

10 使用Redis进行缓存和在Ubuntu (DigitalOcean) 和Netlify上部署 在本章中，我们将探索另一种部署设置—一个经过试验和测试用于Django和其他WSGI以及ASGI Web应用程序的强大的Uvicorn/Gunicorn/Nginx解决方案。这应该为您在开始下一个FARM堆栈项目时提供足够的选择。我们还将添加一个简单的Redis缓存解决方案，减轻MongoDB的一些请求，这些请求可以（而且应该！）直接进行缓存并提供。最后，我们将在Netlify上部署基于React的前端，这是另一个非常流行的部署选项，其简单性与其灵活性相匹配。 在本章中，我们将涵盖以下主题： 在DigitalOcean上创建帐户（可选）

准备我们的Ubuntu服务器使用Nginx

通过Uvicorn、Gunicorn和Nginx部署FastAPI实例 使用Redis进行缓存

在Netlify上创建一个免费帐户 在Netlify上部署React前端 到本章结束时，您应该在不同的服务平台上部署基于FARM堆栈的应用程序时感到自信，包括裸机的Ubuntu（或任何Linux）服务器。您将能够识别何时何地添加缓存，并轻松地使用Redis实现它。最后，有了可能的部署解决方案的知识，当部署应用程序的时候，您将能够做出坚实的决策。

Redis缓存和在Ubuntu (DigitalOcean) 和Netlify上部署 280

将FastAPI部署到DigitalOcean (或任何Linux服务器!) 在本节中, 我们将把简单的分析应用程序部署在DigitalOcean (www.digitalocean.com) 上的Ubuntu服务器上, 作为异步服务器网关接口 (ASGI) 应用程序。我们最终将获得一个相当强大和可定制的设置, 其中包括我们的开发Web服务器-Uvicorn, 以及Gunicorn (<https://gunicorn.org>), 一个与Nginx非常友好且稳健的Web服务器, 并运行Ubuntu的虚拟机-数字海洋水滴。虽然在本例中, 我们将使用DigitalOcean, 但该过程应该适用于任何基于Debian或Ubuntu的设置; 您可以在运行Ubuntu的自己的机器上尝试它。以下说明在Ubuntu上设置DigitalOcean服务器的优秀教程中大力借鉴了Brian Boucheron (<https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-20-04>) 和Mason Egger和Erin Glass部署Async Django应用程序的教程 (<https://www.digitalocean.com/community/tutorials/how-to-set-up-an-asgi-django-app-with-postgres-nginx-and-uvicorn-on-ubuntu-20-04>)。你应该阅读它们, 因为它们非常有用且写得很好!

重要提醒 在本节中, 我们将大量使用SSH-安全外壳协议。

SSH是为在不安全的网络上访问安全网络服务而开发的加密协议。

如果这不太合理, 请不要担心-有很多关于基本SSH操作的优秀资源在互联网上。如果您愿意深入了解DevOps, 您可以阅读以下内容: <https://www.amazon.com/Mastering-Ubuntu-Server-configuring-troubleshooting/dp/1800564643>。掌握Ubuntu服务器是一个优秀的指南。在以下页面中, 我们将只是登录到DigitalOcean

droplet, 这不过是一个我们将能够控制的远程Ubuntu计算机。

虽然我将展示在DigitalOcean droplet上部署完全功能的FastAPI实例的过程, 但尝试此过程的最佳方法是在基于Ubuntu的服务器上练习。

如果您有备用的框 (甚至是较旧的框), 请安装Ubuntu并尝试从主计算机连接到它。

部署过程将分为简单的步骤。

DigitalOcean是提供云计算和基础架构即服务 (IaaS) 的领导者之一。

用户可以从不同类型的虚拟机中受益, 这些虚拟机可以根据我们的需要进行建模。在我们的情况下, 我们只是需要一种解决方案来托管我们的FastAPI服务器, 类似于在前几章中使用Heroku的方法。

虽然DigitalOcean没有提供完全免费的层, 但起步相当便宜 (约4美元/月)。它具有灵活和可扩展的系统, 可以根据您的需求轻松缩放, 它提供了对虚拟机-滴答声的完全控制, 这也为我们带来了一个全新的灵活性水平, 这是我们在本书中经常使用的一个词。另一个优点

在DigitalOcean上部署FastAPI（或任何Linux服务器！） DigitalOcean最出色的地方之一是它的优秀社区和无尽的精心编写的文章，涵盖任何您想要实现的服务或设置，因此如果您进入了部署、数据库设置等领域，则DigitalOcean是一个很好的起点。请注意，DigitalOcean及其竞争对手（例如Linode）完全能够托管我们的完整全栈设置-我们也可以在服务器上安装MongoDB，添加Node.js和Next或React前端，并通过Nginx对所有内容进行编排，一个强大而快速的服务器。然而，在此示例中，我们仅想提供FastAPI实例并展示不同类型的部署。按照以下步骤：

1. 创建DigitalOcean帐户！转到DigitalOcean注册页面 <https://cloud.digitalocean.com/registrations/new> 并填写您的数据。

如果您愿意，可以使用GitHub或Google进行注册，如果您有推荐代码，也可以使用推荐代码，以便您可以在规定的时间内尝试该服务。一旦您提交了数据（并且一旦您有一些信用可以支配-无论是来自推荐计划还是在连接信用卡之后），您将能够创建您的第一个虚拟机。

2. 创建Droplet。我使用的是Ubuntu 22.04 x64

Ubuntu发行版，计划是Basic（最便宜的），CPU选项是\$4 /月，512 MB / 1 CPU（您必须单击左箭头才能找到此计划！）。由于我在欧洲，我选择了法兰克福数据中心区域。最后，为了简化事情，我选择了密码验证，因此我输入了一个根密码（我不会在这里披露！）。我给主机名命名-farmstack。尽管我们将使用IP地址通过SSH访问此全新的机器，但拥有一个用户友好的机器名称很有用。请给DigitalOcean一些时间来准备您的Droplet。大约30秒后，您将能够单击左侧菜单下的Droplets，并进入显示有关Droplet的信息的页面。现在，您拥有了基于Ubuntu的服务器，可以控制它！

3. 为了验证您确实能够作为根登录到您的全新机器上，请单击Droplet的IP地址以将其复制，然后在Windows上打开Cmder（或您一直在使用的任何shell）或如果您是在Linux或macOS上，则打开bash / shell，然后尝试访问Droplet：`ssh root@`

4. Cmder会友善地告诉您无法确认主机的真实性，这在此阶段是正常的，并问您是否要继续连接。键入是；您将被问候以下Shell：`root@farmstack:~#`

5. 为了遵循良好的实践，创建一个具有所有必要特权的新用户帐户，以便我们不使用根帐户进行Web托管。让我们创建一个名为farmuser的帐户：`adduser farmuser`

使用Redis缓存并在Ubuntu (DigitalOcean) 和Netlify上部署 6. 要求提供密码 (两次)、名称和其他信息, 例如房间号码(!)。记住该密码非常重要! 这个新创建的用户需要具备各种管理任务的能力, 因此我们应该授予他们足够的特权。在同一SSH会话中, 输入以下内容: `usermod -aG sudo farmuser` 之后, 当我们以farmuser身份登录时, 只需在执行需要超级用户权限的操作前键入sudo即可。我们将利用UFW防火墙, 以确保只有特定类型的连接被允许连接到我们的服务器。当涉及到DigitalOcean的防火墙时, 有不同的选项, 但这应该足够易于在不同的机器上设置。尽管如此, 事情可能会变得有些棘手 - 我们需要确保当我们离开我们的SSH根shell时, 我们将能够使用我们的farmuser帐户重新登录! 7. 为确保OpenSSH被允许访问该机器, 输入以下命令: `ufw allow OpenSSH` 8.

您应该会看到一条消息, 表示规则已更新。现在, 让我们启用ufw并检查其状态: `ufw enable ufw status` 前述命令应该会警告您可能会干扰现有的SSH连接; 无论如何, 确认第一个命令即可。第二个命令应该只会通知您该服务正在运行。 9. 太好了。现在, 保持SSH会话保持活动状态, 并打开一个新的终端, 以便我们可以测试我们的常规但高度特权的farmuser的连接: `ssh farmuser@`

您应该看到一个提示符, 即farmuser@farmstack:~\$。这很好 -

现在, 我们可以继续使用此 (常规) 用户, 并在需要进行复杂操作时使用sudo! 现在是时候更新我们的Ubuntu软件包并添加一些新的软件包了。以farmuser (或其他常规的非root用户名) 身份登录, 执行以下命令: `sudo apt update sudo apt install python3-venv nginx curl sudo`将提示您输入密码 - 您的常规farmuser密码 - 请提供。除了Python 3之外, 我们还安装了Nginx (我们强大的Web服务器和反向代理解决方案) 和curl (用于在本地测试我们的API服务)。

部署FastAPI到DigitalOcean（或任何Linux服务器！）现在，我们进入第二个与项目有关的阶段，即部署阶段。现在是时候创建虚拟环境了，就像我们在开发阶段做的那样。这是一个基础的服务器，所以我们必须手动完成所有步骤。没有像Heroku或Vercel那样提供有帮助的指导。请按照以下步骤操作：

1. 让我们在我们的主文件夹中创建一个名为apiserver的目录并进入它（您始终可以通过PWD查看当前位置！）：`mkdir /apiserver` `cd /apiserver`
2. 现在，让我们创建一个Python 3环境：`python3 -m venv venv`
3. 安装完成后，使用以下命令激活此环境：`source venv / bin / activate` 您应该在命令提示符前看到venv。
4. 现在是时候获取您为后端创建的GitHub仓库的地址并将目录更改为/apiserver。接下来，克隆GitHub仓库内的所有代码：`git clone <您的repo地址>` 这将创建与存储库名称相同的文件夹-在我的情况下，它有点繁琐：FARM-chapter9-backend。从存储库克隆代码不会同时复制具有MongoDB和Sendgrid（以前的应用程序中的Cloudinary）所需密钥的.env文件。
5. 尽管我们可以通过shell手动设置环境变量，但我们将使用安全复制scp命令来进行明显的复制。确保您位于本地计算机的/ backend文件夹中，并注意远程文件夹。然后，发出以下命令：`scp .env farmuser@207.154.254.114: /apiserver / FARM- chapter9-backend`
6. 现在尝试ls命令，以确保代码文件夹确实存在，但请记住，.env文件不会显示！您必须使用诸如nano .env之类的东西来验证文件确实存在并且包含所需信息。如果您不想用scp搞砸，可以使用nano（由Linux系统提供的强大的命令行文本编辑器）创建并输入.env文件。
7. 一旦代码在Ubuntu droplet中，就进入目录并使用以下命令安装所有依赖项：`pip install -r requirements.txt`

使用Redis缓存和在Ubuntu (DigitalOcean) 和Netlify上部署 重要提示 在提交后端代码之后，您应该更新requirements.txt文件，方法是在本地计算机上已激活的虚拟环境中键入`pip freeze > requirements.txt`。然后将此文件提交到GitHub -

它将是在其他计算机上重建相同虚拟环境的必备材料，包括我们的droplet! 8.

安装完依赖项后，我们可以使用标准的Uvicorn命令测试我们的应用程序：`uvicorn`

`main:app --reload` 提示应该告诉您Uvicorn正在运行`http://`

`127.0.0.1: 8000`，但我们尚无法从外部访问它。 9. 使用`Ctrl + C`停止服务器。要能够

测试API是否起作用，我们必须禁用UFW防火墙。为此，您必须`sudo`通过它：`sudo ufw disable` 注意 这是一种危险的做法，有点像把前门开着。现在，如果您尝试重新运行Uvicorn服务器，您应该能够使用REST客户端或浏览器在droplet的IP地址上，端口为8000，访问您的API! 到目前为止，我们一直在DigitalOcean上尝试我们在本书中所做的事情。现在，是时候介绍Gunicorn了。 重要提示

Gunicorn是一款成熟且经过战斗考验的Unix WSGI Python服务器。它通常与Uvicorn一起使用，因为它高度可配置并能够高效处理Uvicorn工作程序。

Uvicorn文档本身建议包含Gunicorn和Nginx的设置，这正是我们要做的! Gunicorn本身是一个有趣且强大的项目，它的文档是有用的阅读 (<https://gunicorn.org/>)。

现在让我们开始构建我们的部署。按照以下步骤操作： 1.

使用pip简单调用安装gunicorn：`pip install gunicorn`

在DigitalOcean（或任何Linux服务器上）部署FastAPI！ 在安装gunicorn之后，我们可以使用以下命令启动API服务器（保持在源代码目录中！）：`gunicorn --bind 0.0.0.0:8000 main:app -w 4 -k uvicorn.workers.UvicornWorker` 上述命令启动带有四个uvicorn工作进程的gunicorn服务器。Gunicorn还为我们的Uvicorn服务器提供负载均衡功能-异步请求可能会花费一些时间，但不会占用系统资源。现在，我们可以在端口8000上测试我们的应用程序。现在，我们要使用Linux强大的systemd服务和socket文件使服务器能够在程序中启动和停止。重要提示 systemd是Linux系统的进程和系统管理器。如果您想了解其功能和功能，我可以推荐（另一篇）非常有用的DigitalOcean知识库文章：<https://www.digitalocean.com/community/tutorials/systemd-essentials-working-with-services-units-and-the-journal>。在这些页面中，我们只会解释我们将使用的命令-启动、停止、启用和禁用服务、服务器等。我们将不得不使用一些nano，这是大多数Linux发行版的命令行文本编辑器。使用Ctrl + C停止gunicorn服务器，并使用简单的deactivate取消激活虚拟环境。前置的venv应该消失了。现在，让我们创建一个gunicorn socket。套接字是简单地在同一台或不同计算机上的通信点，使系统能够交换数据。当我们创建Gunicorn套接字时，这只是告诉系统创建的套接字可用于访问服务器将提供的的数据：`sudo nano /etc/systemd/system/gunicorn.socket` 文件的内容应如下（完全改编自前面提到的ASGI Django指南）：`[Unit] Description=gunicorn socket [Socket] ListenStream=/run/gunicorn.sock [Install] WantedBy=sockets.target`

使用Redis进行缓存以及在Ubuntu (DigitalOcean) 和Netlify上部署 5.

要退出nano, 请键入Ctrl +

X, 并在要求确认时键入yes。文件名应该与我们最初给出的相同。 6.

现在, 我们将创建gunicorn.service文件。再次使用以下命令启动nano: `sudo nano`

`/etc/systemd/system/gunicorn.service` 7. 开始输入以下内容: [Unit]

Description = gunicorn的守护进程 Requires = gunicorn.socket After =

network.target [Service] User = farmuser Group = www-data WorkingDirectory =

/home/farmuser/apiserver/FARM-chapter9-backend ExecStart =

/home/farmuser/apiserver/venv/bin/gunicorn \ --access-logfile - \ -k

uvicorn工作者.UvicornWorker \ --workers 3 \ --bind unix: /run/gunicorn.sock

\ main: app [Install] WantedBy = multi-user.target 我已经突出显示了您应该三

次检查的重要部分和路径, 然后保存。强调工作目录是托管我们的代码的目录, 而exec start是指虚拟环境目录。在我们的例子中, 它们并排放置在apiserver文件夹中! 这对于systemd应该足够了。 8.

保存文件, 让我们来试试吧。使用以下命令启动和启用新创建的gunicorn套接字:

`sudo systemctl start gunicorn.socket sudo systemctl enable gunicorn.socket`

将FastAPI部署到DigitalOcean（或任何Linux服务器！）

如果一切正常，就不应该有任何错误。但是，您应该检查套接字的状态：`sudo systemctl status gunicorn.socket` 您也应该检查gunicorn.sock文件的存在：`file /run/gunicorn.sock` 现在，激活套接字：`sudo systemctl status gunicorn` 这样，我们应该能够（终于！）用curl测试我们的API：`curl --unix-socket /run/gunicorn.sock localhost/cars/all`

由于我们已经访问了汽车端点，您应该会在终端下看到一堆汽车！

我们临近成功了！现在，我们将使用Nginx来路由传入的流量。请按照以下步骤操作：

重要提示 Nginx是一个非常强大、可靠且快速的Web服务器、负载均衡器和代理服务器。最基本的，Nginx读取其配置，并根据这些信息决定如何处理遇到的每个请求-它可以同时处理多个网站、多个进程和你向它扔出的最多样化的配置。您可能会在服务器上有一堆静态文件、图像和文档、由PM2管理的Node.js API、Django或Flask网站，以及可能同时存在的FastAPI实例。通过适当的配置，Nginx将能够轻松地处理这个混乱，并始终向正确的客户端提供正确的资源。至少了解Nginx如何操作的一些基本知识可以成为您手头非常有用的工具，而nginx.org网站是一个很好的起点。 13.

Nginx在服务器块中操作，因此让我们为我们的apiserver创建一个：`server { listen 80; server_name location = /favicon.ico { access_log off; log_not_found off; } location / { include proxy_params; proxy_pass http://unix:/run/gunicorn.sock;`

Redis缓存和在Ubuntu (DigitalOcean) 和Netlify上的部署 当你习惯了Nginx的服务器块语法, 你将很快为网站(或进程)提供服务。在上面的代码中, 我们指示Nginx在我们机器(IP地址)的默认端口(80)上监听, 并将所有流量重定向到我们的Unix Gunicorn套接字!

14. 现在, 通过以下方式将文件复制到Nginx的sites-enabled文件夹中以启用它: `sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled` 有一种非常方便命令可以让我们检查Nginx配置是否有效: `sudo nginx -t` 15. 如果Nginx没有抱怨, 我们可以通过输入以下命令来重新启动它, 然后我们就可以开始工作了: `sudo systemctl restart nginx` 16. 我们要做的最后一件事是再次设置ufw防火墙, 允许Nginx通过, 并通过删除允许其通过的规则来关闭端口8000: `sudo ufw delete allow 8000 sudo ufw allow 'Nginx Full'` 恭喜! 现在您可以通过由Uvicorn、Gunicorn和Nginx组成的强大系统提供API服务。通过这个设置, 我们有大量的选项。您可以通过Nginx来快速服务静态文件(图片、样式表或文档)。你也可以设置一个Next.js项目, 并通过PM2 (<https://pm2.keymetrics.io/>) 管理它, 这是一个强大的Node.js进程管理器。我们到此为止, 尽管在我们拥有一个生产就绪的系统之前, 还有许多不太复杂的步骤要经历。使用Redis添加缓存 Redis在NoSQL数据存储选项中排名前列, 与MongoDB非常不同。Redis是一个内存数据结构存储, 它可以用作数据库、缓存、消息代理, 也可以用于流媒体。Redis提供简单的数据结构——哈希、列表、字符串、集合等等, 并支持Lua语言脚本。虽然它可以用作主要的数据存储, 但通常用于缓存或运行分析等任务。由于它被构建为非常快(比MongoDB快得多, 要明确), 因此它非常适合缓存数据库或数据存储查询、复杂计算的结果、API调用以及管理会话状态。另一方面, MongoDB虽然快速且灵活, 但如果它足够大规模, 它可能会变慢。请记住, 我们经常(如本章)在一个服务器(Atlas Cloud)上托管MongoDB, 并使用另一个服务器来提供API服务。因此, 我们在这里添加Redis缓存来提高性能。下面是Redis配置的步骤:

17. 在您的服务器上安装Redis: `sudo apt update sudo apt install redis-server`

18. 验证Redis是否运行: `redis-cli ping` 如果响应是pong, 则Redis正在运行。

19. 打开项目的.env文件, 添加以下行: `REDIS_HOST=localhost # Redis服务器`
`REDIS_PORT=6379 # Redis端口 REDIS_DB=0 # Redis数据库`

20. 打开项目的Dockerfile文件, 并追加以下命令: `RUN pip install redis`

这将安装Redis的Python客户端。 21. 重新启动容器: `docker-compose down`

`docker-compose up -d` 22. 打开项目的FastAPI应用, 并引入redis包: `import redis`

23. 更新FastAPI应用程序的根路由以添加缓存。在此示例中, 我们将存储名为my_key和值为my_value的内容: `redis_store = redis.StrictRedis(`

`host=settings.REDIS_HOST, port=settings.REDIS_PORT, db=settings.REDIS_DB)`
`@app.get("/") async def read_root(): redis_store.set("my_key", "my_value")`
`return {"Hello": "World", "redis_value": redis_store.get("my_key")}`

24. 您应该能够看到类似以下输出的响应: `{"Hello": "World", "redis_value": "my_value"}` 恭喜! 现在您已经学会了如何将Redis缓存集成到我们的项目中。要了解有关Redis的更多信息, 请访问<https://redis.io/documentation>。

使用Redis添加缓存 如果我们将FastAPI代码部署到另一个平台（比如DigitalOcean或Heroku），响应时间可能会受到延迟的影响。想象一下，如果我们想执行复杂的聚合操作，而不是在本章中创建的简单操作，那么缓存就派上用场了！什么是缓存？这是一个非常简单的概念，已经存在了几十年，基本思想是将一些经常被请求的数据（从Mongo数据库中，例如）存储在某种类型的临时存储中，直到失效。第一个请求该资源（如汽车列表）的用户将不得不等待整个查询完成并获得结果。这些结果将自动添加到此临时存储（在我们的情况下，即Redis，数据库中的Usain Bolt）中，并提供给所有后续请求相同数据的用户。通常，我们认为相同的数据指的是相同的终端点。此过程持续到Redis中存储的数据（或者您可能使用的任何其他缓存解决方案）过期-如果在缓存中未找到有效数据，则再次进行真实数据库调用并重复此过程。过期时间在这里非常重要-在我们的情况下，如果我们正在与一家汽车销售公司合作，我们可以慷慨地使用缓存并将过期时间延长到10分钟或更长时间。在更动态的应用程序中，如论坛或类似的对话环境，更短的到期时间将是必需的，以维护功能。在Linux上安装Redis非常简单，而在Windows上它并没有官方支持。您可以按照官方指南在Windows上安装Redis进行开发（<https://redis.io/docs/getting-started/installation/install-redis-on-windows/>），但这超出了我们应用程序的范围。然而，我们将在DigitalOcean Linux盒子上安装Redis，并将缓存添加到我们的FastAPI应用程序中！请按照本章中的步骤连接到DigitalOcean盒子（或您选择的Linux系统-如果您正在Linux或Mac上开发，也应在那里安装Redis）：1. 然后，通过键入以下命令安装Redis：`sudo apt install redis-server` 重要提示 在生产环境中，您应该使用一个非常长的密码来保护您的Redis服务器。由于Redis速度快，攻击者可能在几秒钟内对其运行数十万个密码进行暴力攻击。您还应该禁用或重命名一些潜在危险的Redis命令。在这些页面中，我们仅显示如何向我们的设置中添加一个裸骨而不安全的Redis实例。

使用Redis缓存和部署在Ubuntu (DigitalOcean) 和Netlify上 2. 现在, 我们应该重新启动Redis服务。虽然它应该自动发生, 但让我们通过输入以下命令来确保: `sudo systemctl restart redis.service` 3.

通过输入以下命令进行测试, 以查看它是否工作: `sudo systemctl status redis` 终端将发送大量响应, 但您要查找的是绿色单词“Active (运行中)”。它还应自动启动每次重启 - 所以我们为此努力赚钱。 4.

测试Redis是否响应的传统方法是启动客户端: `redis-cli` 然后, 在Redis shell中, 输入ping。Redis应该响应pong, 并且提示应该说127.0.0.1: 6379。这意味着Redis正在本地主机 (Linux服务器) 的6379端口上运行。记住这个地址, 或更好的是, 将其写在某个地方 (我知道, 我知道)。我们将需要它用于我们的FastAPI服务器。

有许多方法使Redis与Python交互, 但在这里, 我们将选择一个名为Fastapi-cache的简单模块 (<https://github.com/long2ice/fastapi-cache>)。现在, 我们需要编辑/back end文件夹中的后端代码。完成后, 我们将推送更改到GitHub并重复部署过程。或者, 如果您只想快速尝试缓存, 则可以直接导航到目录并使用nano在DigitalOcean上直接编辑文件。无论如何, 请激活您选择的虚拟环境并安装软件包和aioredis (异步Python Redis驱动程序): `pip install fastapi-cache2 aioredis` 现在, 我们的FastAPI项目结构规定需要更新哪些文件。我们需要更新我们的main.py文件并添加以下导入:

```
import aioredis from fastapi_cache import FastAPICache from
fastapi_cache.backends.redis import RedisBackend
```

然后, 我们需要更新我们的启动事件处理程序: `@app.on_event (“startup”)`

```
async def startup_db_client ():          app.mongodb_client =
AsyncIOMotorClient (DB_URL)
```

添加 Redis 缓存 291 app.mongoddb = app.mongoddb_client[DB_NAME] redis = aioredis.from_url("redis://localhost:6379", encoding="utf8", decode_responses=True) FastAPICache.init(RedisBackend(redis), prefix="fastapi-cache") 这段代码很容易理解 - 我们获取了一个 Redis 客户端, 就像我们之前使用 Mongo 一样, 并传递了 URL 和一些 (建议性的) 设置。最后, 我们初始化了 FastAPICache。现在, 我们需要在端点中添加缓存装饰器, 这些端点位于 /routers/cars.py 文件中。我们将添加一个导入: from fastapi_cache.decorator import cache 现在, 我们可以装饰我们希望缓存的路由 (只有 GET 请求, 但实际上我们在这个项目中只需要这些)。编辑/sample 路由:

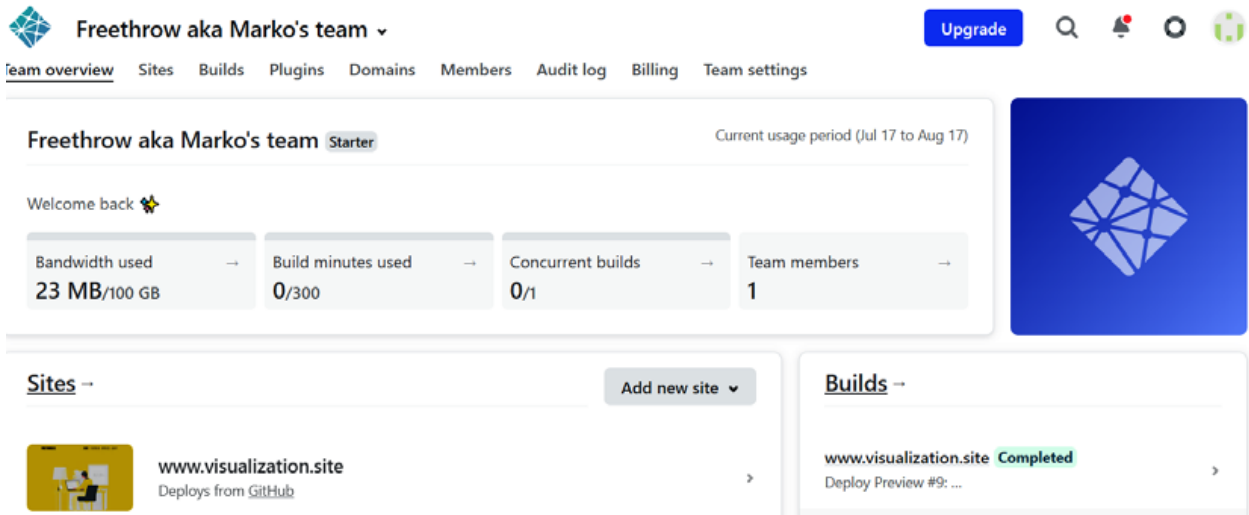
```
@router.get("/sample/{n}", response_description="Sample of N cars")
@cache(expire=60) async def get_sample(n: int, request: Request): query = [
    {"$match": {"year": {"$gt": 2010}}}, {"$project": {"_id": 0, } },
    {"$sample": {"size": n}}, {"$sort": {"brand": 1, "make": 1, "year": 1}}, ]
full_query = request.app.mongoddb["cars"].aggregate(query) results = [el
    async for el in full_query] return results
```

这个路由现在被缓存了, 这意味着当它被访问时, 它将提供一个大小为 N 的样本, 然后, 在接下来的 60 秒内的所有后续请求中, 它将发送相同的缓存响应。现在可以随意测试它, 无论是在您的 DigitalOcean API 上还是本地环境中, 具体取决于您在哪里实现了缓存。尝试在 1 分钟内访问 API - 您应该始终得到相同的结果, 直到缓存过期。恭喜 - 您刚刚为您的 API 添加了一个一流的缓存解决方案!

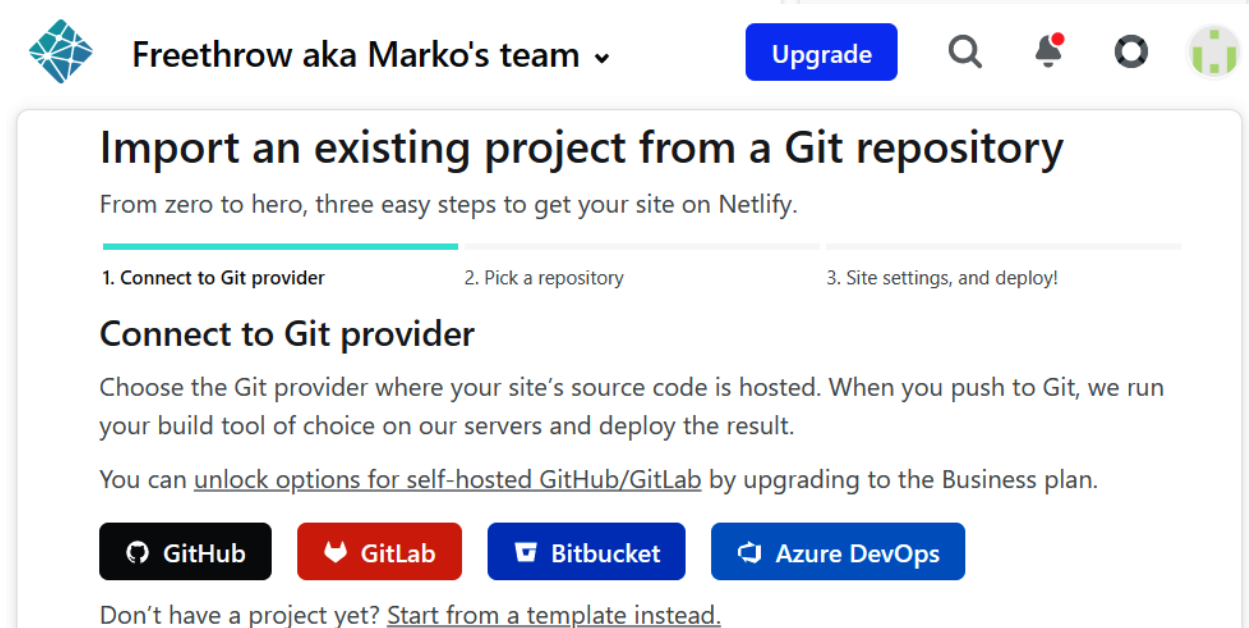
使用Redis缓存和在Ubuntu (DigitalOcean) 和Netlify上部署 292

在Netlify上部署前端 与Vercel类似, Netlify是提供静态Web托管和无服务器计算服务的顶级公司之一, 但还提供相当简单的CMS和优惠, 如表单处理。它被广泛认为是托管JAMStack网站的最佳解决方案之一, 其内容交付网络(CDN)可以显著加快托管的网站速度。这也是我们在本节中要使用它的原因, 来托管我们的React应用程序。使用您的Google或GitHub帐户登录后, 您将看到一个屏幕, 提供您部署新项目的可能性:

图片10.1-Netlify添加新站点按钮 接下来, 您将被要求是否导入现有项目(是!); 您应该从GitHub选择您的React前端项目。如果您使用GitHub登录, 则不必再次授权Netlify-如果没有, 请授权它: 图片10.2-Netlify上的导入现有项目页面



The screenshot shows the Netlify dashboard for a team named 'Freethrow aka Marko's team'. The top navigation bar includes links for 'Team overview', 'Sites', 'Builds', 'Plugins', 'Domains', 'Members', 'Audit log', 'Billing', and 'Team settings'. A blue 'Upgrade' button is visible in the top right. The main content area displays a 'Welcome back' message and a summary of team usage: Bandwidth used (23 MB/100 GB), Build minutes used (0/300), Concurrent builds (0/1), and Team members (1). Below this, there are sections for 'Sites' and 'Builds'. The 'Sites' section shows a site named 'www.visualization.site' with a 'Deploy from GitHub' button. The 'Builds' section shows a build for the same site with a 'Completed' status and a 'Deploy Preview #9: ...' link.



The screenshot shows the 'Import an existing project from a Git repository' page on Netlify. The page has a progress bar with three steps: '1. Connect to Git provider', '2. Pick a repository', and '3. Site settings, and deploy!'. The first step is currently active. Below the progress bar, the heading 'Connect to Git provider' is followed by a paragraph: 'Choose the Git provider where your site's source code is hosted. When you push to Git, we run your build tool of choice on our servers and deploy the result.' Below this, a link states: 'You can [unlock options for self-hosted GitHub/GitLab](#) by upgrading to the Business plan.' At the bottom, there are four buttons for selecting a Git provider: 'GitHub', 'GitLab', 'Bitbucket', and 'Azure DevOps'. A final link at the bottom says: 'Don't have a project yet? [Start from a template instead.](#)'

将React前端指向Netlify并保留所有Netlify能够从项目中巧妙地推断出的默认设置。您将看到一个页面，可以在该页面上修改任何部署设置，但我们将仅限于添加一个单一的环境变量。您已经猜到了——它就是方便的REACT_APP_API_URL！

图10.3-Netlify的预部署设置页面

Import an existing project from a Git repository

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Site settings, and deploy!

Site settings for freethrow/FARM-chapter9-frontend

Get more control over how Netlify builds and deploys your site with these settings.

Owner

Freethrow aka Marko's team



Branch to deploy

main



Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

[Learn more in the docs](#)

Base directory



Build command

npm run build



Publish directory

build



Advanced build settings

Define environment variables for more control and flexibility over your build.

Pro tip! Add a [netlify.toml](#) configuration file to your repository for even more flexibility.

New variable

Functions settings

Serverless functions are built and deployed along with the rest of your site.

[Learn more about Functions in the docs](#)

Functions directory



Deploy site

使用Redis进行缓存和在Ubuntu (DigitalOcean) 和Netlify上进行部署 294
您只需要在高级设置中添加一个变量：您猜对了 - REACT_APP_API_URL。按下相应的按钮创建一个新变量并将其命名为REACT_APP_API_URL。值应为https://yourdomain.com: 图10.4-在Netlify中添加新环境变量 经过一段时间，可能是一分钟左右，您的部署就可以向世界展示！如果出现任何问题（肯定会有问题），您应该检查Netlify的部署控制台并观察是否有问题。您的React前端和所有的炫酷图表及快速分页现在将由Netlify快速内容交付网络（CDN）提供服务，同时在DigitalOcean上由Nginx提供FastAPI（缓存）后端服务。加入之前探讨过的Heroku和Vercel部署，您有很多选择进行调试！这并不意味着这些是您唯一的部署选项！流行而坚实的选择是使用Docker容器并将应用程序（共同或分别）容器化，并向一些巨头 - 亚马逊网络服务（AWS）、Microsoft Azure或Google应用引擎提供此Docker镜像。这种部署类型与Heroku部署没有太大区别，尽管需要创建适当类型的账户并正确设置环境。这些解决方案的前期成本也更高。

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

[Learn more in the docs](#)

Base directory	i
Build command	i
Publish directory	i

Advanced build settings

Define environment variables for more control and flexibility over your build.

Pro tip! Add a [netlify.toml](#) configuration file to your repository for even more flexibility.

Key	Value	
REACT_APP_API_URL	http://207.154.254.1	ⓧ

在本章中，我们添加了一个非常简单但功能强大的基于Redis的缓存解决方案，Redis本身也是一个非常强大的产品。我们通过繁琐但经常必要的程序，在Ubuntu服务器上使用Gunicorn和强大的Nginx托管API，Nginx提供了如此多的灵活性和可配置性，以至于必须将其放在FARM堆栈的讨论中。作为一个额外的奖励，我们探索了另一个廉价（实际上是免费的）的前端托管选项Netlify，它提供了优秀的持续部署，并且与我们的所有前端解决方案非常兼容，无论是纯React、Next.js还是未来可能会用到的React-Remix。现在，您应该足够自信，可以直接跳入您的下一个项目中，通过与FastAPI、React和MongoDB的协作，探索众多可能性。在下一章中，我们将尝试解决涉及项目中每个堆栈组件的最佳实践问题，以及我们尚未涉及但同样重要的主题，例如测试、使用Jinja2的静态模板、站点监控等等。