

## 10. 第 10 章 部署微服务

### 本章内容

- 了解为什么 DevOps 对微服务至关重要
- 配置 EagleEye 服务使用的核心 Amazon 基础设施
- 手动将 EagleEye 服务部署到 Amazon
- 为你的服务设计构建和部署管道
- 从持续集成转向持续部署
- 将你的基础设施视为代码
- 构建不可变的服务器
- 在部署中测试
- 将你的应用程序部署到云

我们到了本书的结尾，但这不是微服务之旅的结束。虽然本书的大部分内容都集中在使用 Spring Cloud 技术设计、构建和操作基于 Spring 的微服务，但我们还没有涉及如何构建和部署微服务。创建构建和部署管道似乎是一件平凡的事情，但实际上它是微服务架构中最重要的部分之一。

为什么？请记住，微服务架构的关键优势之一是微服务是可以快速构建、修改和部署到生产且彼此独立的小代码单元。服务的小尺寸意味着新功能（和关键错误修复）可以快速交付。速度是关键词，因为速度意味着在创建新功能或修复错误和部署服务之间几乎没有冲突。交付时间应该是几分钟，而不是几天。

要做到这一点，你用来构建和部署你的代码的机制需要：

- **自动化**：在构建代码时，在构建和部署过程中不应该有人为干预，特别是在较低的

环境中。构建软件，配置机器镜像，然后部署服务的过程应该是自动的，并且应该由向源存储库提交代码的行为发起。

- **可重复**：用于构建和部署软件的过程应该是可重复的，以便每次构建和部署启动时都会发生同样的情况。过程中的变化往往是细微的错误的来源，难以追查和解决。
- **完整**：部署构件的输出物应该是一个完整的虚拟机或容器镜像（Docker），其中包含该服务的“完整”运行时环境。这是你考虑基础设施方式的重要转变。你的机器镜像的设置需要通过脚本完全自动化，并使服务源代码保持在源代码控制之下。

在微服务环境中，这个职责通常从运营团队转移到拥有该服务的开发团队。请记住，微服务开发的核心原则之一是将服务的完整运营责任推给开发人员。

- **不可变**：一旦构建了包含你的服务的机器镜像，镜像的运行时配置不应该在部署镜像之后被触及或更改。如果需要进行更改，则需要在源代码管理下的脚本中进行配置，服务和基础设施需要再次执行构建过程。

应该将运行时配置更改（垃圾回收设置，正在使用的 Spring 概要文件）作为环境变量传递给镜像，而应用程序配置应与容器保持分离（Spring Cloud Config）。

构建一个强大而通用的构建部署管道是一项重要的工作，通常专门针对你的服务将要运行的运行时环境而设计。它经常涉及到一个专门的 DevOps（开发人员操作）工程师团队，他们的唯一工作是推广构建过程，以便每个团队都可以构建自己的微服务，而无需为自己重新构建整个构建流程。不幸的是，Spring 是一个开发框架，并没有提供实现构建和部署管道的大量功能。

在本章中，我们将看到如何使用一些非 Spring 工具来实现构建和部署管道。你将要为本书编写一套微服务，并执行以下操作：

- 将你使用的 Maven 构建脚本集成到称为 Travis CI 的持续集成/部署云工具中

- 为每个服务构建不可变的 Docker 镜像，并将这些镜像推送到一个集中的存储库
- 使用 Amazon 的 EC2 容器服务（ECS）将整套微服务部署到 Amazon 的云端
- 运行将测试服务正常运行的平台测试

我想带着最终的目标开始讨论：向 AWS 弹性容器服务（ECS）部署一套服务。在我们详细介绍如何实现构建/部署流程之前，我们先来看看 EagleEye 服务将如何在 Amazon 云中运行。然后我们将讨论如何将 EagleEye 服务手动部署到 AWS 云。一旦完成，我们将自动化整个过程。

### 10.1. EagleEye：在云环境中设置核心基础设施

在本书中的所有代码示例中，你已经将所有应用程序在单个虚拟机镜像中运行，每个服务都作为 Docker 容器运行。你现在要通过将你的数据库服务器（PostgreSQL）和缓存服务器（Redis）从 Docker 分离到 Amazon 的云中来改变这种状况。所有其他服务将作为在单节点 Amazon ECS 集群中运行的 Docker 容器继续运行。图 10.1 显示了向 Amazon 云部署 EagleEye 服务。

让我们看看图 10.1，并深入了解更多细节：

- 所有的 EagleEye 服务（减去数据库和 Redis 集群）将被部署为在单节点 ECS 集群内部运行的 Docker 容器。ECS 配置并设置运行 Docker 集群所需的所有服务器。ECS 还可以监控在 Docker 中运行的容器的健康状况，并在服务崩溃时重新启动服务。
- 通过部署到 Amazon 云，你将不再使用自己的 PostgreSQL 数据库和 Redis 服务器，而是使用 Amazon RDS 和 Amazon ElastiCache 服务。你可以继续运行 Docker 中的 Postgres 和 Redis 数据存储，但是我想强调从你拥有和管理的基础

设施转移到由云提供商（在这种情况下，Amazon）完全管理的基础设施是多么容易。在实际部署中，在 Docker 容器之前，通常不会将数据库基础设施部署到虚拟机。

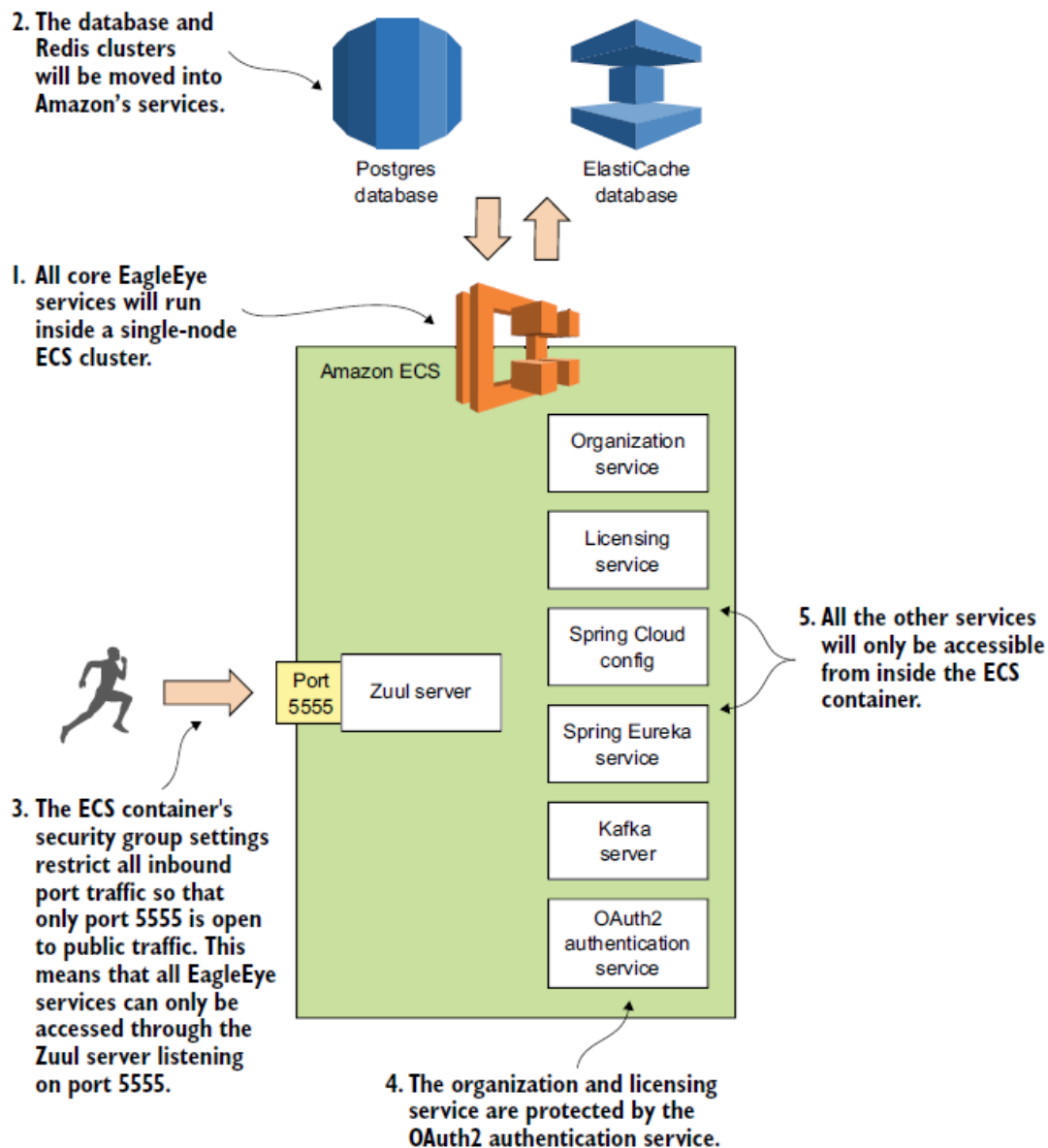


图 10.1 通过使用 Docker，你的所有服务都可以部署到云提供商，例如 Amazon ECS。

① All core EagleEye services will run inside a single-node ECS cluster.

所有 EagleEye 核心服务都将在单节点 ECS 群集内运行。

② The database and Redis clusters will be moved into Amazon's services.

数据库和 Redis 集群将被转移到 Amazon 的服务中。

③ *The ECS container's security group settings restrict all inbound port traffic so that only port 5555 is open to public traffic. This means that all EagleEye services can only be accessed through the Zuul server listening on port 5555.*

ECS 容器的安全组设置限制所有入站端口流量，以便只有端口 5555 对公共流量开放。这意味着所有的 EagleEye 服务只能通过监听端口 5555 的 Zuul 服务器访问。

④ *The organization and licensing service are protected by the OAuth2 authentication service.*

组织服务和许可服务受 OAuth2 认证服务的保护。

⑤ *All the other services will only be accessible from inside the ECS container.*

所有其他服务只能从 ECS 容器内访问。

- 与桌面部署不同，你希望服务器的所有流量都通过你的 Zuul API 网关。你将使用 Amazon 安全组来仅允许部署的 ECS 群集上的端口 5555 可供全球访问。
- 你仍将使用 Spring 的 OAuth2 服务器来保护你的服务。在可以访问组织服务和许可服务之前，用户需要使用认证服务进行验证（有关详细信息，请参阅第 7 章），并在每个服务调用中提供一个有效的 OAuth2 令牌。
- 包括你的 Kafka 服务器在内的所有服务器都不能通过它们暴露的 Docker 端口公开访问。

#### 工作的一些先决条件

要设置你的 Amazon 基础设施，你将需要做以下准备：

- 你自己的 Amazon Web Services ( AWS ) 账户。你应该对 AWS 控制台以及在该环境中工作的概念有基本的了解。
- 一个网页浏览器。对于手动设置，你将在控制台上设置所有内容。

- 使用 Amazon ECS 命令行客户端(<https://github.com/aws/amazon-ecsccli>)进行部署。

如果你没有使用 Amazon 的 Web 服务的经验,我会创建一个 AWS 账户并将安装列表中的工具。我也花时间熟悉这个平台。

如果你完全不熟悉 AWS,我强烈建议你选择一本 Michael 和 Andreas Wittig 的书《Amazon Web Services in Action》(Manning, 2015)。本书的第一章(<https://www.manning.com/books/amazon-web-services-inaction#downloads>)可供下载,并在本章最后的章节中提供了一个精心编写的教程,介绍如何注册并配置你的 AWS 账户。《Amazon Web Services in Action》是一本精心编写和全面的 AWS 书。尽管我一直在使用 AWS 环境多年,但我仍然觉得它是一个有用的资源。

最后,在本章中,我尽可能地尝试使用 Amazon 提供的免费服务。唯一我不能做到这一点的地方是设置 ECS 集群。我使用了一台 t2.large 服务器,每小时运行约 10 美分。如果你不想承担重大费用,确保在完成后关闭服务。

**注意:**如果你想自己运行此代码,则无法保证本章中使用的 Amazon 资源(Postgres, Redis 和 ECS)可用。如果你要运行本章的代码,你需要设置你自己的 GitHub 存储库(用于你的应用程序配置),你自己的 Travis CI 账户, Docker Hub(用于 Docker 镜像)和 Amazon 账户,然后修改你的应用程序配置以指向你的账户和凭证。

### 10.1.1.使用 Amazon RDS 创建 PostgreSQL 数据库

在开始本节之前,你需要创建和配置你的 Amazon AWS 账户。完成之后,你的第一个任务就是创建你要用于 EagleEye 服务的 PostgreSQL 数据库。为此,你将要登录到 Amazon AWS 控制台(<https://aws.amazon.com/console/>)并执行以下操作:

- 当你第一次登录到控制台时，你将看到一个 Amazon Web 服务列表。找到名为 RDS 的链接。点击链接，这将带你到 RDS 仪表板。
- 在仪表板上，你会看到一个大按钮“启动数据库实例”。点击它。
- Amazon RDS 支持不同的数据库引擎。你应该看到一个数据库列表。选择 PostgreSQL，然后单击“选择”按钮。这将启动数据库创建向导。

Amazon 数据库创建向导首先要问你的是这是一个生产数据库还是开发/测试数据库。

你将使用免费服务创建开发/测试数据库。图 10.2 显示了这个屏幕。

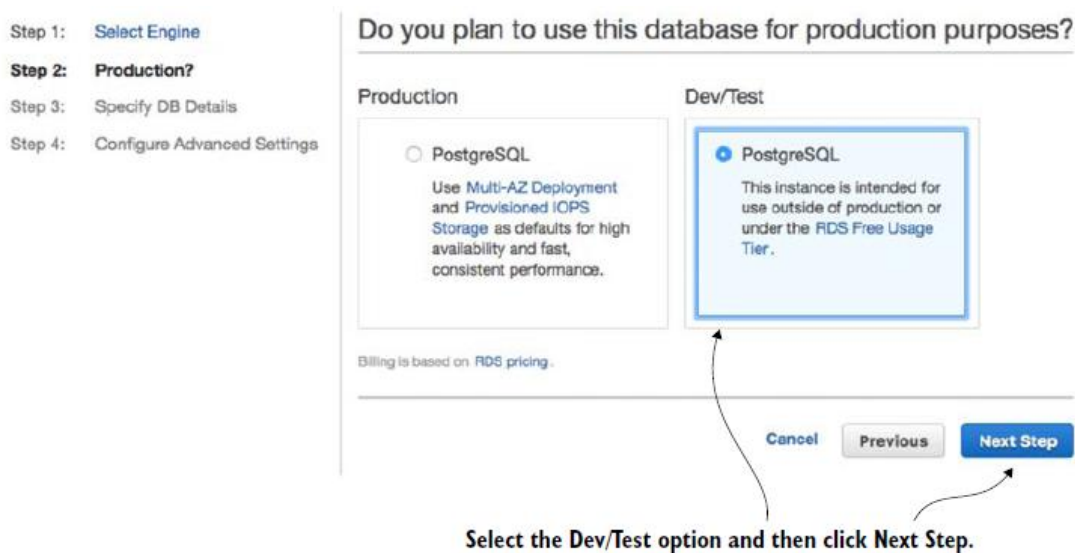


图 10.2 选择数据库是要生产数据库还是测试数据库

① *Select the Dev/Test option and then click Next Step.*

选择开发/测试选项，然后单击下一步。

接下来，你将设置有关 PostgreSQL 数据库的基本信息，并设置要用于登录数据库的主用户标识和密码。图 10.3 显示了这个屏幕。

① *Pick a db.t2.micro. It's the smallest free database and will more than meet your needs. You won't need a multi-AZ deployment.*

选择一个 db.t2.micro。这是最小的免费数据库，将超过你的需求。你不需要多可用地区部

署。

② *Make note of your password. For our examples you' ll use the master to login into the database. In a real system, you' d create a user account specific to the application and never directly use the master user ID/password for the app.*

记下你的密码。对于我们的示例，你将使用 master 登录到数据库。在真实的系统中，你应该为特定的应用程序创建一个用户账户，并且不要直接使用应用程序的主用户 ID/密码。

The Amazon RDS Free Tier provides a single db.t2.micro instance as well as up to 20 GB of storage, allowing new AWS customers to gain hands-on experience with Amazon RDS. Learn more about the RDS Free Tier and the instance restrictions [here](#).

☐ Only show options that are eligible for RDS Free Tier

### Instance Specifications

|                     |                                 |
|---------------------|---------------------------------|
| DB Engine           | postgres                        |
| License Model       | postgresql-license              |
| DB Engine Version   | 9.5.4                           |
| DB Instance Class   | db.t2.micro — 1 vCPU, 1 GiB RAM |
| Multi-AZ Deployment | No                              |
| Storage Type        | General Purpose (SSD)           |
| Allocated Storage*  | 5 GB                            |

**Warning:** Provisioning less than 100 GB of General Purpose (SSD) storage for high throughput workloads could result in higher latencies upon exhaustion of the initial General Purpose (SSD) IO credit balance. [Click here](#) for more details.

### Settings

|                         |                   |
|-------------------------|-------------------|
| DB Instance Identifier* | eagle-eye-aws-dev |
| Master Username*        | postgres_aws_dev  |
| Master Password*        | .....             |
| Confirm Password*       | .....             |

\* Required

Buttons: Cancel, Previous, Next Step

Annotations:

- Pick a db.t2.micro. It's the smallest free database and will more than meet your needs. You won't need a multi-AZ deployment.
- Retype the value you specified for Master Password.

*Make note of your password. For our examples you'll use the master to login into the database. In a real system, you'd create a user account specific to the application and never directly use the master user ID/password for the app.*

图 10.3 设置基本的数据库配置



向导的最后一步是设置数据库安全组，端口信息和数据库备份信息。图 10.4 显示了这个屏幕的内容。

**Configure Advanced Settings**

**Network & Security**

- VPC\*: Default VPC (vpc-fa0dd89d)
- Subnet Group: default
- Publicly Accessible: Yes
- Availability Zone: No Preference
- VPC Security Group(s): Create new Security Group default (VPC)

**Database Options**

- Database Name: eagle\_eyo\_aws\_dev
- Database Port: 5432
- DB Parameter Group: default.postgres9.5
- Option Group: default:postgres-9-5
- Copy Tags To Snapshots: ☐
- Enable Encryption: No

**Backup**

- Backup Retention Period: 0 days
- Backup Window: No Preference

**Monitoring**

- Enable Enhanced Monitoring: No

**Maintenance**

- Auto Minor Version Upgrade: Yes
- Maintenance Window: No Preference

\* Required

Buttons: Cancel, Previous, Launch DB Instance

**Annotations:**

- For now, you'll create a new security group and allow the database to be publicly accessible.
- Note the database name and the port number. The port number will be used as part of your service's connect string.
- As this is a dev database, you can disable backups.
- Select the period in which you want pending modifications (such as changing the DB instance class) or patches applied to the DB instance by Amazon RDS. Any such maintenance should be started and completed within the selected period. If you do not select a period, Amazon RDS will assign a period randomly. [Learn More.](#)

图 10.4 为 RDS 数据库设置安全组，端口和备份选项

① For now, you' ll create a new security group and allow the database to be publicly accessible.

现在，你将创建一个新的安全组，并允许公开访问数据库。

② *Note the database name and the port number. The port number will be used as part of your service's connect string.*

记下数据库名称和端口号。端口号将用作服务连接字符串的一部分。

③ *As this is a dev database, you can disable backups.*

由于这是一个开发数据库，你可以禁用备份。

此时，您的数据库创建过程将开始（可能需要几分钟）。完成后，你需要配置 Eagle Eye 服务以使用数据库。数据库创建后（这将需要几分钟），你将回到 RDS 仪表板并查看你已创建的数据库。图 10.5 显示了这个屏幕。

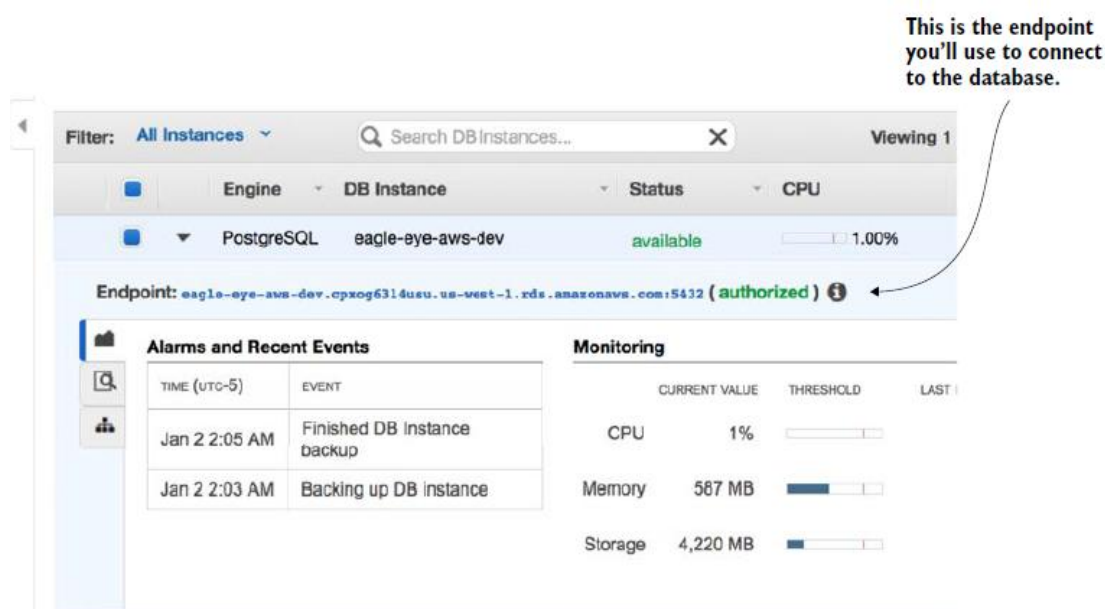


图 10.5 你创建的 Amazon RDS/PostgreSQL 数据库

① *This is the endpoint you'll use to connect to the database.*

这是你将用于连接到数据库的端点。

在本章中，我为每个需要访问基于 Amazon 的 PostgreSQL 数据库的微服务创建了一个名为 aws-dev 的新应用程序配置文件。我在包含 Amazon 数据库连接信息的 Spring Cloud Config GitHub 存储库(<https://github.com/carnellj/config-repo>)中添加了一个新的 Spring Cloud Config 服务器应用程序配置文件。属性文件遵循使用新数据库（许可

服务、组织服务和认证服务) 的每个属性文件中的命名约定( service-name )-aws-dev.yml。

此刻，你的数据库已经准备好了（大约需要点击 5 次才能设置）。让我们转到下一个应用程序基础设施，看看如何创建你的 EagleEye 许可服务将要使用的 Redis 集群。

### 10.1.2. 在 Amazon 创建 Redis 集群

要创建 Redis 集群，你将使用 Amazon ElastiCache 服务。Amazon ElastiCache 允许您使用 Redis 或 Memcached 构建内存数据缓存(<https://memcached.org/>)。对于 EagleEye 服务，你将把在 Docker 中运行的 Redis 服务器迁移到 ElastiCache。

首先，回到 AWS 控制台的主页面（点击页面左上方的橙色立方体），然后单击 ElastiCache 链接。

Create your Amazon ElastiCache cluster

Cluster engine ☒ Redis  
In-memory data structure store used as database, cache and message broker. ElastiCache for Redis offers Multi-AZ with Auto-Failover and enhanced robustness.  
☐ Cluster Mode enabled (Scale Out)

☐ Memcached  
High-performance, distributed memory object caching system, intended for use in speeding up dynamic web applications.

Redis settings

Name  ⓘ This is the name of your ElastiCache server.

Engine version compatibility  ⓘ

Port  ⓘ

Parameter group  ⓘ

Node type  ⓘ The smallest instance type is selected here.

Number of replicas  ⓘ

Advanced Redis settings

Cancel Create

As this is a dev server, you don't need to create replicas of the Redis servers.

图 10.6 只需点击几下，你就可以建立一个由 Amazon 管理基础设施的 Redis 群集。

① This is the name of your ElastiCache server.

这是 ElastiCache 服务器的名称。

② *The smallest instance type is selected here.*

这里选择最小的实例类型。

③ *As this is a dev server, you don't need to create replicas of the Redis servers.*

由于这是一个开发服务器，你不需要创建 Redis 服务器的副本。

从 ElastiCache 控制台中，选择 Redis 链接（屏幕的左侧），然后点击屏幕顶部的蓝色创建按钮。这将启动 ElastiCache/Redis 创建向导。

图 10.6 显示了 Redis 创建屏幕。

在你填写完所有数据后，点击创建按钮。Amazon 将开始 Redis 集群创建过程（这将需要几分钟的时间）。Amazon 将在最小的 Amazon 服务器实例上构建运行的单节点 Redis 服务器。一旦你点击按钮，你会看到你的 Redis 集群正在创建。创建集群后，你可以单击集群的名称，它会显示集群中使用的端点的详细屏幕。图 10.7 显示了 Redis 集群创建后的细节。

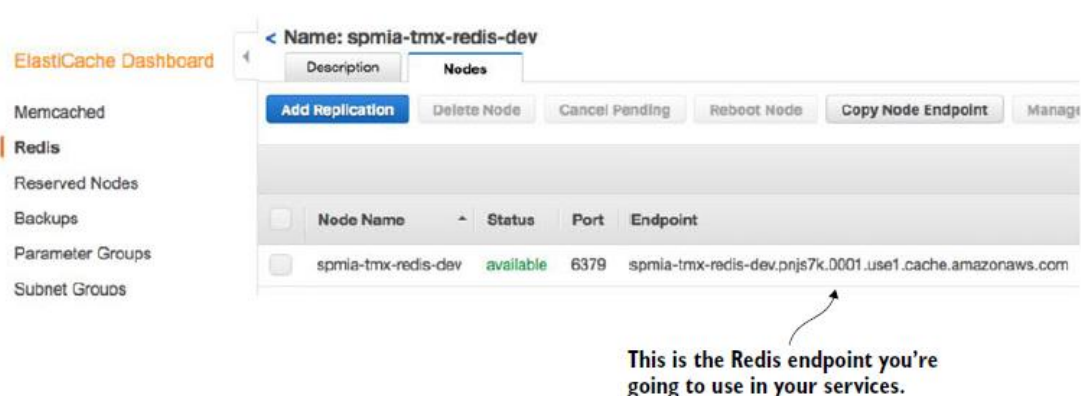


图 10.7 Redis 端点是你的服务连接到 Redis 的关键信息。

① *This is the Redis endpoint you're going to use in your services.*

这是你要在服务中使用的 Redis 端点。

许可服务是你使用 Redis 的唯一服务，因此请确保如果你将本章中的代码示例部署到

你自己的 Amazon 实例中，则可以适当地修改许可服务的 Spring Cloud Config 文件。

### 10.1.3. 创建 ECS（弹性可伸缩的计算服务）集群

部署 EagleEye 服务之前的最后一步是创建 Amazon ECS 集群。创建一个 Amazon ECS 集群提供 Amazon 机器，它托管你的 Docker 容器。要做到这一点，你将再次访问 Amazon AWS 控制台。从那里你将点击 Amazon EC2 容器服务链接。

这会将你带到主要的 EC2 容器服务页面，你将在其中看到“Getting Started”按钮。

点击“开始”按钮。这将带你进入如图 10.8 所示的“选择配置选项”屏幕。

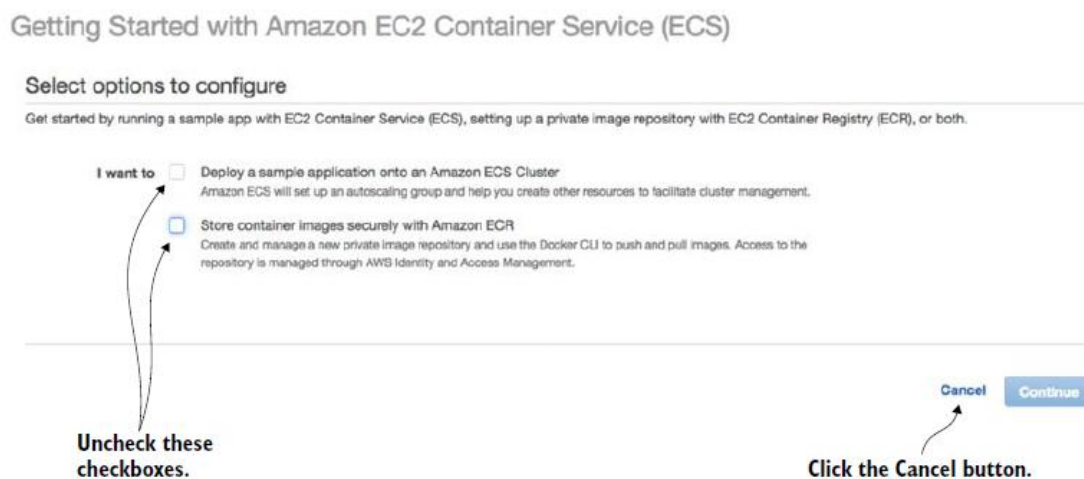


图 10.8 ECS 提供了一个向导来引导一个新的服务容器。 你不会用它。

① *Uncheck these checkboxes.*

取消选中这些复选框。

② *Click the Cancel button.*

点击取消按钮。

取消勾选屏幕上的两个复选框，然后点击取消按钮。ECS 提供了一个基于一组预定义模板来设置 ECS 容器的向导。你不打算使用这个向导。一旦你取消了 ECS 设置向导，你应该会看到 ECS 主页上的“集群”选项卡。图 10.9 显示了这个屏幕。点击“创建集群”按钮开

始创建 ECS 集群的过程。

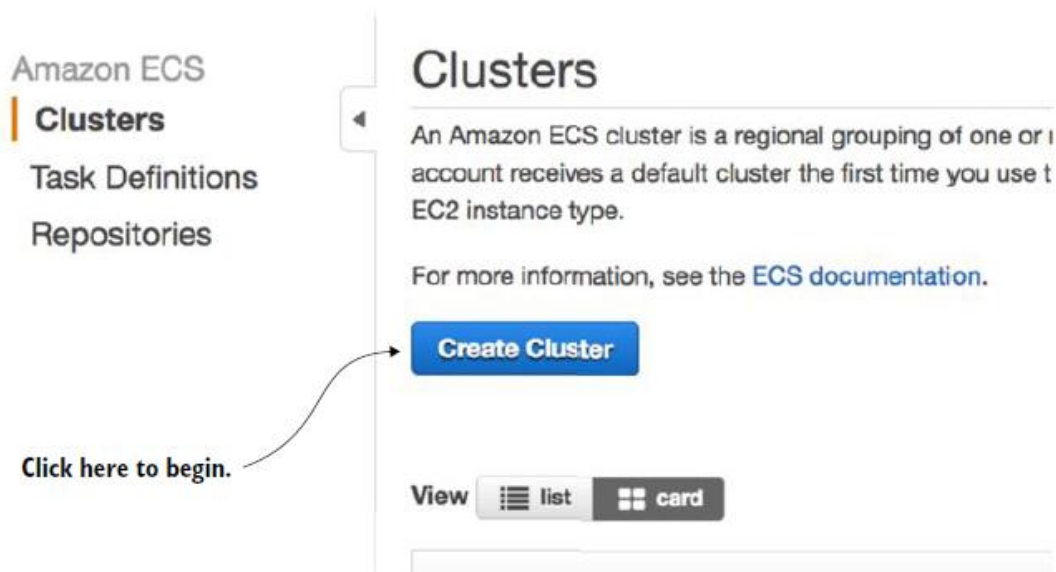


图 10.9 开始创建 ECS 集群的过程

① *Click here to begin.*

点击这里开始。

现在，你将看到一个名为“创建集群”的屏幕，其中包含三个主要部分。第一部分将定义基本的集群信息。在这里你将进入：

- 你的 ECS 集群的名称。
- 你要在其中运行集群的 Amazon EC2 虚拟机的大小
- 你要在集群中运行的实例数。
- 你要分配给集群中的每个节点的弹性块存储（EBS）磁盘空间量

① *You can choose a t2.large server because of its large amount of memory (8 GB) and low hourly cost (.094 cents per hour).*

你可以选择一个 t2.large 服务器，因为它具有大量的内存（8 GB）和较低的每小时成本（0.094 美分/小时）。

② *As this is a dev environment, you're going to run with just a single instance.*



由于这是一个开发环境，所以你将只运行一个实例。

③ *Make sure you define the SSH key pair or you won't be able to SSH onto the box to diagnose problems.*

确定你定义了 SSH 密钥对，否则你将无法通过 SSH 进入该对话框诊断问题。

## Create Cluster

When you run tasks using Amazon ECS, you place them on a cluster, which is a logical grouping of EC2 instances. This wizard will guide you through naming your cluster, and then configuring the container instances that your tasks can be placed on, the security group for your container instances so that they can make calls to the AWS APIs on your behalf.

**Cluster name\*** spmia-tmx-dev ⓘ

☐ Create an empty cluster

**EC2 instance type\*** t2.large ⓘ

**Number of instances\*** 1 ⓘ

**EC2 Ami Id\*** amzn-ami-2016.09.c-amazon-ecs-optimized [ami-1eda8d7e] ⓘ

**EBS storage (GiB)\*** 22 ⓘ

**Key pair** spmia-tmx ⓘ

You will not be able to SSH into your EC2 instances without a key pair.  
You can create a new key pair in the [EC2 console](#).

**Annotations:**

- You can choose a t2.large server because of its large amount of memory (8 GB) and low hourly cost (.094 cents per hour).
- As this is a dev environment, you're going to run with just a single instance.
- Make sure you define the SSH key pair or you won't be able to SSH onto the box to diagnose problems.

图 10.10 在“创建集群”屏幕上，用于托管 Docker 集群的 EC2 实例的大小。

图 10.10 显示了我为本书中的测试示例填充的屏幕。

**注意：**你在创建 Amazon 账户时所做的首要任务之一是定义一个密钥对，其使用 SSH 进入你启动的任何 EC2 服务器。我们不打算在本章中讨论如何创建一个密钥对，但如果你以前从未这样做，我建议你看看 Amazon 关于这方面的说明(<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html>)。

接下来，你将要为 ECS 集群设置网络配置。图 10.11 显示了网络配置屏幕和你正在配置的值。

首先要注意的是，选择 ECS 集群将运行的 Amazon Virtual Private Cloud (VPC)。

默认情况下，ECS 设置向导将提供建立一个新的 VPC。我已经选择在我的默认 VPC 中运行 ECS 集群。默认的 VPC 包含数据库服务器和 Redis 集群。在 Amazon 云中，Amazon 托管的 Redis 服务器只能由与 Redis 服务器处于同一 VPC 的服务器访问。

接下来，你必须选择 VPC 中要访问 ECS 集群的子网。由于每个子网都对应一个 Amazon 可用性区域，因此我通常会选择 VPC 中的所有子网以使集群可用。

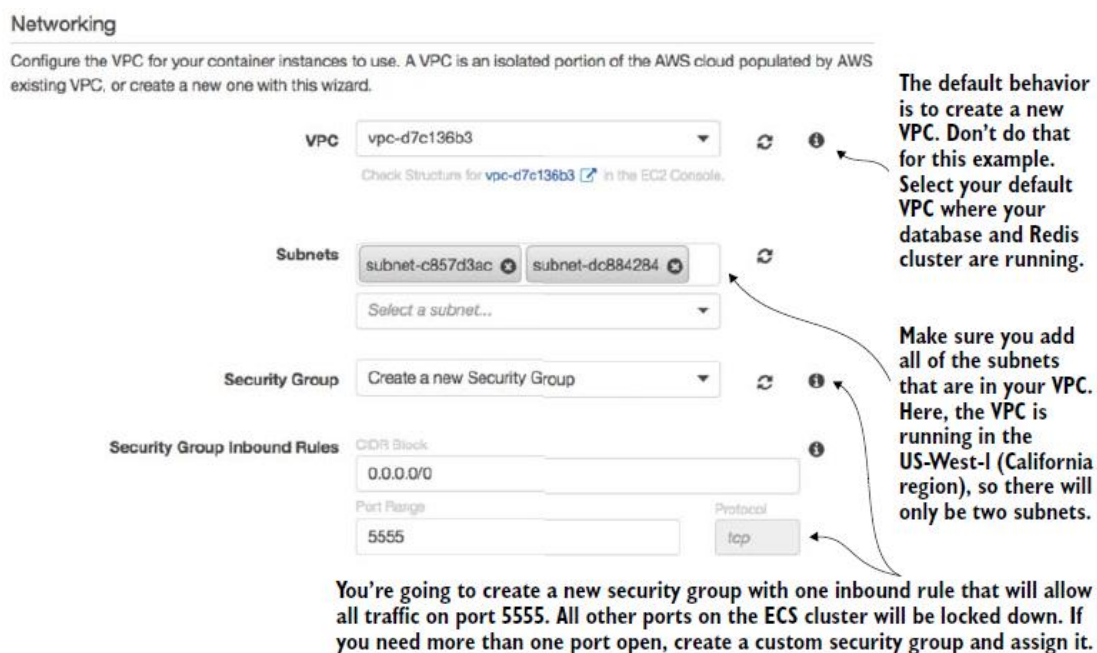


图 10.11 创建服务器后，配置用于访问它们的网络/AWS 安全组。

① *The default behavior is to create a new VPC. Don't do that for this example. Select your default VPC where your database and Redis cluster are running.*

默认行为是创建一个新的 VPC。在这个例子不要这样做。选择你的数据库和 Redis 集群正在运行的默认 VPC。

② *Make sure you add all of the subnets that are in your VPC. Here, the VPC is running in the US-West-1 (California region), so there will only be two subnets.*

确保你添加了 VPC 中的所有子网。在这里，VPC 运行在美国西部 1（加州地区），所以只有两个子网。

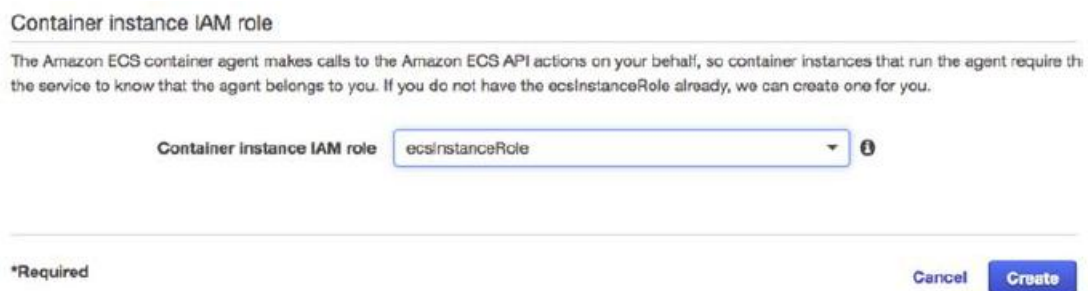


③ *You' re going to create a new security group with one inbound rule that will allow all traffic on port 5555. All other ports on the ECS cluster will be locked down. If you need more than one port open, create a custom security group and assign it.*

你将创建一个新的安全组，其中一个入站规则将允许端口 5555 上的所有流量。ECS 集群上的所有其他端口将被锁定。如果你需要打开多个端口，请创建一个自定义安全组并分配它。

最后，你必须选择创建新的安全组，或选择你已创建的现有 Amazon 安全组以应用于新的 ECS 集群。因为你正在运行 Zuul，你希望所有的流量都通过一个端口，即 5555 端口。你将要配置由 ECS 向导创建的新安全组，以允许来自世界的所有入境流量（0.0.0.0/0 是整个互联网的网络掩码）。

最后一步必须填写的是为在服务器上运行的 ECS 容器代理创建 Amazon IAM 角色。ECS 代理负责与 Amazon 通信，了解服务器上运行的容器的状态。你将允许 ECS 向导为你创建一个名为 `ecsInstanceRole` 的 IAM 角色。图 10.12 显示了这个配置步骤。



Container instance IAM role

The Amazon ECS container agent makes calls to the Amazon ECS API actions on your behalf, so container instances that run the agent require the service to know that the agent belongs to you. If you do not have the `ecsInstanceRole` already, we can create one for you.

Container instance IAM role

\*Required

Cancel Create

图 10.12 配置容器 IAM 角色

此时你应该看到一个跟踪集群创建状态的屏幕。一旦创建了集群，你应该在屏幕上看到一个名为“查看集群”的蓝色按钮。点击“查看集群”按钮。图 10.13 显示了在按下“查看集群”按钮之后将出现的屏幕。

此时，你拥有成功部署 EagleEye 微服务所需的全部基础设施。

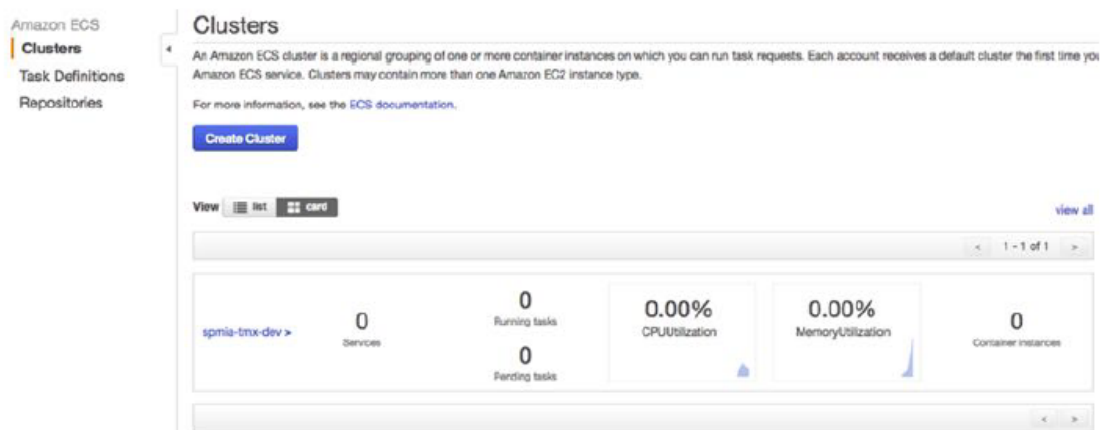


图 10.13 ECS 集群正在运行

### 关于基础设施设置和自动化

现在，你正在通过 Amazon AWS 控制台执行所有操作。在真实环境中，你可以使用 Amazon 的 CloudFormation 脚本 DSL（域特定语言）或 HashCorp 的 Terraform（<https://www.terraform.io/>）等云基础设施脚本工具创建所有这些基础设施。然而，这是一个完整的话题，远远超出了本书的范围。如果你使用 Amazon 的云，你可能已经熟悉 CloudFormation。如果你是 Amazon 云的新手，我建议你先花点时间学习，然后再通过 Amazon AWS 控制台创建核心基础设施。

再次，我想请读者回到 Michael 和 Andreas Wittig 的《Amazon Web Services in Action》（Manning，2015）。他们了解大多数 Amazon Web Services，并演示如何使用 CloudFormation（带有示例）自动创建基础设施。

## 10.2. 除了基础设施：部署 EagleEye

现在你已经建立了基础设施，现在可以进入本章的后半部分。在第二部分中，你将把 EagleEye 服务部署到你的 Amazon ECS 容器。你要做两件事。你工作的第一部分是对于那些不耐烦的（像我一样），并将展示如何手动部署 EagleEye 到你的 Amazon 实例。这将帮助你了解部署服务的机制，并查看在容器中运行的部署服务。尽管弄脏你的手并手动部署你的

服务是有趣的，但它是不可持续的或推荐的。

这是本节的第二部分开始发挥作用的地方。你将自动执行整个构建和部署流程，并将人员排除在外。这是你的目标最终状态，通过演示如何设计、构建和将微服务部署到云，确实减轻了你在本书中所做的工作。

### 10.2.1. 在 ECS 中手动部署 EagleEye 服务

要手动部署你的 EagleEye 服务，你将要切换并离开 Amazon AWS 控制台。要部署 EagleEye 服务，你将使用 Amazon ECS 命令行客户端(<https://github.com/aws/amazon-ecs-cli>)。安装 ECS 命令行客户端之后，需要将 ecs-cli 运行时环境配置为：

- 使用你的 Amazon 凭证配置 ECS 客户端
- 选择客户端将要工作的地区
- 定义 ECS 客户端需要处理的默认 ECS 集群
- 这个工作是通过运行 `ecs-cli configure` 命令完成的：

```
ecs-cli configure --region us-west-1 \  
                  --access-key $AWS_ACCESS_KEY \  
                  --secret-key $AWS_SECRET_KEY \  
                  --cluster spmia-tmx-dev
```

`ecs-cli configure` 命令将设置你的集群所在的区域，你的 Amazon 访问密钥和秘密密钥，以及你部署到的集群名称（`spmia-tmx-dev`）。如果你查看前面的命令，我正在使用环境变量（`$AWS_ACCESS_KEY` 和 `$AWS_SECRET_KEY`）来保存我的 Amazon 访问密钥和秘密密钥。

**注意：**我选择了美国西部 1 区域纯粹是出于示范的目的。根据你所在的国家，你可以选择一个更适合你的 Amazon 区域。

接下来，让我们看看如何做一个构建。与其他章节不同，你必须设置构建名称，因为

本章中的 Maven 脚本将在本章稍后设置的 builddeploy 管道中使用。你将要设置一个名为 \$BUILD\_NAME 的环境变量。\$BUILD\_NAME 环境变量用于标记由构建脚本创建的 Docker 镜像。切换到从 GitHub 下载的第 10 章代码的根目录，并执行以下两个命令：

```
export BUILD_NAME=TestManualBuild
mvn clean package docker:build
```

这将使用位于项目目录根目录下的父 POM 来执行 Maven 构建。父 pom.xml 被设置来构建你将在本章中部署的所有服务。Maven 代码完成执行后，你可以将 Docker 镜像部署到你在前面 10.1.3 节中设置的 ECS 实例。要进行部署，请执行以下命令：

```
ecs-cli compose --file docker/common/docker-compose.yml up
```

ECS 命令行客户端允许你使用 Docker-compose 文件部署容器。通过允许你从桌面开发环境重用 Docker-compose 文件，Amazon 已经大大简化了将服务部署到 Amazon ECS。

ECS 客户端运行后，可以通过执行以下命令验证服务正在运行，发现服务器的 IP 地址：

```
ecs-cli ps
```

图 10.14 显示了 ecs-cli ps 命令的输出。

These are ports that are mapped in the Docker containers. However, only port 5555 is open to the outside world.

| Name  | State   | Ports  |
|---|---------|--|
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/authentication-service | RUNNING | 54.153.112.116:8901->8901/tcp                                |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/organization-service   | RUNNING | 54.153.112.116:8085->8085/tcp                                |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/kafka-server           | RUNNING | 54.153.112.116:2181->2181/tcp, 54.153.112.116:9092->9092/tcp |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/licensing-service      | RUNNING | 54.153.112.116:8080->8080/tcp                                |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/zuul-server            | RUNNING | 54.153.112.116:5555->5555/tcp                                |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/eureka-server          | RUNNING | 54.153.112.116:8761->8761/tcp                                |
| bfd5d7f7-515a-4ff5-b848-f3bb60bd9996/config-server          | RUNNING | 54.153.112.116:8888->8888/tcp                                |

Individual docker services deployed.

IP addresses of the deployed services.

图 10.14 检查已部署服务的状态

① *These are ports that are mapped in the Docker containers. However, only port 5555 is open to the outside world.*

这些是在 Docker 容器中映射的端口。但是，只有 5555 端口对外开放。

② *Individual docker services deployed.*

单独部署的 Docker 服务。

③ *IP addresses of the deployed services.*

已部署服务的 IP 地址。

请注意图 10.14 中输出的三个事项：

- 你可以看到已经部署了七个 Docker 容器，每个 Docker 容器都运行你的一个服务。
- 你可以看到 ECS 集群的 IP 地址（54.153.122.116）。
- 看起来你不仅仅打开 5555 端口。事实并非如此。图 10.14 中的端口标识符是 Docker 容器的端口映射。但是，唯一对外开放的端口是 5555 端口。请记住，设置 ECS 集群时，ECS 设置向导会创建一个仅允许来自端口 5555 的流量的 Amazon 安全组。

此时，你已经成功将第一组服务部署到 Amazon ECS 客户端。现在，让我们看看如何设计一个构建和部署管道，以便将编译、打包和部署你的服务到 Amazon 的过程自动化。

#### 调试为什么 ECS 容器不启动或待机

ECS 有有限的工具来调试为什么容器不能启动。如果你在 ECS 部署的服务启动或待机时遇到问题，你需要通过 SSH 进入到 ECS 集群来查看 Docker 日志。为此，你需要将端口 22 添加到运行 ECS 集群的安全组，然后使用你在集群设置时定义的 Amazon 密钥对（请参见图 10.9）将该 SSH 添加到该框中，作为 ec2 用户。一旦你在服务器上，你可以通过运行 `docker ps` 命令来获得在服务器上运行的所有 Docker 容器的列表。找到要调试的容器镜像后，可以运行 `docker logs -f <<container id>>` 命令来跟踪目标 Docker 容器的日志。

这是调试应用程序的基本机制，但有时您只需要登录到服务器并查看实际的控制台输出以确定发生了什么。

### 10.3. 构建/部署管道的架构

本章的目标是为你提供构建/部署管道的工作组件,以便你可以定制这些组件以适应特定的环境。

让我们开始我们的讨论,看看你的构建部署管道的一般架构以及它所代表的一些一般模式和主题。为了保持这些例子的流畅性,我做了一些我通常不会在自己的环境中做的事情,我会相应地调用这些组件

我们关于部署微服务的讨论将从你在第 1 章中看到的图片开始。图 10.15 是我们在第 1 章中看到的图的副本,并显示了构建微服务构建和部署管道所涉及的部分和步骤。

图 10.15 应该看起来有些熟悉,因为它基于用于实现持续集成 ( CI ) 的一般构建 - 部署模式:

- 开发人员将其代码提交给源代码库。
- 构建工具监视源代码版本控制存储库的更改,并在检测到更改时启动构建。
- 在构建期间,运行应用程序的单元和集成测试,如果都通过,将创建可部署的软件构件 ( JAR , WAR 或 EAR )。
- 这个 JAR , WAR 或 EAR 可能会被部署到运行在服务器上的应用程序服务器上 ( 通常是一个开发服务器 )。

① *A developer commits service code to a source repository.*

开发人员将服务代码提交到源代码存储库。

② *The build/deploy engine checks out the code and runs the build scripts.*

构建/部署引擎检出代码并运行构建脚本。

③ *Engine compiles code, runs tests, and creates an executable artifact(JAR file with self-contained server).*

引擎编译代码、运行测试，并创建一个可执行的构件（具有独立服务器的 JAR 文件）。

④ *A virtual machine image (container) is created, with the service and its run-time engine installed.*

创建虚拟机镜像（容器），并安装该服务及其运行时引擎。

⑤ *Platform tests are run against the machine image before it can be promoted to the new environment.*

在升级到新环境之前，平台测试是针对机器镜像运行的。

⑥ *Before the machine image can be promoted to the next environment, platform tests for that environment must be run.*

在机器镜像可以升级到下一个环境之前，必须运行该环境的平台测试。

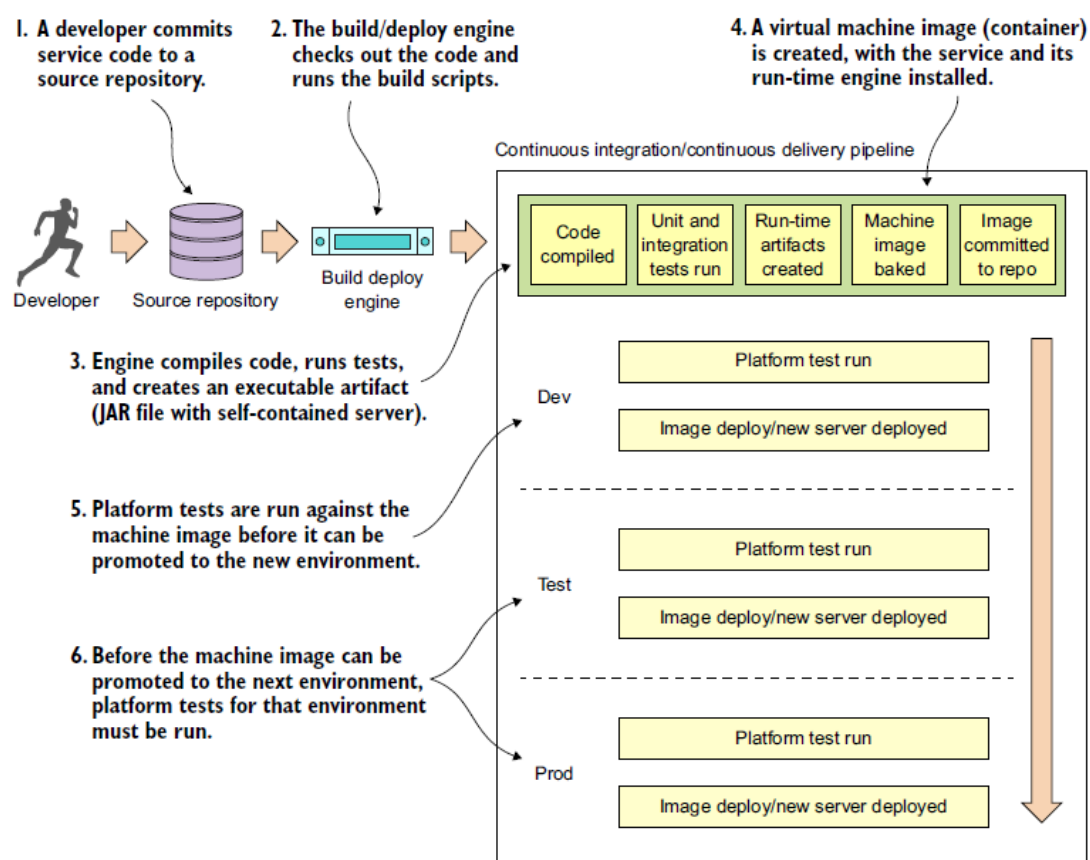


图 10.15 构建和部署管道中的每个组件都会自动执行一个本来可以手动完成的任务。

通过构建和部署流水线（如图 10.15 所示），可以进行类似的流程，直到代码准备好部

署。在图 10.15 所示的构建和部署过程中，你将在这个流程进行持续交付（CD）：

- 开发人员将其服务代码提交到源代码库。
- 构建/部署引擎监视源代码库的更改。如果代码已被提交，构建/部署引擎将检出代码并运行代码的构建脚本。
- 构建/部署过程的第一步是编译代码，运行其单元和集成测试，然后将该服务编译为可执行的构件。由于你的微服务是使用 Spring Boot 构建的，因此你的构建过程将创建一个包含服务代码和内嵌的 Tomcat 服务器的可执行 JAR 文件。
- 这是你的构建/部署管道开始偏离传统的 Java CI 构建过程的地方。在构建可执行 JAR 之后，你将使用部署到其中的微服务来“制作”机器镜像。这个制作过程将基本上创建一个虚拟机镜像或容器（Docker）并将你的服务安装到它上面。虚拟机镜像启动后，你的服务将启动并准备开始接受请求。与传统的 CI 构建过程不同，你可能（也就是说可能）将已编译的 JAR 或 WAR 部署到应用程序服务器，这个应用程序服务器是独立（通常由一个单独的团队）管理的应用程序，并且用 CI/CD 过程来部署微服务，服务的运行时引擎和机器镜像都是由编写该软件的开发团队管理的一个单独的单元。
- 在正式部署到新环境之前，将启动机器镜像，并针对正在运行的镜像运行一系列平台测试，以确定是否一切正常运行。如果平台测试通过，则机器镜像被升级到新的环境并可供使用。
- 在将服务升级到下一个环境之前，必须运行环境的平台测试。将服务升级到新环境中，需要启动在较低环境中使用到下一环境的确切的机器镜像。

这是整个过程的秘诀。整个机器镜像被部署。在创建服务器之后，不会对已安装的软件（包括操作系统）进行更改。通过升级并始终使用相同的机器镜像，可



以保证从一个环境升级到另一个环境时服务器的不变性。

### 单元测试 vs. 集成测试 vs. 平台测试

从图 10.15 中可以看到,在构建和部署一个服务的过程中,我做了几种类型的测试(单元,集成和平台)。在构建和部署管道中有三种典型的测试类型是:

- **单元测试**:在编译服务代码且在部署到环境之前,将立即运行单元测试。它们被设计成完全隔离,每个单元测试关注的范围都是小而有限的。单元测试不应该依赖于第三方基础设施,如数据库、服务等等。通常单元测试范围将包含单个方法或功能的测试。
- **集成测试**:打包服务代码后立即运行集成测试。这些测试旨在测试整个工作流程并 stub 或模拟需要从黑盒中取出的主要服务或组件。在集成测试期间,你可能正在运行内存数据库来保存数据,模拟第三方服务调用等等。集成测试测试整个工作流程或代码路径。对于集成测试,第三方依赖关系被模拟或 stub,以便调用远程服务的任何调用都被模拟或 stub,调用永远不会离开构建服务器。
- **平台测试**:平台测试在将服务部署到环境之前运行。这些测试通常测试整个业务流程,并调用通常在生产系统中调用的所有第三方依赖。平台测试在特定的环境中运行,不涉及任何模拟的服务。运行平台测试以确定第三方服务的集成问题,在集成测试期间,当第三方服务被删除时通常不会被检测到。

这个构建/部署过程构建在四个核心模式之上。这些模式不是我创建的,而是从开发团队构建微服务和基于云的应用程序的集体经验中涌现出来的。这些模式包括:

- **持续集成/持续交付 (CI/CD)**:使用 CI/CD,你的应用程序代码不仅在提交时被构建和测试,它也不断被部署。代码的部署应该是这样的:如果代码通过了它的单元、集成和平台测试,它应该被立即升级到下一个环境。大多数组织唯一的停止点是发

布到生产。

- **基础设施即代码**：将被发布到开发和远处的最终软件构件是机器镜像。在你的微服务的源代码被编译和测试之后，机器镜像和安装在其上的微服务将立即启动并提供服务。机器镜像的配置是通过一系列与每个版本一起运行的脚本来实现的。服务器建成后，不需要人为干预服务器。配置脚本保持在源代码版本控制下，并像其他任何代码一样进行管理。
- **不可变的服务器**：一旦创建了服务器镜像，在配置过程之后，服务器和微服务的配置就不会被触及。这可以保证你的环境不会因开发人员或系统管理员进行导致中断的“一个小小的更改”而导致“配置漂移”。如果需要进行更改，配置服务器的配置脚本将被更改，并触发新的构建。

#### 关于凤凰服务器的不变性和复活

有了不可变服务器的概念，我们应该始终保证服务器的配置与服务器的机器镜像完全匹配。服务器应该可以选择从机器镜像中终止并重新启动，而无需更改服务或微服务行为。这个终止和复活的新服务器被 Martin Fowler 称为“凤凰服务器” (<http://martinfowler.com/bliki/PhoenixServer.html>)，因为当旧服务器被终止时，新服务器应该从灰烬中拉起。凤凰服务器模式有两个关键的好处。

首先，它暴露并驱动配置离开你的环境。如果你不断地卸载和创建新的服务器，你很可能提前发现配置偏移。这对确保一致性有很大帮助。由于配置的漂移，我已经花费了太多的时间和生活，离开我的家人处理“紧急情况”。

其次，Phoenix 服务器模式通过帮助查找在服务器或服务被终止并重新启动后无法恢复的情况，有助于提高恢复能力。记住，在一个微服务架构中，你的服务应该是无状态的，服务器的死亡应该是一个小问题。随机查杀并重新启动服务器会快速暴露出你的服务或基础

设施中存在状态的情况。在部署管道的早期发现这些情况和依赖关系更好，而不是当你与一个愤怒的公司联系时。

我工作的公司使用 Netflix 的 Chaos Monkey(<https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey>)来随机选择和终止服务器。Chaos Monkey 是一个非常宝贵的工具，用于测试微服务环境的不变性和可恢复性。Chaos Monkey 随机选择你的环境中的服务器实例并终止它们。使用 Chaos Monkey 的想法是，你正在寻找无法从服务器崩溃中恢复的服务，并且在启动新服务器时，服务器的行为方式与被终止的服务器相同。

#### 10.4. 构建和部署管道

从第 10.3 节中介绍的通用架构中，可以看到构建/部署管道背后有许多移动部件。由于本书的目的是向你展示“实战”的内容，因此我们将详细介绍实现 EagleEye 服务的构建/部署流程的具体细节。图 10.16 列出了你将用来实现管道的不同技术：

- *GitHub (<http://github.com>)* : GitHub 是我们的源代码控制库。本书的所有应用程序代码都在 GitHub 中。选择 GitHub 作为源代码控制库有两个原因。首先，我不想管理和维护我自己的 Git 源代码控制服务器。其次，GitHub 提供了各种各样的 Web 钩子和强大的基于 REST 的 API，用于将 GitHub 集成到构建过程中。
- *Travis CI (<http://travis-ci.org>)* : Travis CI 是我用来构建和部署 EagleEye 微服务和配置 Docker 镜像的持续集成引擎。Travis CI 是一个基于云的，基于文件的 CI 引擎，易于设置，并且与 GitHub 和 Docker 具有很强的集成能力。虽然 Travis CI 不像 Jenkins(<https://jenkins.io>)这样的 CI 引擎功能全面，但对我们的应用来说已经足够了。我在 10.5 和 10.6 节中描述了使用 GitHub 和 Travis CI。
- *Maven/Spotify Docker Plugin (<https://github.com/spotify/docker-maven>)*

*plugin*) : 虽然我们使用 vanilla Maven 来编译、测试和打包 Java 代码,但是我们使用的关键 Maven 插件是 Spotify 的 Docker 插件。这个插件允许我们从 Maven 内部开始创建一个 Docker 构建。

- *Docker (<https://www.docker.com/>)* : 我选择 Docker 作为我们的容器平台有两个原因。首先, Docker 跨多个云提供商可移植。我可以采用相同的 Docker 容器, 并以最少的工作将其部署到 AWS, Azure 或 Cloud Foundry。其次, Docker 是轻量级的。在本书的最后, 你已经构建并部署了大约 10 个 Docker 容器 (包括数据库服务器、消息平台和搜索引擎)。在本地桌面上部署相同数量的虚拟机将是困难的, 因为每个镜像的大小和速度。Docker, Maven 和 Spotify 的设置和配置将不在本章中讨论, 而是在附录 A 中进行介绍。
- *Docker Hub (<https://hub.docker.com/>)* : 在构建服务并创建了 Docker 镜像之后, 它将使用唯一的标识符进行标记, 并将其推送到中央存储库。对于 Docker 镜像存储库, 我选择使用 Docker hub, 即 Docker 公司的公共镜像存储库。
- *Python (<https://python.org>)* : 为了编写部署 Docker 镜像之前执行的平台测试, 我选择了 Python 作为编写平台测试的工具。我坚信在工作中使用正确的工具, 坦率地说, 我认为 Python 是一个非常棒的编程语言, 特别是编写基于 REST 的测试用例。
- *Amazon' s EC2 Container Service (ECS)* : 我们微服务的最终目标是将 Docker 实例部署到 Amazon 的 Docker 平台。我选择 Amazon 作为我的云平台, 因为它是迄今为止最成熟的云提供商, 并且使得部署 Docker 服务变得微不足道。

**等等...你说 Python 吗?**

你可能会发现我用 Python 而不是 Java 写平台测试有点奇怪。我有目的地做了这个。

Python ( 如 Groovy ) 是编写基于 REST 的测试用例的绝妙脚本语言。我相信在工作中使用正确的工具。对于采用微服务的公司来说,我所见过的最大的思想转变之一是选择语言的责任应该在开发团队中。在太多的公司中,我看到了教条式的标准拥抱(“我们的企业标准是 Java ... ,所有的代码都必须用 Java 编写”)。因此,当 10 行的 Groovy 或 Python 脚本可以完成这个工作时,我已经看到开发团队跳过了一大堆 Java 代码。

我选择 Python 的第二个原因是,与单元和集成测试不同,平台测试是真正的“黑匣子”测试,你在这里扮演着一个在真实环境中运行实际 API 消费者的角色。单元测试运行最低级别的代码,运行时不应该有任何外部依赖。集成测试已经达到了一个水平并测试了 API,但是关键的外部依赖(例如对其他服务的调用,数据库调用等)却被模拟或删除。平台测试应该是对底层基础设施的真正的独立测试。

## 10.5. 开始你的构建/部署管道 : GitHub 和 Travis CI

很多的源代码控制引擎和构建部署引擎(内部部署和基于云)都可以实现你的构建和部署管道。对于本书中的示例,我特意选择了 GitHub 作为源代码控制库,并使用 Travis CI 作为构建引擎。Git 源代码控制库是非常受欢迎的代码库, GitHub 是当今可用的最大的基于云的源代码控制库之一。

Travis CI 是一个与 GitHub 紧密集成的构建引擎(它也支持 Subversion 和 Mercurial)。它非常容易使用,并完全由你的项目的根目录中的单个配置文件( .travis.yml ) 驱动。它的简单和自用的特性使得建立一个简单的构建管道变得非常容易。

到目前为止,本书中的所有代码示例都可以在桌面单独运行(除了连接到 GitHub 外)。在本章中,如果你想完全遵循代码示例,则需要创建你自己的 GitHub , Travis CI 和 Docker hub 账户。我们不会介绍如何创建这些账户,但个人 Travis CI 账户和你的 GitHub 账户的

创建都可以在 Travis CI 网页 ( <http://travis-ci.org> ) 完成。

### 在我们开始之前快速记录

为了本书的目的 ( 和我的理智 ) , 我为书中的每一章设置了一个单独的 GitHub 存储库。本章的所有源代码都可以作为一个独立单元来构建和部署。但是, 在本书之外, 我强烈建议你在自己的环境中使用自己的独立构建过程设置每个微服务。这样, 每个服务可以相互独立地部署。在构建过程中, 我将所有服务作为一个单元部署, 这是因为我想用一个构建脚本将整个环境推送到 Amazon 云, 而不是为每个单独的服务管理构建脚本。

## 10.6. 在 Travis CI 中构建服务

在本书中构建的每个服务的核心是一个 Maven pom.xml 文件, 用于构建 Spring Boot 服务, 将其打包到可执行 JAR 中, 然后构建可用于启动服务的 Docker 镜像。直到本章, 服务的编译和启动都是通过:

- 在本地机器上打开一个命令行窗口。
- 运行本章的 Maven 脚本。这将构建本章的所有服务, 然后将它们打包成一个 Docker 镜像, 并将其推送到本地运行的 Docker 存储库。
- 从本地 Docker 仓库启动新创建的 Docker 镜像, 使用 docker-compose 和 docker-machine 启动本章的所有服务。

问题是, 你如何在 Travis CI 中重复这个过程? 这一切都始于一个叫做 .travis.yml 的文件。 .travis.yml 是一个基于 YAML 的文件, 它描述了当 Travis CI 执行你的构建时你想要采取的操作。这个文件存储在你的微服务的 GitHub 仓库的根目录下。在第 10 章, 这个文件可以在 `spmia-chapter10-code/.travis.yml` 中找到。

当一个提交发生在 Travis CI 正在监视的 GitHub 存储库上时, 它将查找 .travis.yml 文

件，然后启动构建过程。图 10.17 显示了在提交到用于保存本章代码的 GitHub 存储库 (<https://github.com/carnellj/spmia-chapter10>) 时，.travis.yml 文件将执行的步骤。

- 开发人员对第 10 章 GitHub 存储库中的一个微服务进行了更改。
- GitHub 通知 Travis CI 发生了一个提交。当你向 Travis 注册并提供你的 GitHub 账户通知时，此通知配置将无缝地发生。Travis CI 将启动一个虚拟机，用来执行构建。然后，Travis CI 将从 GitHub 中检出源代码，再使用 .travis.yml 文件开始整个构建和部署过程。
- Travis CI 在构建中设置基本配置并安装任何依赖。基本配置包括你将在构建中使用哪种语言 (Java)，是否需要 Sudo 执行软件安装和访问 Docker (用于创建和标记 Docker 容器)，设置在构建中所需的安全环境变量，并定义如何通知构建的成功或失败。
- 在实际构建执行之前，可以指示 Travis CI 安装可能需要的任何第三方库或命令行工具作为构建过程的一部分。你使用两个这样的工具，即 Travis 和 Amazon ecs-cli (EC2 容器服务客户端) 命令行工具。
- 对于构建过程，始终要在源代码库中标记代码，以便在将来的任何时候都可以根据构建的标记提取源代码的完整版本。
- 你的构建过程将执行服务的 Maven 脚本。Maven 脚本将编译你的 Spring 微服务，运行单元和集成测试，然后基于构建构建一个 Docker 镜像。
- 一旦构建的 Docker 镜像完成，构建过程会将镜像推送到 Docker hub，使用与标记源代码存储库相同的标记名称。
- 然后，你的构建过程将使用项目的 docker-compose 文件和 Amazon 的 ecs-cli 将你构建的所有服务部署到 Amazon 的 Docker 服务，Amazon ECS。



- 一旦服务部署完成，你的构建过程将启动一个完全独立的 Travis CI 项目，该项目将针对开发环境运行平台测试。

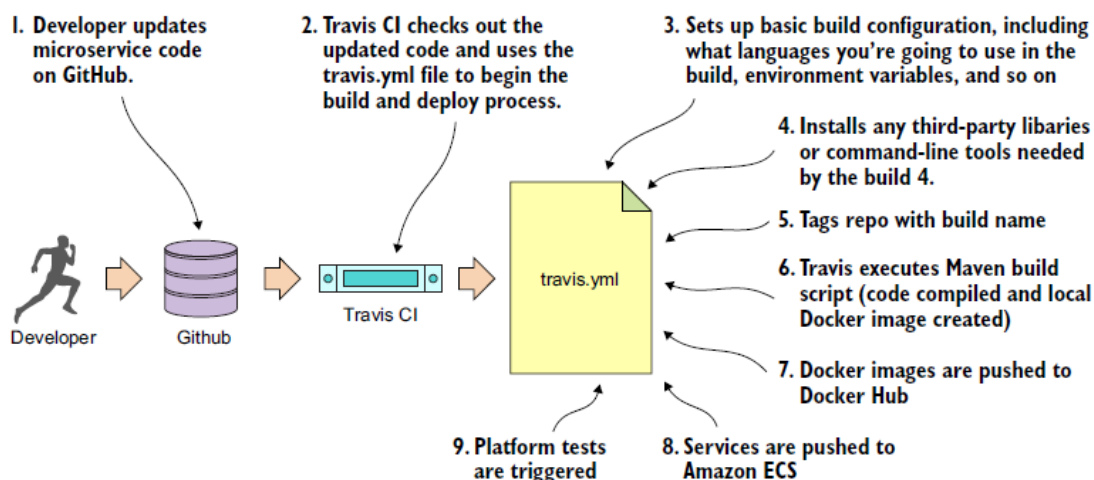


图 10.17 .travis.yml 文件构建和部署软件的具体步骤

#### ① Developer updates microservice code on GitHub.

开发人员更新 GitHub 上的微服务代码。

#### ② Travis CI checks out the updated code and uses the `travis.yml` file to begin the build and deploy process.

Travis CI 检出更新的代码并使用 `travis.yml` 文件开始构建和部署过程。

#### ③ Sets up basic build configuration, including what languages you're going to use in the build, environment variables, and so on

设置基本的构建配置，包括将在构建中使用的语言，环境变量等

#### ④ Installs any third-party libraries or command-line tools needed by the build 4.

安装构建版本 4 所需的任何第三方库或命令行工具。

#### ⑤ Tags repo with build name

使用构建名称标记仓库

#### ⑥ Travis executes Maven build script (code compiled and local Docker image



*created)*

Travis 执行 Maven 构建脚本 ( 编译代码并创建本地 Docker 镜像 )

⑦ *Docker images are pushed to Docker Hub*

Docker 镜像被推送到 Docker Hub

⑧ *Services are pushed to Amazon ECS*

服务被推送到 Amazon ECS

⑨ *Platform tests are triggered*

平台测试被触发






现在我们已经介绍了.travis.yml 文件中涉及的一般步骤,让我们来看看.travis.yml 文件的具体细节。清单 10.1 显示了.travis.yml 文件的不同部分。

**注意：**清单 10.1 中的代码注释与图 10.17 中的数字排列对应。

#### 清单 10.1 .travis.yml 构建的剖析

```
language: java
jdk:
  - oraclejdk8
cache:
  directories:
    - "$HOME/.m2"
sudo: required
services:
  - docker
notifications:
  email:
    - youremail@gmail.com
  on_success: always
  on_failure: always
branches:
  only:
    - master
env:
  global:
```

③ 设置构建的核心运行时配置

```
# Remove for conciseness
before_install:  ④执行必要的命令行工具的预构建安装
- gem install travis -v 1.8.5 --no-rdoc --no-ri
- sudo curl -o /usr/local/bin/ecs-cli
  ➡ https://s3.amazonaws.com/amazon-ecs-cli/
  ➡ ecs-cli-linux-amd64-latest
- sudo chmod +x /usr/local/bin/ecs-cli
- export BUILD_NAME=chapter10-$TRAVIS_BRANCH-
  ➡ $(date -u "+%Y%m%d%H%M%S")-$TRAVIS_BUILD_NUMBER
- export CONTAINER_IP=52.53.169.60
- export PLATFORM_TEST_NAME="chapter10-platform-tests"
script:
- sh travis_scripts/tag_build.sh  ⑤执行一个 shell 脚本，它将
  使用构建名称标记源代码
- sh travis_scripts/build_services.sh  ⑥使用 Maven 构建服务器和本地 Docker 镜像
- sh travis_scripts/deploy_to_docker_hub.sh  ⑦将 Docker 镜像推送到 Docker Hub
- sh travis_scripts/deploy_amazon_ecs.sh  ⑧在 Amazon ECS 容器中启动服务
- sh travis_scripts/trigger_platform_tests.sh
  ⑨触发一个 Travis 构建，执行构建服务的平台测试
```

现在我们将更详细地介绍构建过程中涉及的每个步骤。

### 10.6.1. 核心构建运行时配置



travis.yml 文件的第一部分涉及配置 Travis 构建的核心运行时配置。通常，.travis.yml

文件的这一部分将包含 Travis 特定的功能，这些功能将执行类似的操作：

- 告诉 Travis 你将使用什么编程语言？
- 定义你的构建过程是否需要 Sudo 访问权限
- 定义你是否要在构建过程中使用 Docker
- 声明你将要使用的安全环境变量

下一个清单显示了构建文件的这个特定部分。

#### 清单 10.2 为你的构建配置核心运行时

```
language: java  ①告诉 Travis 使用 Java 和 JDK 8 作为你的主要运行时环境
jdk:
- oraclejdk8
cache:  ②告诉 Travis 在构建之间缓存和重用你的 Maven 目录
directories:
```

```

- "$HOME/.m2"
sudo: required ← ③允许构建在正在运行的虚拟机上使用 Sudo 访问
services:
- docker
notifications: ← ④配置用于通知构建成功或失败的电子邮件地址
email:
- youremail@gmail.com
on_success: always
on_failure: always
branches: ← ⑤指示 Travis 它应该只建立在对主分支的提交上
only:
- master
env: ← ⑥设置要在脚本中使用的安全环境变量
global:
-secure: IAs5WrQIYjH0rpO6W37wbLAixjMB7kr7DBAeWhjeZFwOkUMJbfuHNC=z...
#d Remove for conciseness

```

Travis 构建脚本的第一件事就是告诉 Travis 将使用哪种主要语言来执行构建。通过将 `java` 和 `jdk` 属性指定为 `java` 和 `oraclejdk8`，① Travis 将确保为你的项目安装和配置 JDK。

`.travis.yml` 文件的下一部分 `cache.directories` 属性② 告诉 Travis 在执行构建时缓存此目录的结果，并在多个构建中重复使用它。在处理像 Maven 这样的包管理器时，这非常有用，每次构建启动时，都需要花费大量时间来下载 jar 依赖的新副本。如果没有设置 `cache.directories` 属性，则本章的构建可能需要 10 分钟的时间才能下载所有相关的 jar。

清单 10.2 中的下两个属性是 `sudo` 属性和 `service` 属性。③ `sudo` 属性用来告诉 Travis 你的构建过程需要使用 `sudo` 作为构建的一部分。UNIX `sudo` 命令用于临时提升用户到 root 权限。一般来说，当你需要安装第三方工具时，你可以使用 `sudo`。当你需要安装 Amazon ECS 工具时，你将在稍后完成此操作。

`services` 属性用于告诉 Travis 在执行构建时是否要使用某些关键服务。例如，如果你的集成测试需要本地数据库供其运行，则 Travis 允许你在构建中直接启动 MySQL 或 PostgreSQL 数据库。在这种情况下，你需要运行 Docker 为每个 EagleEye 服务构建 Docker 镜像，并将镜像推送到 Docker hub。你已经将服务属性设置为在构建启动时启动 Docker。

下一个属性，`notifications` ④ 定义了构建成功或失败时使用的通信通道。现在，你始终通过将构建的通知渠道设置为电子邮件来传达构建结果。Travis 会通过电子邮件通知你构建的成功与失败。Travis CI 可以通过多种渠道（包括 Slack，IRC，HipChat 或自定义 Web 钩子）通知电子邮件。

`branches.only` ⑤ 属性告诉 Travis Travis 应该建立什么样的分支。对于本章中的示例，你只需执行 Git 主分支的构建。这样可以防止每次在 GitHub 中标记仓库或提交到分支时启动构建。这很重要，因为每次你标记仓库或创建发布时，GitHub 都会回调 Travis。`branch.only` 属性被设置为 `master` 来防止 Travis 进入无尽的构建。

构建配置的最后一部分是敏感环境变量的设置。⑥ 在构建过程中，你可能会与第三方供应商（如 Docker，GitHub 和 Amazon）进行通信。有时候你通过命令行工具进行通信，而其他的时候则是使用 API。无论如何，你经常需要提交敏感的凭证。Travis CI 使你能够添加加密的环境变量来保护这些凭证。

要添加一个加密的环境变量，必须使用桌面上的 `travis` 命令行工具对你的源代码所在的项目目录中的环境变量进行加密。要在本地安装 Travis 命令行工具，请在 <https://github.com/travis-ci/travis.rb> 查看该工具的文档。对于本章中使用的 `.travis.yml`，我创建并加密了以下环境变量：

- `DOCKER_USERNAME`：Docker hub 用户名。
- `DOCKER_PASSWORD`：Docker hub 密码。
- `AWS_ACCESS_KEY`：Amazon ecs-cli 命令行客户端使用的 AWS 访问密钥。
- `AWS_SECRET_KEY`：Amazon ecs-cli 命令行客户端使用的 AWS 密钥。
- `GITHUB_TOKEN`：GitHub 生成的令牌，用于表示允许调用应用程序针对服务器执行的访问级别。这个令牌必须先用 GitHub 应用程序生成。

一旦安装了 Travis 工具，以下命令会将加密的环境变量 DOCKER\_USERNAME 添加到你的.travis.yml 文件的 env.global 部分：

```
travis encrypt DOCKER_USERNAME=somerandomname --add env.global
```

运行此命令后，现在应在.travis.yml 文件的 env.global 部分中看到一个安全属性标记，后跟一长串文本。图 10.18 显示了一个加密的环境变量的例子。

The Travis encryption tools don't put the name of the encrypted environment variable in the file.

```
env:
  global:
    - secure: IAs5WrQIYjH0rp06W37wbLAixjMB7kr7DBAeWhjeZFwOk
    - secure: HRSq780tWtfkKXZSq10ue/wV07TZIU+0mYPn1DctCnovs
    - secure: m4IkvLGXq6LBzSEHJbabS/0cfCD1IRcMjfgp8BaN+wFY+
```

Each encrypted environment variable will have a secure attribute tag.

图 10.18 加密的 Travis 环境变量直接放在.travis.yml 文件中。

① *The Travis encryption tools don't put the name of the encrypted environment variable in the file.*

Travis 加密工具不会将加密的环境变量的名称放在文件中。

② *Each encrypted environment variable will have a secure attribute tag.*

每个加密的环境变量都有一个安全属性标记。

不幸的是，Travis 没有在你的.travis.yml 文件中标记你的加密环境变量的名字。

**注意：**加密变量只适用于加密的，且 Travis 正在构建的单个 GitHub 存储库。你无法剪切和粘贴跨多个.travis.yml 文件的加密环境变量。你的构建将无法运行，因为加密的环境变量不能正确解密。

**不管构建工具如何，总是要加密你的凭证**

尽管我们所有的例子都使用 Travis CI 作为构建工具，但所有现代构建引擎都允许你加密凭证和令牌。请，请确保你加密您的凭据。嵌入在源代码存储库中的凭证是一个常见的安全漏洞。不要相信这个信念，你的源代码控制库是安全的，因此其中的凭证是安全的。

**10.6.2. 安装预构建工具**

哇，预构建配置是庞大的，但下一个部分是小的。构建引擎通常是大量“粘合代码”脚本的来源，以将构建过程中使用的不同工具绑定在一起。为使用你的 Travis 脚本，你需要安装两个命令行工具：

- *Travis*：这个命令行工具用于与 Travis 构建进行交互。稍后在本章中使用它来检索一个 GitHub 标记，以编程方式触发另一个 Travis 构建。
- *ecs-cli*：这是与 Amazon Elastic Container 服务交互的命令行工具。

.travis.yml 文件的 before\_install 部分中列出的每项是一个 UNIX 命令，将在构建启动之前执行。以下清单显示了 before\_install 属性以及需要运行的命令。

**清单 10.3 预构建安装步骤**

before\_install:

```
- gem install travis -v 1.8.5 --no-rdoc --no-ri
- sudo curl -o /usr/local/bin/ecs-cli
  ➡ https://s3.amazonaws.com/amazon-ecs-cli/
  ➡ ecs-cli-linux-amd64-latest
- sudo chmod +x /usr/local/bin/ecs-cli
- export BUILD_NAME=chapter10-$TRAVIS_BRANCH-
  ➡ $(date -u "+%Y%m%d%H%M%S")-$TRAVIS_BUILD_NUMBER
- export CONTAINER_IP=52.53.169.60
- export PLATFORM_TEST_NAME="chapter10-platform-tests"
```

①安装 Travis 命令行工具

②安装命令行工具 Amazon ECS 客户端

③将 Amazon ECS 客户端上的权限更改为可执行

④设置你的流程中使用的环境变量

在构建过程中要做的第一件事是在远程构建服务器上安装 travis 命令行工具：

```
gem install travis -v 1.8.5 --no-rdoc --no-ri
```

在构建过程中，你将通过 Travis REST API 启动另一个 Travis 作业。你需要使用 travis 命令

行工具来获取调用此 REST 调用的令牌。

安装 Travis 工具之后，你将安装 Amazon ecs-cli 工具。这是用于部署、启动和停止在 Amazon 中运行的 Docker 容器的命令行工具。首先下载二进制文件，然后将下载的二进制文件的权限更改为可执行文件来安装 ecs-cli：

```
- sudo curl -o /usr/local/bin/ecs-cli https://s3.amazonaws.com/amazon-ecscli/ecs-cli-linux-amd64-latest
- sudo chmod +x /usr/local/bin/ecs-cli
```

你在.travis.yml的 before\_install 部分中执行的最后一件事是在你的构建中设置了三个环境变量。这三个环境变量将有助于驱动构建的行为。这些环境变量是

- BUILD\_NAME
- CONTAINER\_IP
- PLATFORM\_TEST\_NAME

在这些环境变量中设置的实际值是

```
- export BUILD_NAME=chapter10-$TRAVIS_BRANCH-
  ➡ $(date -u "+%Y%m%d%H%M%S")-$TRAVIS_BUILD_NUMBER
- export CONTAINER_IP=52.53.169.60
- export PLATFORM_TEST_NAME="chapter10-platform-tests"
```

第一个环境变量 BUILD\_NAME 生成一个唯一的构建名称，其中包含构建的名称，后面是日期和时间(直到秒字段)，然后是 Travis 中的构建编号。当它被推送到 Docker hub 仓库时，这个 BUILD\_NAME 将被用来在 GitHub 和你的 Docker 镜像中标记你的源代码。

第二个环境变量 CONTAINER\_IP 包含你的 Docker 容器将运行的 Amazon ECS 虚拟机的 IP 地址。这 CONTAINER\_IP 稍后将传递给另一个将执行你的平台测试的 Travis CI 作业。

**注意：**我不会把静态 IP 地址分配给转换的 Amazon ECS 服务器。如果我完全卸载容器，我会得到一个新的 IP。在真正的生产环境中，ECS 集群中的服务器可能会分配静态(不变)IP，集群将具有 Amazon 企业负载均衡器 (ELB) 和 Amazon Route 53 DNS 名称，以便实际

ECS 服务器的 IP 地址对于服务是透明的。但是，设置这么多的基础设施已经超出了本章试图演示的范围。

第三个环境变量 `PLATFORM_TEST_NAME` 包含正在执行的构建作业的名称。我们将在本章的后面探讨它的用法。

### 关于审计和可追溯性

许多金融服务和医疗保健公司的一个共同要求是，他们必须证明所部署的生产软件的可追溯性，一直回溯到所有较低的环境，回到构建软件的构建工作，然后返回到代码被检入到源代码库中。不变的服务器模式真的能够帮助企业满足这一要求。正如你在构建示例中看到的那样，你将使用相同的构建名称标记将要部署的源代码管理存储库和容器镜像。该构建名是唯一的，并绑定到 Travis 构建号中。由于你只是通过每个环境升级容器镜像，并且每个容器镜像都使用构建名称进行标记，因此你已将该容器镜像的可追溯性建立回到与其关联的源代码。因为这些容器一旦被标记就永远不会被更改，所以你具有强大的审计地位，以显示部署的代码与底层的源代码存储库相匹配。现在，如果你想要额外安全地使用它，那么在你为项目源代码添加标签时，你还可以使用与构建生成相同的标签来标记驻留在 Spring Cloud Config 存储库中的应用程序配置。

### 10.6.3. 执行构建

此时，所有预构建配置和依赖项安装完成。要执行你的构建，你要使用 Travis 脚本属性。像 `before_install` 属性一样，脚本属性也会执行一系列将被执行的命令。由于这些命令冗长，我选择将构建中的每个主要步骤封装到其自己的 shell 脚本中，并让 Travis 执行 shell 脚本。以下清单显示了构建中将要进行的主要步骤。

#### 清单 10.4 执行构建

script:

```
- sh travis_scripts/tag_build.sh
```



```
- sh travis_scripts/build_services.sh
- sh travis_scripts/deploy_to_docker_hub.sh
- sh travis_scripts/deploy_amazon_ecs.sh
- sh travis_scripts/trigger_platform_tests.sh
```

让我们来看看在脚本步骤中执行的每个主要步骤。

#### 10.6.4. 标记源代码控制代码

travis\_scripts/tag\_build.sh 脚本负责使用构建名称标记代码库中的代码。对于这里的例子，我正在通过 GitHub REST API 创建一个 GitHub 版本。一个 GitHub 版本不仅会标记源代码控制库，而且还会允许你将版本注释等内容发布到 GitHub 网页上，以及源代码是否代码的预发布版本。

由于 GitHub 发行版 API 是一个基于 REST 的调用，因此你将在 shell 脚本中使用 curl 来执行实际的调用。以下清单显示了 travis\_scripts/tag\_build.sh 脚本中的代码。

##### 清单 10.5 使用 GitHub 发行版 API 为第 10 章代码存储库添加标签

```
echo "Tagging build with $BUILD_NAME"
export TARGET_URL="https://api.github.com/ ← ①GitHub 发布版本 API 的目标端点
    ➤ repos/carnellj/spmia-chapter10/
    ➤ releases?access_token=$GITHUB_TOKEN"

body="{ ← ②REST 调用的主体
    \"tag_name\": \"$BUILD_NAME\",
    \"target_commitish\": \"master\",
    \"name\": \"$BUILD_NAME\",
    \"body\": \"Release of version $BUILD_NAME\",
    \"draft\": true,
    \"prerelease\": true
}"

curl -k -X POST \ ← ③使用 curl 来调用用于启动构建的服务
    -H "Content-Type: application/json" \
    -d "$body" \
    $TARGET_URL
```

这个脚本很简单。你要做的第一件事就是构建 GitHub 发布版本 API 的目标 URL：

```
export TARGET_URL="https://api.github.com/repos/
    ➤ carnellj/spmia-chapter10/
```

```
➡ releases?access_token=$GITHUB_TOKEN"
```

在 TARGET\_URL 中，你传递了一个名为 access\_token 的 HTTP 查询参数。该参数包含一个 GitHub 个人访问令牌，它被设置为明确允许你的脚本通过 REST API 采取行动。你的 GitHub 个人访问令牌存储在名为 GITHUB\_TOKEN 的加密环境变量中。要生成个人访问令牌，请登录到你的 GitHub 账户并导航至 <https://github.com/settings/tokens>。当你生成令牌时，请确保你将其剪切并立即粘贴。当你离开 GitHub 屏幕时，它将会消失，你需要重新生成它。

脚本中的第二步是为 REST 调用设置 JSON 正文：

```
body="{
  \"tag_name\": \"${BUILD_NAME}\",
  \"target_commitish\": \"master\",
  \"name\": \"${BUILD_NAME}\",
  \"body\": \"Release of version ${BUILD_NAME}\",
  \"draft\": true,
  \"prerelease\": true
}"
```

在前面的代码片段中，你提供 \$BUILD\_NAME 作为 tag\_name 值，并使用 body 字段设置基本版本说明。

一旦建立了调用的 JSON 主体，通过 curl 命令执行调用很简单：

```
curl -k -X POST \
  -H "Content-Type: application/json" \
  -d "$body" \
  $TARGET_URL
```

### 10.6.5. 构建微服务和创建 Docker 镜像

Travis 脚本属性的下一步是构建各个服务，然后为每个服务创建 Docker 容器镜像。你可以通过一个名为 travis\_scripts/build\_services.sh 的小脚本来完成。该脚本将执行以下命令：

```
mvn clean package docker:build
```

这个 Maven 命令执行第 10 章代码仓库中所有服务的父 Maven `spmia-chapter10-code/pom.xml` 文件。父 `pom.xml` 为每个服务执行单独的 Maven `pom.xml`。每个单独的服务都会构建服务源代码，执行任何单元和集成测试，然后将服务打包到可执行的 jar 文件中。

在 Maven 构建中发生的最后一件事是创建一个 Docker 容器镜像 将其推送到在 Travis 构建机器上运行的本地 Docker 存储库。Docker 镜像的创建是使用 Spotify Docker 插件 (<https://github.com/spotify/docker-maven-plugin>) 来完成的。如果你对构建过程中 Spotify Docker 插件的工作方式感兴趣，请参阅附录 A “设置你的桌面环境”。这里解释了 Maven 构建过程和 Docker 配置。

#### 10.6.6. 将镜像发布到 Docker Hub

在构建的这一点上，服务已经被编译和打包，并且在 Travis 构建机器上创建了一个 Docker 容器镜像。你现在要通过 `travis_scripts/deploy_to_docker_hub.sh` 脚本将 Docker 容器镜像推送到中央 Docker 存储库。Docker 存储库就像你创建的 Docker 镜像的 Maven 存储库。Docker 映像可以被标记并上传到其中，其他项目可以下载和使用这些镜像。

对于这个代码示例，你将使用 Docker hub (<https://hub.docker.com/>)。以下清单显示了 `travis_scripts/deploy_to_docker_hub.sh` 脚本中使用的命令。

##### 清单 10.6 将创建的 Docker 镜像推送到 Docker Hub

```
echo "Pushing service docker images to docker hub ...."
docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
docker push johncarnell/tmx-authentication-service:$BUILD_NAME
docker push johncarnell/tmx-licensing-service:$BUILD_NAME
docker push johncarnell/tmx-organization-service:$BUILD_NAME
docker push johncarnell/tmx-confsvr:$BUILD_NAME
docker push johncarnell/tmx-eurekasvr:$BUILD_NAME
docker push johncarnell/tmx-zuulsvr:$BUILD_NAME
```

这个 shell 脚本的流程很简单。你要做的第一件事就是使用 Docker 命令行工具和 Docker

Hub 账户的用户凭据登录到 Docker Hub，镜像将被推送到 Docker Hub。请记住，你的 Docker Hub 凭证存储为加密的环境变量：

```
docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
```

脚本登录后，代码会将驻留在 Travis 构建服务器上运行的本地 Docker 存储库中的各个微服务的 Docker 镜像推送到 Docker Hub 存储库：

```
docker push johncarnell/tmx-confsvr:$BUILD_NAME
```

在前面的命令中，你告诉 Docker 命令行工具使用 johncarnell 账户将镜像推送到 Docker hub（这是 Docker 命令行工具使用的默认中心）。正在推送的镜像将是 tmx-confsvr 镜像，其标记名称是 \$BUILD\_NAME 环境变量的值。

#### 10.6.7. 在 Amazon ECS 环境中启动服务

此刻，所有的代码已经构建和标记，并创建了一个 Docker 镜像。你现在已准备好将你的服务部署到你在 10.1.3 节中创建的 Amazon ECS 容器。执行此部署的操作可在 travis\_scripts/deploy\_to\_amazon\_ecs.sh 中找到。下面的清单显示了这个脚本的代码。

##### 清单 10.7 将 Docker 映像部署到 EC2

```
echo "Launching $BUILD_NAME IN AMAZON ECS"
ecs-cli configure --region us-west-1 \
    --access-key $AWS_ACCESS_KEY
    --secret-key $AWS_SECRET_KEY
    --cluster spmia-tmx-dev
ecs-cli compose --file docker/common/docker-compose.yml up
rm -rf ~/.ecs
```

**注意：**在 Amazon 控制台中，Amazon 只显示区域所在的州/城市/国家的名称，而不是实际的区域名称（us-west-1，useast-1 等）。例如，如果你要查看 Amazon 控制台并想要查看北加利福尼亚地区，则不会显示地区名称是 us-west-1。有关所有 Amazon 区域（以及每个服务的端点）的列表，请参阅 <http://docs.aws.amazon.com/general/latest/gr/rande.html>。

由于每个构建都由 Travis 启动新的构建虚拟机,因此需要使用 AWS 访问密钥和秘密密钥来配置构建环境的 ecs-cli 客户端。完成后,你可以使用 `ecs-cli compose` 命令和 `docker-compose.yml` 文件启动到 ECS 集群的部署。你的 `docker-compose.yml` 文件被参数化来使用构建名称(包含在环境变量 `$BUILD_NAME` 中)。

### 10.6.8.开展平台测试

你的构建过程还有最后一步:开展平台测试。每次部署到新环境后,你都会开展一系列的平台测试,以确保你的所有服务都能正常运行。平台测试的目标是调用部署版本中的微服务,并确保服务正常运行。

我将平台测试作业从主版本中分离出来,以便可以独立于主版本进行调用。为此,我使用 Travis CI REST API 以编程方式调用平台测试。`travis_scripts/trigger_platform_tests.sh` 脚本完成这项工作。下面的清单显示了这个脚本的代码。

#### 清单 10.8 使用 Travis CI REST API 开展平台测试

```
echo "Beginning platform tests for build $BUILD_NAME"
travis login --org --no-interactive \
    --github-token $GITHUB_TOKEN
export RESULTS=`travis token --org`
export TARGET_URL="https://api.travis-ci.org/repo/
    carnellj%2F$PLATFORM_TEST_NAME/requests"
echo "Kicking off job using target url: $TARGET_URL"

body="{
  \"request\": {
    \"message\": \"Initiating platform tests for build $BUILD_NAME\",
    \"branch\": \"master\",
    \"config\": {
      \"env\": {
        \"global\": [\"BUILD_NAME=$BUILD_NAME\",
                    \"CONTAINER_IP=$CONTAINER_IP\"]
      }
    }
  }
}"
```

①使用你的 GitHub 令牌登录 Travis CI。将返回的令牌存储在 RESULTS 变量中。

②构建调用的 JSON 正文,将两个值传递给下游作业。

```
curl -s -X POST \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -H "Travis-API-Version: 3" \
  -H "Authorization: token $RESULTS" \
  -d "$body" \
  $TARGET_URL
```

③使用 Curl 来调用 Travis CI REST API

清单 10.8 中的第一件事是使用 Travis CI 命令行工具登录到 Travis CI 并获取一个 OAuth2 令牌，你可以使用它来调用其他 Travis REST API。你将此 OAUTH2 令牌存储在 \$RESULTS 环境变量中。

接下来，为 REST API 调用构建 JSON 正文。你的下游 Travis CI 工作启动了一系列测试你的 API 的 Python 脚本。这个下游作业需要设置两个环境变量。在清单 10.8 中构建的 JSON 正文中，传递了两个环境变量 \$BUILD\_NAME 和 \$CONTAINER\_IP，这些变量将传递给你的测试作业：

```
\ "env\": {
  \ "global\": [ \ "BUILD_NAME=$BUILD_NAME\",
                \ "CONTAINER_IP=$CONTAINER_IP\" ]
}
```

脚本中的最后一个操作是调用运行平台测试脚本的 Travis CI 构建作业。这是通过使用 curl 命令来调用你的测试作业的 Travis CI REST 端点来完成的：

```
curl -s -X POST \
  -H "Content-Type: application/json" \
  -H "Accept: application/json" \
  -H "Travis-API-Version: 3" \
  -H "Authorization: token $RESULTS" \
  -d "$body" \
  $TARGET_URL
```

平台测试脚本存储在一个名为 chapter10-platform-tests ( <https://github.com/carnellj/chapter10-platform-tests> ) 的单独的 GitHub 仓库中。这个存储库有三个测试 Spring Cloud Config 服务器，Eureka 服务器和 Zuul 服务器的 Python 脚本。Zuul 服务器平台测试还测试许可服务和组织服务。这些测试并不全面，因为它们对服务的每一个方面

都有所行动，但是它们确实运用了足够多的服务来确保它们的功能。

**注意：**我们不打算介绍平台测试。测试很简单，介绍测试不会为本章增加大量的价值。

## 10.7. 关于构建/部署管道的总结

当本章（和本书）结束时，我希望你对构建构建/部署管道所需的工作量表示赞赏。功能良好的构建和部署管道对于部署服务至关重要。你的微服务架构的成功与否不仅取决于服务中涉及的代码：

- 了解这个构建/部署管道中的代码是为了本书的目的而被简化了。一个好的构建/部署管道将更加通用化。它将得到 DevOps 团队的支持，并分解成一系列独立的步骤（编译>打包>部署>测试），开发团队可以使用这些步骤来“钩”他们的微服务构建脚本。
- 本章中使用的虚拟机镜像制作过程非常简单，每个微服务都是使用 Docker 文件构建的，以定义将要安装在 Docker 容器上的软件。许多厂商将使用像 Ansible(<https://github.com/ansible/ansible>)，Puppet(<https://github.com/puppetlabs/puppet>)或 Chef(<https://github.com/chef/chef>)这样的配置工具来安装和配置操作系统到正在构建的虚拟机或容器镜像上。
- 你的应用程序的云部署拓扑已经整合到单台服务器上。在真正的构建/部署管道中，每个微服务都将有自己的构建脚本，并将彼此独立部署到集群 ECS 容器。

## 10.8. 小结

- 构建和部署管道是交付微服务的关键部分。功能良好的构建和部署管道应该允许在几分钟内部署新功能和 bug 修复程序。



- 构建和部署管道应该是自动化的，不需要直接的人员干预来提供服务。该流程的任何手动部分都代表了可变性和失败的机会。
- 构建和部署管道自动化确实需要大量的脚本和配置才能正确使用。构建所需的工作量不容小觑。
- 构建和部署管道应交付不可变的虚拟机或容器镜像。一旦创建了服务器镜像，就不应该修改它。
- 特定环境的服务器配置应在服务器设置时作为参数传入。