CHAPTER 2

第二章

Adopting DevOps in Financial Systems

在金融系统中采用DevOps

Enough of the challenges. Let’s look at the drivers for adopting DevOps in financial systems, and how it can be done effectively.

挑战已经足够多了，让我们看看在金融系统中采用DevOps的驱动因素，以及如何有效地实现它。

# Chapter 2.1 Entering the Cloud

进入云端

One of the major drivers for DevOps in financial enterprises is the adoption of cloud services. Online financial institutions like exchanges or clearinghouses are essentially cloud services providers to the rest of the market. And most order and execution management system vendors are, or are becoming, SaaS providers to trading firms. So it makes sense for them to adopt some of the same ideas and design approaches as cloud providers: Infrastructure as Code; virtualization; rapid, automated system provisioning and deployment.

金融企业中采用DevOps的主要驱动因素之一是云服务的采用。像交易所或票据交换所这样的在线金融机构基本上是通过云服务提供商的方式向市场提供服务。大多数订单和执行管理系统供应商现在或正在成为贸易公司的SaaS供应商。因此，他们采用与云提供商相同的一些想法和设计方法是有意义的：基础设施即代码；虚拟化；快速、自动化的系统供应和部署。

The financial services industry is spending billions of dollars on building private internal clouds and using public cloud SaaS and PaaS (or private/public hybrid) solutions. This trend started in general-purpose backend systems, with HR, CRM, and office services using popular SaaS platforms and services like Microsoft’s Office 360 or Azure. Then it extended to development and testing,

providing on-demand platforms for Agile teams.

金融服务业正在花费数十亿美元建造私有内部云，并使用公共云SaaS和PaaS（或私有/公共混合）解决方案。这一趋势始于通用后端系统，包括使用流行的SaaS平台和服务（如Microsoft的Office 360或Azure）的HR、CRM和Office服务。然后，它扩展到开发和测试， 为敏捷团队提供随需应变的平台。

Now more financial services providers are taking advantage of public cloud platforms and tools like Hadoop for data intelligence and analytics, using cloud storage services for data archival. NASDAQ, for example, uses Amazon’s Redshift platform to run a massive data warehouse for data analytics and surveillance applications, adding several billion records per day.

现在，越来越多的金融服务提供商正在利用公共云平台和诸如Hadoop等工具进行数据智能和分析，并将云存储服务用于数据存档。例如，纳斯达克（Nasdaq）利用亚马逊的Redshift平台运行一个庞大的数据仓库，用于数据分析和监控应用，每天新增数十亿条记录。

Today, even regulators are in the cloud. The UK’s Financial Conduct Authority (FCA) is operating its new regulatory reporting systems on Amazon AWS, and FINRA’s new surveillance platform also runs on Amazon AWS.1 The SEC has moved its SEC.gov website and Edgar company filing system, as well as its MIDAS data analytics platform, to a private/public cloud to save operations and maintenance costs, improve availability, and handle surges in demand (such as the one that happened during Facebook’s IPO).2

如今，甚至连监管者也在云端。英国金融行为管理局（FCA）正在亚马逊AWS上运行其新的监管报告系统，Finra的新监控平台也在亚马逊AWS上运行。SEC已将其sec.gov网站、Edgar公司备案系统以及Midas数据分析平台移至私有/公共云，以节省运营和维护成本，提高可用性和应对需求激增（如Facebook IPO期间发生的情况）。

Cloud adoption has been held back by concerns about security and data privacy, data residency and data protection, and other compliance restrictions, according to a recent survey from the Cloud Security Alliance.3 However, as cloud platform providers continue to raise the level of reliability and transparency of their services, and improve auditing controls over operations, encryption, and ediscovery, and as regulators provide clearer guidance on the use of cloud

services, more and more financial data is making its way into the cloud.

云安全联盟（Cloud Security Alliance）最近的一项调查显示，由于对安全和数据隐私、数据驻留和数据保护以及其他合规性限制的担忧，云应用受到阻碍。然而，随着云平台提供商不断提高其服务的可靠性和透明度，并改进对操作、加密和电子数据展示的审计控制，随着监管机构对云服务的使用提供更清晰的指导，越来越多的金融数据正在进入云。

Cloud infrastructure giants like Amazon, Microsoft, and Google have made massive investments over the past few years in upgrading their data centers and improving their operational security and governance programs, learning with, and from, their customers along the way.

过去几年中，像亚马逊、微软和谷歌这样的云基础设施巨头在升级其数据中心、改进其运营安全和治理计划等方面进行了大量投资，向客户学习并和客户一起成长。

Amazon has worked with government regulatory agencies and industry pioneers including Intuit and Capital One to build advanced operational, security, and compliance capabilities into AWS. Unlike 10 years ago, when Netflix and a few internet startups gambled on moving their operations to the cloud despite major reliability and security risks, financial services organizations are now looking to cloud platforms like AWS to take advantage of its security and compliance strengths, as well as operational scalability.

亚马逊已经与政府监管机构和行业先锋（包括Intuit和Capital One）合作，将先进的运营、安全和合规能力构建到AWS中。与10年前不同的是，当Netflix和一些互联网初创公司冒着重大的可靠性和安全风险冒险将业务转移到云计算领域时，金融服务组织现在正寻求像AWS这样的云平台，以利用其安全性和合规性优势以及运营可扩展性。

This has provided financial technology startups like Monzo in the UK and Nubank in Brazil with a fast, scalable, and cost-effective path to launching new cloud-native services. But it is also clearing the road ahead for enterprises.

这为英国的Monzo和巴西的Nubank等金融技术初创企业提供了一条快速、可扩展且经济高效的新云本机服务发布途径。但这也为企业扫清了前进的道路。

One example: after running a series of experiments and successful production pilots, Capital One is now moving all of its business systems to AWS, and plans to completely shut down its internal data center operations within the next five years. According to Rob Alexander, Capital One’s CIO, they selected AWS because they could see clear advantages from a security and compliance perspective:

举个例子：Capital One在进行了一系列试验和成功的生产试点后，现在正将其所有业务系统转移到AWS，并计划在未来五年内完全关闭其内部数据中心运营。根据Capital One首席信息官Rob Alexander的说法，他们之所以选择AWS，是因为他们可以从安全和合规的角度看到明显的优势：

The financial service industry attracts some of the worst cyber criminals. We work closely with AWS to develop a security model, which we believe enables us to operate more securely in the public cloud than we can in our own data centers.

金融服务业吸引了一些最严重的网络罪犯。我们与AWS密切合作，开发一种安全模型，我们相信这种模型使我们能够在公共云中比在我们自己的数据中心中更安全地运行。

Operating a core financial service in the cloud still requires a lot of work. In the cloud provider’s Shared Responsibility Model, they set up and run secure data centers and networking for you and provide a set of secure platform configuration options and services. But it is still up to you to understand how to use these options and services correctly—and to make sure that your application code is secure.

在云端运行核心金融服务仍然需要大量的工作。在云提供商的共享责任模型中，他们为你设置和运行安全数据中心和网络，并提供一组安全平台配置选项和服务。但是，你仍然需要了解如何正确使用这些选项和服务，并确保应用程序代码是安全的。

# Chapter 2.2 Containers in Continuous Delivery

持续交付中的容器

Containers, and especially Docker—a lightweight and portable way to package and ship applications and to isolate them at runtime—are quickly becoming a standard part of many organizations’ DevOps toolkits. Now that Docker has mostly stabilized its platform ecosystem and APIs and is focusing on addressing security and enterprise management requirements, containers are making their way out of innovation labs and into enterprise development and test environments— and even into production.

容器，尤其是Docker——一种轻量级的、可移植的打包和运输应用程序以及在运行时隔离应用程序的方法，正在迅速成为许多组织的DevOps工具包的标准部分。现在，Docker已经基本上稳定了平台生态系统和API，并专注于解决安全和企业管理需求，容器正在走出创新实验室，进入企业开发和测试环境，甚至进入生产。

Some of the organizations that we’ll look at in this report, such as ING, Intuit, and Capital One, are using Docker to package and ship applications for developers and for testing as part of their build pipelines, and in production pilots.

我们将在本报告中看到的一些组织，如Ing、Intuit和Capital One，正在使用Docker为开发人员打包和发布应用程序，并将其作为构建管道以及生产试点的一部分进行测试。

Others have gone much further. PayPal, which operates one of the world’s largest private clouds, managing hundreds of thousands of virtual machines in data centers across the world, has moved thousands of production payment applications onto Docker in order to reduce its operations footprint and to speed up deployment and rollback. PayPal is also using containers to run older legacy applications on modern OS kernels. The International Securities Exchange runs its low-latency production data centers on CoreOS. And Goldman Sachs is in the process of moving thousands of applications into Docker to simplify operations and reduce costs. It expects to shift 90% of all its production computing workloads into containers.

其他组织走得更远。PayPal运营者世界上最大的私有云之一，管理着世界各地数据中心的数十万台虚拟机，已经将数以千计的生产支付应用程序移到了DOCKER上，以减少其业务占用，加速部署和回滚。PayPal还使用容器在旧OS内核上运行旧的遗留应用程序。国际证券交易所在Coreos上运行其低延迟生产数据中心。高盛正在将数千个应用程序转移到Docker中，以简化操作并降低成本。它预计将把90%的生产计算工作转移到容器中。

# Chapter 2.3 Introducing DevOps: Building on Agile

介绍DevOps：建立在敏捷基础之上

DevOps is a natural next step in organizations where Agile development has been adopted successfully. Development teams who have proven that they can iterate through designs and deliver features quickly, and the business sponsors who are waiting for these features, grow frustrated with delays in getting systems into production. They start looking for ways to simplify and streamline the work of acceptance testing and security and compliance reviews; dependency analysis and packaging; and release management and deployment.

在成功采用敏捷开发的组织中，DevOps是一个自然的下一步。已经证明可以快速迭代设计和交付功能的开发团队，以及等待这些功能的业务发起人，对系统投入生产的延迟感到沮丧。他们开始寻找简化和简化验收测试、安全性和合规性审查、依赖性分析和打包以及发布管理和部署工作的方法。

Agile development has already been proven to reduce software project costs and risks. DevOps aims to solve even more important problems for financial services enterprises: mitigating operational risks and reducing operations support and maintenance costs.

敏捷开发已经被证明可以降低软件项目的成本和风险。DevOps旨在为金融服务企业解决更重要的问题：降低运营风险，降低运营支持和维护成本。

**Capital One: From Agile to DevOps**

**Capital One：从敏捷走向Devops**

The ING story is continuing in a way at Capital One, the largest digital bank in the US, which purchased ING Direct USA in 2012. Until then, Capital One outsourced most of its IT. Today, Capital One is fully committed to Agile and DevOps.

美国最大的数字银行Capital One在2012年收购了美国ING Direct。在此之前，Capital One将大部分业务外包出去。今天，Capital One完全致力于敏捷和开发。

Capital One’s Agile experiment started in late 2011, with just two teams. As more teams were trained in Agile development, as at ING, they found that they were building software quickly, but it was taking too long to get working software into production. Development sprints led to testing and hardening sprints before the code was finally ready to be packaged and handed off to production. This wasn’t Agile; it was “Agilefall.”

Capital One的敏捷实验始于2011年底，只有两个团队。随着越来越多的团队接受了敏捷开发方面的培训，比如在ING，他们发现他们正在快速构建软件，但要将工作软件投入生产还需要很长时间。在代码最终准备好打包并交付生产之前，开发冲刺导致了测试和强化冲刺。这不是敏捷，而是“敏捷瀑布”。

Capital One developers were following the Scaled Agile Framework (SAFe). They leveraged the idea of System Teams in SAFe, creating dedicated DevOps teams in each program to help streamline the handoffs between development and operations. These teams were responsible for setting up and managing the development and test environments, for automating build and deployment processes, and for release management, acting as “air traffic controllers to navigate through the CABs.”

Capital One开发人员正在遵循规模化敏捷框架（SAFE）。他们充分利用了系统团队的思想，在每个项目中创建专门的DevOps团队，以帮助简化开发和运维之间的切换。这些团队负责建立和管理开发和测试环境，自动化构建和部署过程，以及发布管理，充当“空中交通管制员在驾驶室中导航”。

Integration testing, security testing, and performance testing were all being done outside of development sprints by separate test teams. They brought this testing into the dedicated DevOps teams and automated it. Then they moved all testing into the development sprints, adopting behavior-driven/acceptance test–driven development and wiring integration, security, and performance testing into a Continuous Delivery pipeline. Today they have 700 Agile teams following Continuous Delivery. Some teams are pushing changes to production as often as 20 times per day.

集成测试、安全测试和性能测试都是由独立的测试团队在开发冲刺之外完成的。他们将这个测试引入专门的DevOps团队并自动化它。然后，他们将所有测试都转移到开发冲刺中，采用行为驱动/验收测试——驱动开发和连接集成、安全性和性能测试，并将其引入持续的交付管道中。如今，他们拥有700个敏捷团队以持续交付。一些团队正在提交变更到生产环境，每天多达20次。

Agile ideas and principles—prioritizing working software over documentation, frequent delivery, face-to-face collaboration, and a focus on technical excellence and automation—form the foundation of DevOps. And Continuous Delivery, which is the control framework for DevOps, is also built on top of a fundamental Agile development practice: Continuous Integration.

敏捷的思想和原则优先于工作文档，频繁交付，面对面协作，注重技术卓越和自动化，这是DeVOPS的基础。而持续交付，作为DevOps的控制框架，也建立在一个基本的敏捷开发实践之上：持续集成。

# Chapter 2.4 From Continuous Integration to Continuous Delivery

从持续集成到持续交付

In Continuous Integration, developers make sure that the code builds and runs correctly each time that a change is checked in. Continuous Delivery takes this to the next step.

在持续集成中，开发人员确保每次提交更新的代码能正确地构建和运行。连续交付将处理下一步。

It’s not just about automating build steps and unit testing (something that the development team owns). Continuous Delivery is about provisioning and configuring test environments to match production as closely as possible, automatically; packaging the code and deploying it to test environments, automatically; running acceptance tests and stress tests and performance tests and security tests and other checks, with pass/fail feedback to the team—again, automatically. It’s about making sure that the system is always ready to be deployed to production, and making sure that it can be deployed safely. And it’s about tracking all of these steps and making the status transparent to everyone.

这不仅仅是自动化构建步骤和单元测试（开发团队拥有的东西）。持续交付是指提供和配置测试环境，以尽可能地自动匹配生产；自动打包代码并将其部署到测试环境；运行验收测试、压力测试、性能测试、安全测试和其他检查，并向团队自动提供通过/失败反馈。这是为了确保系统始终准备好部署到生产环境中，并确保它可以安全部署。它能够跟踪所有这些步骤，使状态对每个人都可见。

Continuous Delivery is the backbone of DevOps. It’s an automated framework for making software and infrastructure changes, and pushing out software upgrades, patches, and changes to configurations. Making sure that all changes are repeatable, predictable, efficient, transparent, and fully audited.

持续交付是DevOps的支柱。它是一个自动化的框架，用于对软件和基础设施进行更改，并推出软件升级、补丁和配置更改。确保所有更改都是可重复、可预测、高效、透明和完全审计的。

Putting a Continuous Delivery pipeline together requires a high degree of cooperation between development and operations, and a much greater shared understanding of how the system works, what production really looks like, and how it runs. It forces teams to start talking to each other, exposing details about how they work—and shining a bright light on problems and inefficiencies.

把一个连续的交付管道放在一起需要开发和操作之间高度的合作，并且需要对系统如何工作、生产实际是什么样子以及它如何运行有更大程度的共享理解。它迫使团队开始互相交谈，公开他们如何工作的细节，并能够及时的发现问题和低效率。

There is a lot of work that needs to be done:

这就有很多工作要做：

1. Mapping out and understanding the engineering workflows and dependencies from check-in to release

2. Standardizing configurations, and bringing configuration into code

3. Cleaning up the build—getting rid of inconsistencies, hardcoding, and jury-rigging

4. Putting everything into version control: application code and configuration, tests, binary dependencies (like the Java Runtime), infrastructure configuration recipes and manifests, database schemas, deployment scripts, and configurations for the CI/CD pipeline itself

5. Replacing time-consuming manual reviews and testing steps and acceptance checklists with fast automated scans and repeatable automated test suites (and checking all of this into version control too)

6. Getting all of the steps for deployment together and automating them carefully, replacing operations runbooks and checklists with automated deployment instructions and release orchestration

7. Doing all of this in a heterogeneous environment, with different architectures and technology platforms and languages

1。映射和理解从代码提交到版本发布整个工程工作流和依赖关系

2。标准化配置，并将配置转换为代码

3。清理构建，消除不一致、硬编码和临时或者应急配置

4。将所有内容放入版本控制：应用程序代码和配置、测试、二进制依赖关系（如Java运行时）、基础设施配置和清单、数据库模式、部署脚本和CI/CD流水线本身的配置

5。用快速自动扫描和可重复的自动测试套件替换耗时的手动检查、测试步骤和验收检查表（并将所有这些检查都加入到版本控制中）

6。将部署的所有步骤放在一起并小心地实现自动化，用自动部署说明和版本发布说明替换操作运行手册和检查表。

7。在不同的体系结构、技术平台和语言的异构环境中完成所有这些工作

This work isn’t product development, and it’s not operations either. This can make it hard to build a business case for: it’s not about delivering specific business features or content, and it can take time to show results. But the payoff can be huge.

这项工作不是产品开发，也不是运维。这可能会使为其构建业务案例变得困难：这与提供特定的业务功能或内容无关，而且得到结果可能需要一些时间。但回报可能是巨大的。

Continuous Delivery at LMAX

持续交付在LMAX

The London Multi-Asset Exchange (LMAX) is a highly regulated FX retail market in the UK, where Dave Farley (coauthor of the *Continuous Delivery* book) helped pioneer the model of Continuous Delivery. LMAX’s systems were built from scratch following Agile best practices: test-driven development (TDD), pair programming, and Continuous Integration. But LMAX took this further, automatically deploying code to integration, acceptance, and performance testing environments, building up a Continuous Delivery pipeline.

伦敦多资产交易所（LMAX）是英国一个受高度管制的外汇零售市场，戴夫·法利（Continuous Delivery Book的合著者）在那里帮助开拓了连续交付模式。LMAX的系统是根据敏捷最佳实践从头构建的：测试驱动开发（TDD）、结对编程和持续集成。但LMAX更进一步，自动将代码部署到集成、验收和性能测试环境中，从而构建了一个连续的交付管道。

LMAX has gone all in on automated testing. Each build runs through 25,000 unit tests with code coverage failure, simple code analysis (using tools like FindBugs, PMD, and custom architectural dependency checks), and automated integration sanity checks. All of these tests and checks must pass for every piece of code submitted.

LMAX已经在自动化测试上投入了所有工作。每个构建运行25000个单元测试，其中包括代码覆盖失败、简单的代码分析（使用FindBugs、PMD和自定义体系结构依赖性检查等工具）和自动化的集成健全性检查。所有这些测试和检查都必须通过提交的每一段代码。

The last good build is automatically picked up and promoted to integration and acceptance testing, where more than 10,000 end-to-end tests are run on a test cluster, including API-level acceptance tests, multiple levels of performance tests, and fault injection tests that selectively fail parts of the system and verify that the system recovers correctly without losing data. More than 24 hours’ worth of tests are executed in parallel in less than 1 hour.

最后一个良好的构建被自动挑选出来，并进一步进行集成和验收测试，在一个测试集群上运行10000多个端到端测试，包括API级别的验收测试、多个性能测试和故障容灾测试，这些测试有选择地使系统的某些部分失效，并验证系统恢复正常，并且不会丢失数据。超过24小时价值的测试在不到1小时的时间内并行执行完成。

If all of the tests and reviews pass, the build is tagged. All builds are kept in a secure repository, together with dependent binaries (such as the Java Runtime). Everything is tracked in version control.

如果所有的测试和评审都通过了，那么构建就会被标记。所有构建都保存在安全的存储库中，以及依赖的二进制文件（如Java运行时）。所有内容都通过版本控制来跟踪。

QA can conduct manual exploratory testing or other kinds of tests on a build. Operations can then pull a tagged build from the development repository to their separate secure production repository, and use the same automated tools to deploy to production. Releases to production are scheduled every two weeks, on a Saturday, outside of trading hours.

QA可以在一个构建上执行手动探索性测试或其他类型的测试。然后，运维人员可以将标记的构建从开发存储库拉到各自独立的安全生产存储库中，并使用相同的自动化工具部署到生产环境中。从而可以在每两周的周六，在交易时间之外，发布到生产中。

There is nothing sexy about the technology involved: they rolled a lot of the tooling on their own using scripts and simple conventions. But it’s everything that we’ve come to know today as Continuous Delivery.

这项技术并没有什么吸引人的地方：他们利用脚本和简单的约定开发了很多工具。但这就其实就是我们今天所熟知的持续交付。

Protecting Your Pipeline

保护你的管道

DevOps in a high-integrity, regulated environment relies heavily on the audit trail and checks in the Continuous Delivery pipeline. The integrity and security of this environment must therefore be ensured:

DevOps在一个高度集成、受监管的环境中，严重依赖于持续交付管道中的审计跟踪和检查。因此，必须确保该环境的完整性和安全性：

1. Every step must be audited, from check-in to deployment. These audit logs need to be archived as part of records retention.

2. You have to be able to prove the identity of everyone who performed an action: developers checking in code, reviewers, people pulling or pushing code to different environments. Do not allow anonymous, public access to repos or build chains.

3. You need to ensure the integrity of the CI/CD pipeline and all the artifacts created by it, which means securing access to the version control system, the Continuous Integration server configuration, the artifact repositories and registries containing the

binaries and system configuration data and other dependencies, and all of the logs.

4. Build and deployment tools require keys and other credentials. Keep credentials and other sensitive information out of code and runtime configuration using a secure secrets manager like HashiCorp’s Vault.

5. Separate your development and production repositories. Only authorized people should be able to pull from a development repository to the production repository, and again, make sure that all of these actions are audited.

6. Use “PhoenixServers” for build and test steps. Take advantage of tools like Docker, Packer, Ansible, and Chef to automatically provision and configure servers when you need them, ensuring that they are always in a known and reproducible state, and then tear them down after the work is done, to reduce your attack surface.

7. Harden all of the tools, and the infrastructure that they run on. Never rely on vendor defaults, especially for developer tools.

1。从提交到部署，每个步骤都必须经过审计。这些审计日志需要作为记录保留的一部分进行存档。

2。你必须能够证明执行操作的每个人的身份：开发人员签入代码、审查人员、将代码拉入或推送到不同环境的人员。不允许匿名、公开访问repos或构建链条。

3。你需要确保CI/CD管道及其创建的所有构件的完整性，这意味着要确保对版本控制系统、持续集成服务器配置、构件存储库和包含二进制文件和系统配置数据以及其他依赖项，以及所有日志。

4。构建和部署工具需要密钥和其他安全凭据。使用安全机密管理器（如Hashicorp的保险库）将凭证和其他敏感信息保存在代码和运行时配置之外。

5。分离开发和生产存储库。只有授权人员才能从开发存储库拉到生产存储库，并且再次确保所有这些操作都经过审计。

6。使用“phoenix服务器”进行构建和测试步骤。利用Docker、Packer、Ansible和Chef等工具的优势，在需要时自动提供和配置服务器，确保它们始终处于已知和可复制状态，然后在完成工作后将其拆除，以减少可攻击范围。

7。强化所有工具以及它们运行的基础设施。不要依赖供应商的默认值，尤其是开发人员工具。

Treat your build and deployment pipeline as an extension of your production operations environment. Because that’s what attackers will do.

将构建和部署管道视为生产环境操作的延申。因为攻击者就是这样做的。

Test Automation

自动化测试

Financial services organizations spend enormous amounts of time and money on testing: functional testing of course, integration testing, performance and capacity testing, resilience testing, compliance testing, security testing, and regression testing to make sure that changes don’t break existing behavior and interfaces. Regulators require proof that core financial systems have been thoroughly tested before they can be released to production. Some regulatory guidance even lays out how testing needs to be conducted. For example, MiFID II requires trading firms to test their trading systems and algorithms with exchanges, which in turn have to maintain production-like testing facilities with representative data.

金融服务组织在测试上花费了大量的时间和金钱：功能测试、集成测试、性能和容量测试、弹性测试、合规性测试、安全测试和回归测试，以确保新的改动不会破坏现有的行为和接口。监管者需要证明核心金融系统在投入生产之前已经经过了彻底的测试。一些监管的法规至规定了如何进行测试。例如，MiFID II要求交易公司在交易所测试其交易系统和算法，而交易所又必须类似生产环境中有代表性数测试数据的测试设施。

Automated testing is fundamental to Continuous Delivery. Without automated tests, you can’t do Continuous Delivery of changes through a pipeline. While some organizations (like exchanges) have invested a lot in automating testing, many financial institutions still rely heavily on manual testing for important areas like functional acceptance testing, integration testing, and security testing. A PwC study in 2014 found that only 15% of testing activities had been automated at major financial institutions.5

自动化测试是持续交付的基础。如果没有自动化测试，你就不能通过管道进行持续交付。虽然一些组织（如交易所）在自动化测试方面投入了大量资金，但许多金融机构仍然严重依赖于功能验收测试、集成测试和安全测试等重要领域的手动测试。普华永道2014年的一项研究发现，只有15%的测试活动在主要金融机构实现了自动化。

Because manual testing for large systems is so expensive, many firms outsource or offshore testing to take advantage of lower-cost skills, handing the code off to teams in another time zone in a “follow the sun” approach to be tested overnight.

由于大型系统的手工测试非常昂贵，许多公司通过外包或离岸测试从而降低成本，将代码交给另一个时区的团队，采用“跟随太阳”的方法，从而可以整晚进行测试。

But Agile development, especially for web and mobile applications, is already pushing organizations away from manual acceptance testing and offshore test teams and toward automated testing in-phase by developers, in order to keep up with the pace of change in rapid, iterative development. DevOps and Continuous Delivery push this even further, creating continuous and immediate feedback loops for each and every change.

但是，敏捷开发，特别是对于Web和移动应用程序，已经将组织从手动验收测试和离岸测试团队推到开发人员正在进行的自动化测试阶段，以跟上快速、迭代开发的变化步伐。DevOps和持续交付进一步推动了这一点，为每一个变化创建了连续和即时的反馈循环。

The path toward automated testing is straightforward, but it’s not easy. It starts with the basics of Continuous Integration: automating unit testing and functional testing, and moving responsibility for regression testing away from QA and onto developers.

实现自动化测试的方法很简单，但并不容易。它从持续集成的基础开始：自动化单元测试和功能测试，并将回归测试的责任从QA转移到开发人员身上。

This makes sense to teams already practicing Agile development and TDD. It’s much harder when you’re working on monolithic legacy systems that were never designed to be testable. Here, you can get help from Michael Feathers and his excellent book *Working Effectively with Legacy Code* (Prentice Hall), which explains how to cover a system with tests before, and as, you refactor the code.

对于已经在实践敏捷开发和TDD的团队来说，这是有意义的。当你在处理从未被设计成可测试的单块遗留系统时，这要困难得多。你可以从Michael Feathers和他的优秀著作《有效地处理遗留代码》（Prentice Hall）中获得帮助，这本书解释了如何用以前的测试覆盖系统，以及如何重构代码。

Continuous Delivery requires a big investment up front in setting up testing infrastructure, moving testing responsibilities from offshore test teams into development, creating virtualized test platforms, selecting test tools and wiring them into your build pipelines, and generating synthetic test data or anonymizing copies of production data to protect confidentiality and privacy of information.

持续交付需要大量的投资，从建立测试基础设施，将测试职责从离岸测试团队转移到开发，创建虚拟化测试平台，选择测试工具并将其连接到构建管道，到生成合成测试数据或者脱敏处理生产数据从而保护信息的机密性和隐私。

It will take a long time to write the thousands (or tens or hundreds of thousands) of automated tests needed to cover a big financial system. Many of the most important of these tests—integration tests, performance and capacity tests, security tests—are particularly difficult to automate in Continuous Delivery. Let’s look at what is needed to get this done.

要编写覆盖一个大型金融系统所需的数千（或数万或数十万）自动测试需要很长时间。在这些测试中，许多最重要的集成测试、性能和容量测试、安全测试在连续交付中尤其难以自动化。让我们看看完成这项工作需要什么。

Integration Testing

集成测试

With the exception of online retail applications such as online banking, most financial transactions are done system-to-system through APIs. Central capital markets institutions like exchanges or major clearinghouses can be connected to hundreds of trading firms, while large OMS/EMS systems at trading firms may be connected to dozens of different trading venues and market data sources and backoffice systems, all through different protocols. This makes integration testing and end-to-end testing at the API level critically important.

除了网上银行等网上零售应用程序外，大多数金融交易都是通过API进行系统到系统的交易。中央资本市场机构（如交易所或主要票据交换所）可以与数百家贸易公司相连，而贸易公司的大型OMS/EMS系统可以通过不同的协议与数十个不同的贸易场所、市场数据源和后台系统相连。这使得集成测试和API级别的端到端测试变得非常重要。

Regression testing of these interfaces is expensive and difficult to set up. Because test systems are not always available and are often not deterministic, you’ll need to stub them out, creating test doubles or simulators that behave in predictable ways.

这些接口的回归测试是昂贵的，并且很难设置。因为测试系统并不总是可用的，并且通常不具有确定性，所以你需要排除不确定性，创建以可预测方式运行的测试双重对象或模拟器。

There are risks to testing using simulators or service virtualization test harnesses. Because you’ve made the testing environment predictable and deterministic, you won’t catch the kinds of exceptions and problems that happen in real-life systems, and that can lead to wide-scale failures. Race conditions, timeouts, disconnections, random behavior, and other exceptions will escape your automated testing safety net—which means that your exception-handling code needs to be carefully reviewed.

使用模拟器或服务虚拟化测试工具进行测试存在风险。因为你已经使测试环境具有可预测性和确定性，所以你不会捕获到在现实系统中发生的各种异常和问题，而这些异常和问题会导致大规模的失败。竞争条件、超时、断开连接、随机行为和其他异常将逃脱你的自动化测试安全网，这意味着需要仔细检查异常处理代码。

This also means that if you’re making changes that could affect outside behavior, you need to do certification testing with other parties. Luckily, for widely used financial protocols like FIX or SWIFT at least, there are several automated tools and test platforms to help with this.

这也意味着，如果你正在进行可能影响外部行为的更改，则需要与其他方进行认证测试。幸运的是，对于广泛使用的金融协议，如FIX或SWIFT，至少有几个自动化工具和测试平台可以帮助实现这一点。

One potential shortcut to automating integration testing in large systems is through model-based testing. According to Bob Binder at Carnegie Mellon’s Software Engineering Institute, a well-defined protocol specification such as FIX or SWIFT can be used to automatically generate many of the integration test cases needed to cover the behavior of a complex system, including catching mistakes in detailed scenarios that could trip up a system in production.6

在大型系统中自动化集成测试的一个潜在捷径是通过基于模型的测试。根据卡内基梅隆软件工程研究所的Bob Binder的说法，一个定义良好的协议规范（如FIX或SWIFT）可以用来自动生成覆盖复杂系统行为所需的许多集成测试用例，包括在可能使产品中的系统崩溃的详细场景中捕捉错误。

Model-based testing is still a niche idea outside of safety-critical systems engineering, but this is changing, at least in some parts of the financial industry.

基于模型的测试仍然是安全关键系统工程之外的一个新的市场概念，但这正在改变，至少在金融行业的某些领域是如此。

Jim Northey, a financial systems integration testing expert and the Global Technical Committee Chair of the FIX Trading Community, is helping to lead an initiative called “FIX Orchestra” to build open source machine-readable versions of FIX specifications. These formal, machine-readable rules of engagement can be exchanged and compared between systems to catch incompatibilities, and eventually used by protocol engines to automatically map between protocol implementations. They could also be fed into model-based testing tools to automatically generate integration tests. The first release of the FIX Orchestra specifications and tools was published at the end of 2016.

Jim Northey是金融系统集成测试专家，也是Fix贸易界的全球技术委员会主席，他正在帮助领导一项名为“Fix Orchestration”的计划，以构建Fix规范的开源的机器可读版本。这些正式的、机器可读的规则可以在系统之间进行交换和比较，以捕获不兼容性，并且最终被协议引擎用来在协议实现之间自动映射。它们还可以被输入到基于模型的测试工具中，以自动生成集成测试。Fix Orchestration规范和工具的第一个版本于2016年底发布。

Engineers at NASDAQ are also working on model-based testing for their matching engine. And Aesthetic Integration, a London-based fintech startup, offers a formal verification tool called Imandra that will build a mathematical model of a trading algorithm or blockchain contract so that its specifications can be automatically checked for consistency and correctness.

纳斯达克的工程师们也在为他们的匹配引擎进行基于模型的测试。Aesthetic Integration，一家总部位于伦敦的金融技术初创公司，提供了一个名为Imandra的正式验证工具，该工具将构建交易算法或区块链合同的数学模型，以便自动检查其规范的一致性和正确性。

Performance and Capacity Testing

性能和能力测试

Regulators mandate regular capacity testing to ensure that financial systems can hold up to demand. Online trading and market data and risk management systems are all extremely sensitive to latency, which means that even small changes have to be carefully tested to ensure that they don’t slow down latency-critical parts of the system.

监管机构要求定期进行能力测试，以确保金融系统能够满足需求。在线交易和市场数据以及风险管理系统对延迟都非常敏感，这意味着即使是小的更改也必须经过仔细测试，以确保它们不会降低系统可能会带来延迟的关键部分。

There are three basic kinds of performance tests that need to be automated:

1. System load testing using standard workloads against a baseline

2. Stress testing to find the edge of the system’s capability

3. Micro-benchmark tests at the functional/unit level in performance-critical sections of code, to closely measure latency and catch small degradations

有三种基本的性能测试需要自动化：

1。根据基线使用标准工作负载进行系统负载测试

2。通过压力测试找出系统能力的边界

3。在代码的性能关键部分的功能/单元级别进行微基准测试，以密切测量延迟并捕获小的降级。

Some of the challenges in implementing automated, continuous performance testing include:

• Creating a controlled test environment and protecting tests from runtime variability, including runtime jitter for microbenchmarks

• Designing representative scenarios and building load generation tools that handle financial protocols

• Putting an accurate measurement system in place (including instrumenting the system and capturing metrics, down to microseconds)

• Deciding on clear pass/fail criteria

在实施自动化的、连续的性能测试方面的一些挑战包括：

•创建一个受控的测试环境，保护测试不受运行时变化的影响，包括运行时微小的抖动

•设计具有代表性的方案并构建处理财务协议的负载生成工具

•建立准确的测量系统（包括对系统进行检测和捕获度量，最短为微秒）

•确定明确的合格/不合格标准

The tricky part will be integrating all of this cleanly into Continuous Delivery, in a simple, repeatable way. From a legal standpoint, you should also be careful in how you design and implement automated performance testing in Continuous Delivery, to make sure that you don’t step on the patent that HPE has filed on doing this.

这之中比较棘手的部分是将以一种简单、可重复的方式将所有这些内容清晰地集成到连续交付中。从法律的角度来看，你还应该注意如何在持续交付中设计和实现自动化性能测试，以确保你不会碰到HPE为此提交的专利的麻烦。

Security Testing

安全测试

Automating security testing in Continuous Delivery requires a rethink of how security testing is done. We’ll look at how to do this in detail in “DevOpsSec: Security as Code” on page 42.

在持续交付中自动化安全测试需要重新考虑如何进行安全测试。我们将在 “DevOpsSec: security as code”的第42页详细介绍如何实现这一点。

Automated Infrastructure Testing

自动基础设施测试

Infrastructure as Code introduces a new dimension to operations engineering. It requires a disciplined software engineering approach to provisioning and configuring systems: no more ad hoc scripting or manual configuration or hardening steps.

基础设施作为代码为运维工程引入了一个新的维度。它要求使用一种规范的软件工程方法来提供和配置系统：不再需要临时脚本、手动配置或强化步骤。

Operations engineers need to understand and follow the same coding disciplines as application developers. This includes writing automated unit tests and integration tests using frameworks like rspecpuppet, Chef Test Kitchen, or Serverspec; learning about test-driven infrastructure and how tests should drive design and implementation; and wiring these tests into Continuous Integration and Continuous Delivery as part of an automated configuration management pipeline. Like developers, they need to learn to spend as much time, or more time, writing tests as they do writing code. And, like developers, they need to learn how to make changes in small, safe, incremental and iterative steps.

运维工程师需要理解和遵循与应用程序开发人员相同的编码规则。这包括使用rspecpupper、chef test kitchen或serverspec等框架编写自动化单元测试和集成测试；学习测试驱动的基础设施以及测试应如何驱动设计和实现；以及将这些测试关联到持续集成和持续交付中，作为自动化配置管理管道的一部分。和开发人员一样，他们需要学习花费和编写代码一样多的时间或更多的时间来编写测试。而且，和开发人员一样，他们需要学习如何在小的、安全的、增量的和迭代的步骤中进行更改。

Manual Testing in Continuous Delivery

持续交付中的手工测试

In Continuous Delivery, you try to automate testing as much as possible. All of these tests have to be designed to run within short time constraints, which might mean breaking tests into parallel pipelines and executing them across a grid (like LMAX did, as discussed earlier in this chapter).

在连续交付中，你尽量使测试自动化。所有这些测试都必须设计为在短时间内运行，这可能意味着将测试拆分为并行管道，并跨交错地执行它们（就像本章前面讨论的LMAX做的一样）。

But there is still an important place for manual testing in large, business-critical systems. In particular, a manual approach is important for:

• Risk-based exploratory testing to look for holes and edge cases, including group-based multiparty testing sessions or “war games.” Multiparty testing can be an especially useful way to find important bugs (like timing problems and race conditions and workflow problems) in interactive, online systems such as trading systems, by trying to recreate real-world conditions and introducing some randomness and stressors into testing. This is about bug hunting, not acceptance testing.

• Usability testing for any user interfaces.

• Penetration testing and other kinds of adversarial or destructive testing: trying to break things to see what happens.

但是在大型的、业务关键的系统中，手工测试仍然有一个重要的位置。特别是，手动方法对于以下方面很重要：

•基于风险的探索性测试，以寻找漏洞和边界用例，包括基于组的多方测试会话或“战争游戏”。多方测试可以是一种特别有用的方法，通过尝试重新定义交易系统等交互式在线系统中的重要错误（如计时问题、比赛条件和工作流问题）。创造真实的环境，在测试中引入一些随机性和压力因素。这是用来搜寻bug，而不是验收测试。

•任何用户界面的可用性测试。

•渗透测试和其他类型的对抗性或破坏性测试：尝试打破东西，看看会发生什么。

With Continuous Delivery, there is always a window where this kind of testing can and should be done.

对于持续交付，总需要有这样的时间窗口应该在其中进行这种测试。