来配置各 OpenStack 服务。这些 playbook 会运行任务来在目标节点上配置 OpenStack 服务。

3.4.1　配置 OpenStack Ansible

使用 OSA 部署 OpenStack 是通过一系列配置文件来控制的。配置文件必须放在 `/etc/openstack_deploy` 文件夹中。AIO 存储库中有示例目录结构。我们可以从存储库中复制示例配置文件开始，然后再根据需求进行调整：

```
#cp /opt/openstack-ansible/etc/openstack_deploy /etc/openstack_deploy
#cd /etc/openstack_deploy
#cp openstack_user_config.yml.example openstack_user_config.yml
```

现在开始为我们的部署定制用户配置文件。openstack_user_config.yml 使用主机组（host group）来定义运行特定服务的目标主机。

3.4.2　网络配置

下一步是为我们在前面讨论中提到的不同网络路径配置网络地址。这包括用于访问服务容器（service container）的管理网络、存储和隧道网络的 CIDR。以下是 AIO 存储库中 `openstack_user_config.yaml` 文件的示例：

```
cidr_networks:
  container: 172.47.36.0/22
  tunnel: 172.47.40.0/22
  storage: 172.47.44.0/22
```

用于容器的 CIDR 应与实验室管理网络相同。这些 CIDR 地址映射到管理、存储和隧道网桥。使用 `used_ips` 组将 CIDR 范围内的已在基础架构中被使用了的所有 IP 地址都排除掉。使用提供者网络级别（provider network level）来定义各种网络路径，例如 `br-mgmt`、`br-storage`、`br-vxlan` 和 `br-vlan`。OSA 存储库中的示例配置文件 `openstack_user_config.yml.example` 中有示例配置和详细注释。

3.4.3　配置主机组

表 3-1 列出了 OpenStack Ansible 使用的主机组及其用途。一个主机可以在多个主机组中。主机使用其管理 IP 地址进行标识。在 `openstack_user_config.yml` 文件中，可以添加新的分段（section）来描述我们的 OpenStack 基础架构，包括控制节点。在多节点安装环境中，每个分段描述一组按角色组织的目标主机，汇总如表 3-1 所示。

表 3-1　OpenStack Ansible 主机角色定义

OpenStack Ansible 主机组	角色
repo-infra_hosts	用于运行软件包仓库。shared-infra_hosts 可以被分配 repo-infra_hosts 角色
shared-infra_hosts	运行共享基础架构服务，比如 MySQL、RabbitMQ 和 memcached
os-infra_hosts	运行 OpenStack API 服务，比如 Glance API、Nova API、Ceilometer API 等
identity_hosts	运行身份服务
network_hosts	运行网络服务
compute_hosts	列出运行 `nova-compute` 服务的主机
storage-infra_hosts	运行 Cinder API 服务
storage_hosts	运行 Cinder volume 服务
log_hosts	列出运行日志服务的主机
haproxy_hosts	列出运行 HAProxy 服务的主机。在生产环境中，目标主机可以指向硬件负载均衡器

如第 1 章中所述，我们将使用三个控制节点来运行公共 OpenStack 服务，包括 API 服务。通过指定将哪个主机组分配给控制节点，从一个文件就可以轻松地把控制节点部署出来。在 `openstack_user_config.yml` 文件中，首先调整共享服务的 `shared-infra_hosts` 定义段，例如数据库和消息队列：

```
shared-infra_hosts:
cc-01:
  ip: 172.47.0.10
cc-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

然后 os-infra_hosts 定义段将告知 Ansible 在 OpenStack 控制节点上安装 OpenStack 的 API 服务：

```
os-infra_hosts:
cc-01:
  ip: 172.47.0.10
cc-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

我们还可以通过将控制节点列表加入 storage-infra_hosts 定义段，来将块存储 API 服务部署到这些节点上：

```
storage-infra_hosts:
cc-01:
  ip: 172.47.0.10
cc-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

调整 identity_hosts 定义段，使 Keystone 部署在控制节点上：

```
identity_hosts:
cc-01:
  ip: 172.47.0.10
ccr-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

可选地，通过修改 repo-infra_hosts 定义段，控制节点还可用于运行安装包仓库：

```
repo-infra_hosts:
cc-01:
  ip: 172.47.0.10
ccr-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

要对 API 做负载均衡，在 haproxy_hosts 分段，可配置 Ansible 在每个控制节点上安装 HAProxy 作为虚拟负载均衡器：

```
haproxy_hosts:
cc-01:
  ip: 172.47.0.10
cc-02:
  ip: 172.47.0.11
cc-03:
  ip: 172.47.0.12
```

根据控制节点的 HA 设置，建议在生产环境中使用硬件负载均衡器。在 Ansible 中配置这两种负载均衡器的方法有所区别。

要在每个控制节点上安装 HAProxy，需要在 Ansible 配置文件 /etc/openstack_deploy/user_variables.yml 中做额外的设置来配置 Keepalived：

- 配置所有 HAProxy 实例共享的外部和内部 CDIR 虚拟 IP：

```
haproxy_keepalived_external_vip_cidr
haproxy_keepalived_internal_vip_cidr
```

- 配置所有 HAProxy 实例上 keepalived 绑定内部和外部 VIP 的虚拟网络设备：

```
haproxy_keepalived_external_interface
haproxy_keepalived_internal_interface
```

控制节点和 API 服务的高可用设置将在第 9 章中更详细地介绍。

Ansible OpenStack 采用不同的方式来处理服务配置的秘密（secret）和密钥（key），例如数据库根密码和计算服务密码短语（passphrase）。秘密保存在另一个文件 /etc/openstack_deploy/user_secrets.yml 中。

以纯文本形式在存储库中保存密码和密码短语存在高安全性问题，必须使用加密机制来解决。Ansible 提供了更多高级功能，可使用 Vaults 功能将密码数据和密钥保存在加密文件中。要阅读 Ansible 中有关 Vaults 的更多信息，请参阅官方 Ansible 网页：http://docs.ansible.com/ansible/playbooks_vault.html。

在重新审视 Ansible 控制平面的网络设置和主机服务设置之后，我们继续简要介绍哪些 playbook 将用于部署控制节点和公共 OpenStack 服务。

3.4.4　用于集群部署的 playbook

OpenStack Ansible playbook（剧本）是描述如何在目标系统上部署服务的蓝图（blueprint）。playbook 由任务组成，Ansible 必须执行这些任务才能启动 OpenStack 服务。另一方面，这些任务指向封装了服务部署所有细节的特定角色。

表 3-2 列出了部署各种服务的 playbook。使用 openstack-ansible 命令来运行 playbook。这是一个封装脚本，它为 ansible-playbook 命令提供了 OpenStack 配置：

```
# openstack-ansible playbook-name.yaml
```

可将 playbook-name 替换为您想要运行的 playbook 的名称。

表 3-2 OpenStack Ansible playbook

playbook 名称	OpenStack 服务
os-keystone-install.yml	身份（Identity）服务
os-glance-install.yml	镜像（Image）服务
os-cinder-install.yml	块存储（Block Storage）服务
os-nova-install.yml	计算（Compute）服务
os-neutron-install.yml	网络（Network）服务
os-heat-install.yml	编排（Orchestration）服务
os-horizon-install.yml	仪表板（Horizon）服务
os-ceilometer-install.yml os-aodh-install.yml os-gnocchi-install.yml	计量（Telemetry）服务

每个 playbook 都使用 Ansible 变量（variable）提供配置选项。playbook 将角色应用于目标主机。OpenStack Ansible 将 OpenStack 服务维护为外部角色存储库。例如，Keystone 的部署使用 os_keystone 角色，这可以在 playbook 源代码中找到：

```
...
roles:
- role: "os_keystone"
...
```

该角色的 Git 仓库地址是 https://github.com/openstack/openstack-ansible-os_keystone。

要部署控制节点，只需运行 playbook，然后 Ansible 将负责根据用户和变量配置文件编排每个服务的安装。这首先需要运行在 /etc/openstack_deploy/ 下的 setup-hosts.yml playbook，以在目标控制器主机上安装和配置容器（container）：

```
# openstack-ansible setup-hosts.yml
```

对于控制平面，我们需要安装基础架构服务，包括 Galera、MariaDB、memcached 和 RabbitMQ。这可以通过运行在 /etc/openstack_deploy/ 下的 setup-infrastructure.yml playbook 来完成：

```
# openstack-ansible setup-infrastructure.yml
```

包括镜像、网络、身份、仪表板、计算 API、计量和编排服务的 OpenStack API 服务，可以通过运行在 /etc/openstack_deploy/ 下的 setup-openstack.yml playbook 来安装：

```
# openstack-ansible setup-openstack.yml
```

由于尚未定义计算和网络目标主机，OpenStack 环境的 Ansible 配置仍然是不完整的。在接下来的章节中更详细地介绍它们之前，了解 OpenStack-Ansible playbook 如何按角色组

织是至关重要的。

要通过在每个控制节点中部署 HAProxy 来安装虚拟负载均衡器，请使用 /etc/openstack_deploy/ 下的 haproxy-install.yml playbook，如下所示：

```
# openstack-ansible haproxy-install.yml
```

使用 HAProxy 来设置 HA 的更详细细节，会在第 9 章中进行介绍。

此外，在运行 Ansible 封装的命令行接口之前，运维人员可以通过为每个 playbook 配置文件设置多个选项来获得更大的灵活性。可以通过 https://github.com/openstack/openstack-ansible-os_Service/blob/master/defaults/main.yml 上 的 每个 OpenStack playbook 角色来自定义 OpenStack 配置控制平面。在使用 Ansible 进行部署之前，您需要对每个 OpenStack 服务进行配置。位于 defaults 目录下的每个 main.yml 文件都暴露了一系列选项和指令，它们会在应用角色时被应用于目标节点。例如，通过设置 /openstack-ansible-os-horizon/defaults/main.yml 文件中的 horizon_enable_neutron_lbaas 值为 true，可自定义 Horizon playbook 以默认启用 Neutron Load Balancer as a Service（LBaaS）。

应用 Ansible OpenStack playbook 时，user_variables.yml 会被更高优先级地使用。在 user_variables 文件中被调整了的附加指令都将默认被加载并应用于目标主机。

3.5　总结

在本章中，我们研究了 OpenStack 控制节点。我们简要讨论了高可用性的需求和集群的重要性。我们将在第 9 章中详细讨论高可用和故障切换。我们讨论了在控制节点上运行的各个服务，以及运行 OpenStack 集群所需的公共服务。我们分析了 Keystone 服务及其对各种后端的支持，这些后端可用于提供身份和认证。我们还讨论了 Keystone 支持 Federated 身份验证这一新趋势。

之后，我们介绍了 OpenStack Ansible 工具，以及启动控制节点所涉及的各个 playbook。我们还研究了将用于启动 OpenStack 服务的目标服务器的基本配置。我们讨论了网络配置，并查看了 OpenStack Ansible 自动化工具提供的一些自定义选项。下一章将讨论计算节点安装和各种配置选项。