AWS 白皮书

在 AWS 上练习持续集成和持续交付

在 AWS 上练习持续集成和持续交付: AWS 白皮书

Table of Contents

在 AWS 上练习持续集成和持续交付

发布日期：2021 年 10 月 27 日（文档修订）

摘要

本白皮书介绍了在软件开发环境中使用持续集成/持续交付 (CI/CD) 以及 Amazon Web Services (AWS) 工具的功能和益处。持续集成和持续交付是最佳实践，也是 DevOps 计划的重要组成部分。

软件交付面临的挑战

企业现今面临着快速变化的竞争格局、不断变化的安全要求和性能可扩展性的挑战。企业必须弥合运营稳定性和快速功能开发之间的差距。持续集成和持续交付 (CI/CD) 是在保持系统稳定性和安全性的同时实现快速软件更改的实践。

Amazon 早就意识到，向 Amazon.com 零售买家、Amazon 附属公司和 Amazon Web Services (AWS) 交付功能的业务需求需要全新且创新性的软件交付方式。对于像 Amazon 这样规模的公司，成千上万的独立软件团队必须能够并行工作，以便快速、安全、可靠地交付软件，并且对停机零容忍。

通过了解如何高速度交付软件，Amazon 和其他具有前瞻性思维的组织开创了 DevOps。DevOps 是文化理念、实践和工具的组合，可以提高组织高速度交付应用程序和服务的能力。使用 DevOps 原则，组织可以比使用传统软件开发和基础设施管理流程的组织更快地发展和改进产品。这种速度使组织能够更好地服务其客户，并在市场上更高效地参与竞争。

其中一些原则，例如双披萨团队和微服务/面向服务的架构 (SOA)，不在本白皮书的讨论范围之内。本白皮书讨论 Amazon 已经构建并持续改进的 CI/CD 功能。CI/CD 是快速、可靠地交付软件功能的关键。

AWS 现在将这些 CI/CD 功能作为一组开发人员服务提供：AWS CodeStar、AWS CodeCommit、AWS CodePipeline、AWS CodeBuild、AWS CodeDeploy 和 AWS CodeArtifact。实践 DevOps 的开发人员和 IT 运营专业人员可以使用这些服务快速、安全且有保障地交付软件。它们共同帮助您安全地存储版本控制并将其应用于应用程序的源代码。您可以使用 AWS CodeStar，通过这些服务快速编排端到端软件发布工作流。对于现有环境，AWS CodePipeline 可以灵活地将每项服务与现有工具独立集成。这些是高度可用且易于集成的服务，可以像任何其他 AWS 服务一样通过 AWS Management Console、AWS 应用程序编程接口 (API) 和 AWS 软件开发工具包 (SDK) 进行访问。

什么是持续集成和持续交付/部署？

本节讨论持续集成和持续交付的实践，并解释持续交付和持续部署之间的区别。

持续集成

持续集成 (CI) 是一种软件开发实践，采用持续集成时，开发人员会定期将他们的代码更改合并到一个中央存储库中，之后系统会自动运行构建和测试操作。CI 通常涉及软件发布流程的构建或集成阶段，并同时需要自动化部分（例如，CI 或构建服务）和文化部分（例如，频繁集成方面的知识）。CI 的主要目标是更快地发现并解决错误，提高软件质量，并缩短验证和发布新软件更新所需的时间。

持续集成侧重于要集成的较小提交和较小的代码更改。开发人员定期提交代码，每天至少提交一次。开发人员从代码存储库中提取代码，以确保在推送到编译服务器之前合并本地主机上的代码。在此阶段，编译服务器将运行各种测试，并接受或拒绝代码提交。

实施 CI 的基本挑战包括更频繁地提交到公共代码库、维护单一源代码存储库、自动执行构建以及自动执行测试。其他挑战包括：在与生产环境类似的环境中进行测试，向团队提供过程的可见性，以及允许开发人员轻松获得应用程序的任何版本。

持续交付和部署

持续交付 (CD) 是一种软件开发实践，通过持续交付，系统可以自动构建和测试代码更改，并为生产发布做好准备。它可以在持续集成的基础之上进行扩展，也即，在完成构建阶段后，将所有代码更改都部署到测试环境和/或生产环境中。持续交付可以通过工作流程实现完全自动化，也可以在关键点通过手动步骤实现部分自动化。当持续交付得以正确实施时，开发人员将始终能够获得一个已通过标准化测试流程的部署就绪型构建构件。

采用持续部署时，系统会在未经开发人员明确批准的情况下自动将修订部署到生产环境中，从而实现整个软件发布流程的自动化。这反过来又允许在产品生命周期的早期建立持续的客户反馈循环。

持续交付不是持续部署

关于持续交付的一个误解是，它意味着提交的每一项更改都会在通过自动化测试后立即应用于生产。但是，持续交付的目的不是立即将所有更改应用到生产中，而是要确保每个更改都已准备好投入生产。

在将更改部署到生产环境之前，您可以实施决策流程，以确保对生产部署进行授权和审计。这个决定可以由人做出，然后由工具执行。

使用持续交付，投入生产的决定变成了业务决策，而不是技术决策。每次提交时都会进行技术验证。

向生产环境推出更改不是破坏性事件。部署不要求技术团队停止处理下一组更改，也不需要项目计划、移交文档或维护窗口。部署成为一个可重复的过程，而这一过程已经在测试环境中得以多次执行和验证。

持续交付的益处

CD 为您的软件开发团队提供了众多益处，包括自动执行流程、提高开发人员工作效率、提升代码质量以及更快地向客户交付更新。

自动执行软件发布流程

CD 为您的团队提供了一种签入代码的方法，代码经自动构建、测试和准备以供发布到生产环境，这使您的软件交付高效、灵活、快速且安全。

提高开发人员工作效率

CD 实践可让开发人员摆脱手动任务，清理复杂的依赖关系，并将重点放在交付软件的新功能方面，从而帮助您的团队提高工作效率。开发人员可以专注于可交付所需功能的编码逻辑，而不是将其代码与业务的其他部分集成以及花时间来研究如何将此代码部署到平台上。

提升代码质量

CD 可以帮助您在交付过程及早发现和纠正错误，以免它们在以后变成更大的问题。您的团队可以轻松地执行其他类型的代码测试，因为整个过程已实现自动化。通过更频繁地进行更多测试，团队可以更快地进行迭代，并获得有关更改所带来的影响的即时反馈。这使团队能够在高度保证稳定性和安全性的情况下，推动获得高质量的代码。开发人员将通过即时反馈了解新代码是否有效，以及是否引入了任何重大更改或错误。在开发过程早期发现的错误最容易修复。

更快交付更新

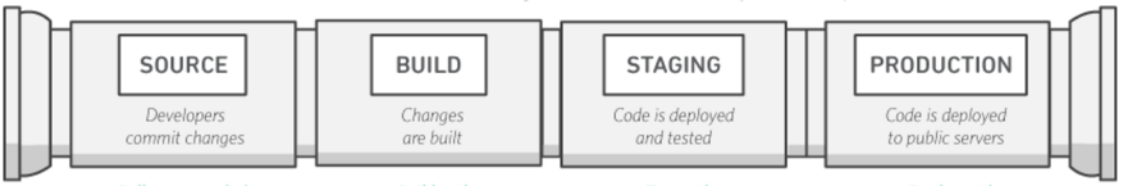
CD 可帮助您的团队快速、频繁地向客户交付更新。实施 CI/CD 后，整个团队（包括功能发布和错误修复）的速度都会提高。企业可以更快地应对市场变化、安全挑战、客户需求和成本压力。例如，如果需要一个新的安全功能，您的团队可以通过自动化测试来实现 CI/CD，以便满怀信心地将修复程序快速可靠地引入生产系统。过去需要数周甚至数月的时间，现在几天甚至几小时就可以完成。

实施持续集成和持续交付

本节讨论可以着手在组织中实施 CI/CD 模型的方法。本白皮书不讨论拥有成熟 DevOps 和云转型模式的组织如何构建和使用 CI/CD 管道。为了帮助您踏上 DevOps 之旅，AWS 拥有许多经认证的 DevOps 合作伙伴，它们可提供资源和工具。有关准备迁移到 AWS 云的更多信息，请参阅构建云运营模型。

持续集成/持续交付的途径

可以将 CI/CD 描绘成一个管道（请参阅下图），其中新代码在一端提交，在一系列阶段（源代码、构建、暂存和生产）中进行测试，然后作为生产就绪代码发布。如果您的组织不熟悉 CI/CD，则可以通过迭代方式处理此管道。这意味着您应该从小处着手，在每个阶段进行迭代，以便您能够以可帮助组织发展壮大的方式理解和开发代码。



CI/CD 管道

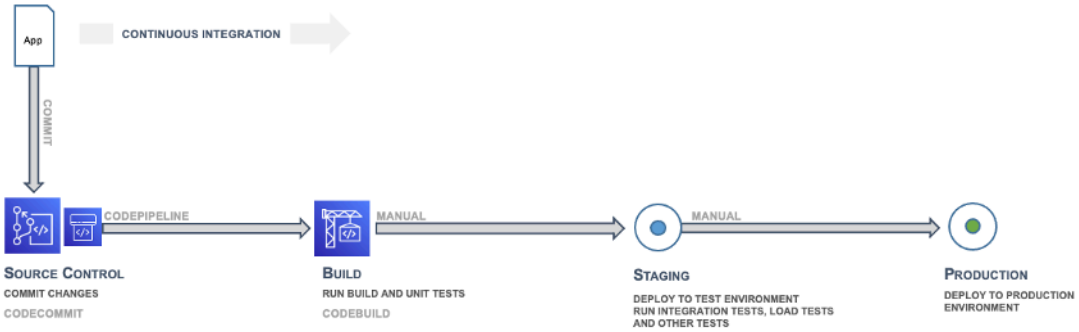
CI/CD 管道的每个阶段都被构造为交付过程中的一个逻辑单元。此外，每个阶段都充当审查代码某个方面的一道关卡。随着代码在管道中推进，假设代码的质量在后面的阶段会更高（因为代码的更多方面持续得到验证）。在早期阶段发现的问题会阻止代码在管道中推进。测试的结果会立即发送给团队，如果软件没有通过该阶段，所有后续的构建和发布工作都会停止。

这些阶段是建议。您可以根据业务需求调整各个阶段。对于多种类型的测试、安全性和性能，可以重复某些阶段。根据项目的复杂性和团队的结构，某些阶段可以在不同的级别重复多次。例如，一个团队的最终产品可以成为下一个团队的项目中的依赖项。这意味着第一个团队的最终产品随后将作为下一个团队的项目中的构件。

CI/CD 管道的存在将对组织能力的成熟过程产生重大影响。组织应该从小步骤开始，而不是试图建立一个完全成熟的管道（即，一开始就有多个环境、多个测试阶段并在所有阶段都实现自动化）。请记住，即使是拥有高度成熟的 CI/CD 环境的组织，仍需要不断地改进其管道。

建立一个支持 CI/CD 的组织是一个过程，在这一过程中有许多目的地。下一节将讨论您的组织可以采取的一种可能途径，从持续集成开始直至持续交付的各个层面。

持续集成



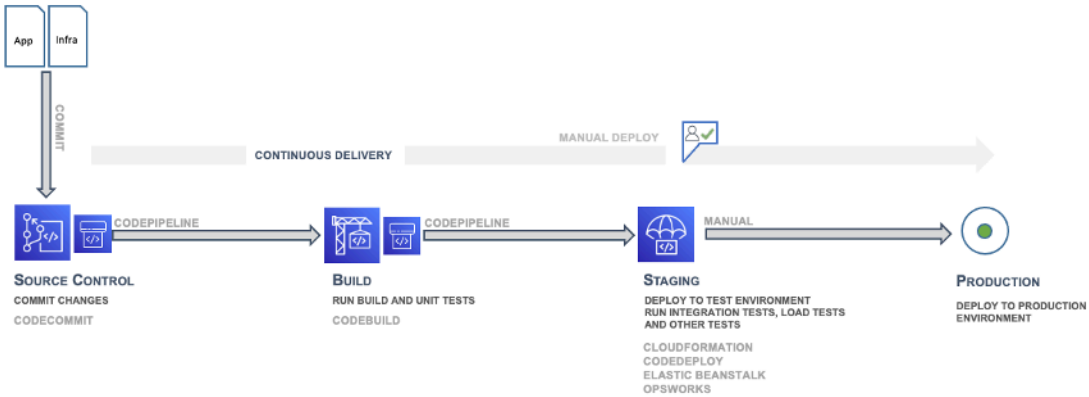
持续集成 — 源代码和构建

CI/CD 过程的第一阶段是在持续集成中发展成熟度。您应确保所有开发人员定期将其代码提交到中央存储库（例如托管在 CodeCommit 或 GitHub 中的存储库），并将所有更改合并到应用程序的发布分支中。任何开发人员都不应孤立地持有代码。如果在一段时间内需要功能分支，则应通过尽可能频繁地从上游合并以使其保持最新状态。建议团队经常提交内容并与完整的工作单元进行合并，此做法应成为团队的纪律并受到流程鼓励。早期且经常合并代码的开发人员在将来可能会遇到较少的集成问题。

还应该鼓励开发人员尽早为其应用程序创建单元测试，并在将代码推送到中央存储库之前运行这些测试。在软件开发过程早期发现的错误最容易修复，且修复成本最低。

将代码推送到源代码存储库中的分支时，监控该分支的工作流引擎会向构建器工具发送命令，以在受控环境中构建代码并运行单元测试。应适当调整构建过程的规模以处理所有活动，包括提交阶段可能发生的推送和测试，以便快速获得反馈。其他质量检查（例如单元测试覆盖率、样式检查和静态分析）也可以在此阶段进行。最后，构建器工具会为应用程序创建一个或多个二进制版本和其他构件，例如图像、样式表和文档。

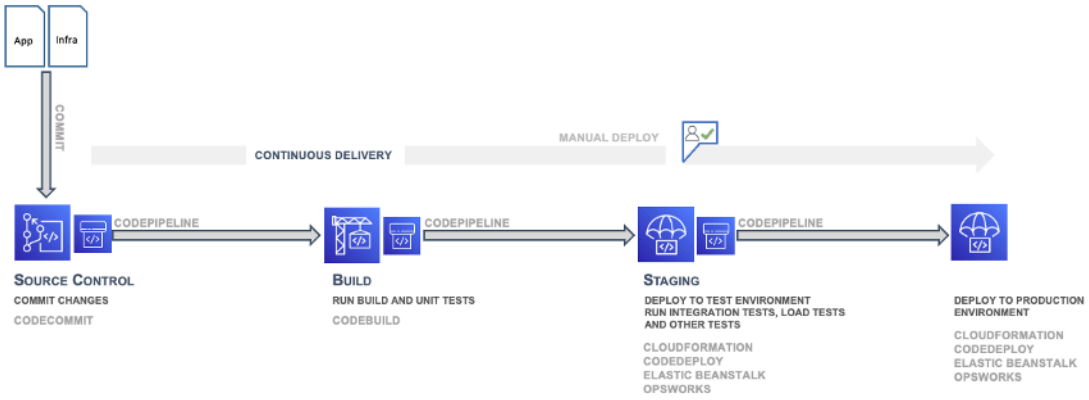
持续交付：创建暂存环境



持续交付 — 暂存

持续交付 (CD) 是下一阶段，它需要在作为生产堆栈副本的暂存环境中部署应用程序代码，并运行更多功能测试。暂存环境可以是为进行测试而预构建的静态环境，或者您可以使用已提交的基础设施和配置代码来预置和配置动态环境，以测试和部署应用程序代码。

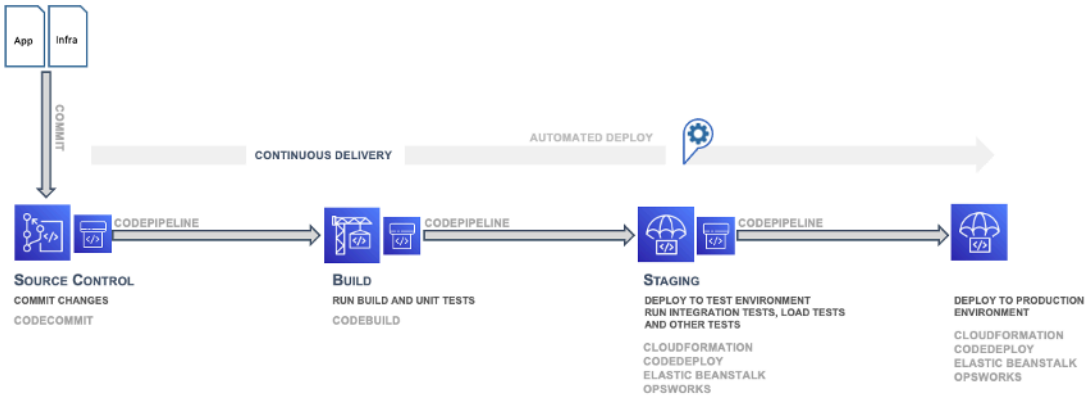
持续交付：创建生产环境



持续交付 — 生产

在部署/交付管道序列中，暂存环境之后是生产环境，该环境也是使用基础设施即代码 (IaC) 构建的。

持续部署



持续部署

CI/CD 部署管道的最后一个阶段是持续部署，这可能包括整个软件发布流程（包括部署到生产环境）的完全自动化。在完全成熟的 CI/CD 环境中，通往生产环境的路径是完全自动化的，这样就可以信心十足地部署代码。

成熟度及更多

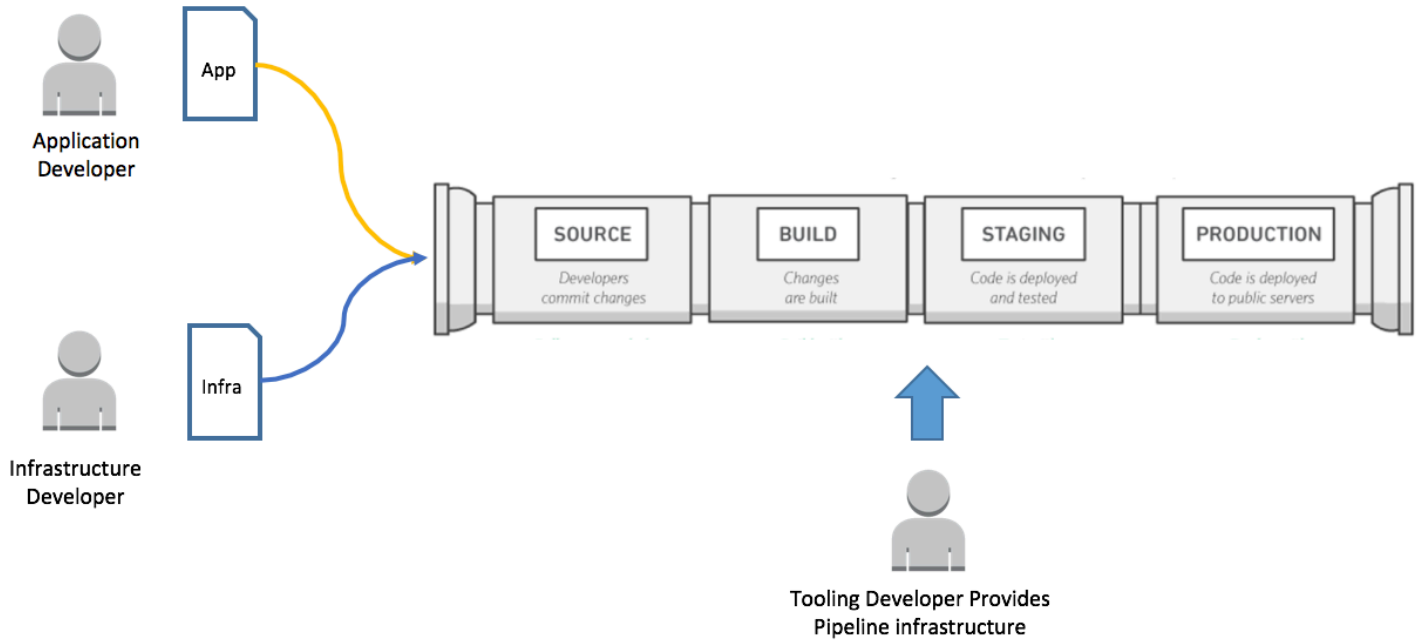
随着组织日渐成熟，它将继续发展 CI/CD 模式，以包括以下更多改进功能：

- 提供更多暂存环境，以进行特定的性能、合规性、安全性和用户界面 (UI) 测试
- 基础设施和配置代码以及应用程序代码的单元测试
- 与其他系统和流程（如代码审查、问题跟踪和事件通知）集成
- 与数据库架构迁移集成（如果适用）
- 审计和业务批准的其他步骤

即使是拥有复杂的多环境 CI/CD 管道的最成熟组织，也在继续寻求改进。DevOps 是一个过程，而不是一个终点。通过开发团队不同部门之间的协作，不断收集有关管道的反馈，并实现速度、规模、安全性和可靠性方面的改进。

团队

AWS 建议组成三个开发人员团队来实施 CI/CD 环境：应用程序团队、基础设施团队和工具团队（请参见下图）。该组织代表了一套最佳实践，这些实践已在快速发展的初创公司、大型企业组织和 Amazon 本身中得以开发和应用。团队规模应不超过两个比萨饼可以承受的人数，或大约 10-12 人。这遵循了一条沟通规则，即随着群体规模的增加和沟通渠道的增多，有意义的对话会达到极限。



应用程序、基础设施和工具团队

应用程序团队

应用程序团队创建应用程序。应用程序开发人员拥有待办事项、情节和单元测试，并且根据指定的应用程序目标开发功能。该团队的组织目标是最大限度地减少这些开发人员花在非核心应用程序任务上的时间。

除了具备应用程序语言方面的功能编程技能外，应用程序团队还应具备平台技能并了解系统配置。这将使他们能够专注于开发功能和强化应用程序。

基础设施团队

基础设施团队负责编写代码，以同时创建和配置运行应用程序所需的基础设施。该团队可能使用原生 AWS 工具（例如 AWS CloudFormation）或通用工具，例如 Chef、Puppet 或 Ansible。基础设施团队负责指定所需的资源，并与应用程序团队密切合作。对于一个小型应用程序，基础设施团队可能只由一两个人组成。

团队应该具备基础设施预置方法方面的技能，例如 AWS CloudFormation 或 HashiCorp Terraform。团队还应使用 Chef、Ansible、Puppet 或 Salt 等工具来培养配置自动化技能。

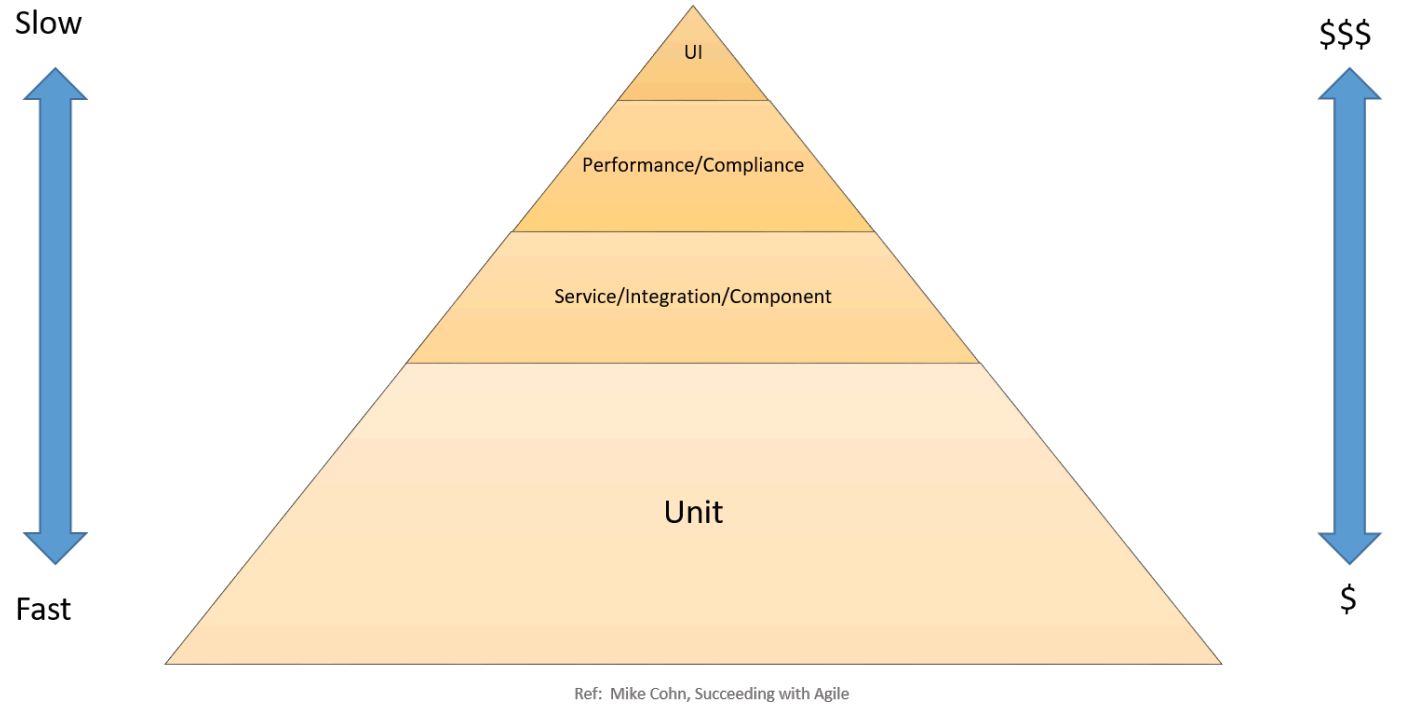
工具团队

工具团队负责构建和管理 CI/CD 管道。他们负责构成管道的基础设施和工具。他们不是双比萨团队的成员；但是，他们创建的工具可供组织中的应用程序团队和基础设施团队使用。组织需要不断完善其工具团队，以便工具团队比日趋成熟的应用程序团队和基础设施团队领先一步。

工具团队必须能够熟练地构建和集成 CI/CD 管道的所有部分。这包括构建源代码管理存储库、工作流引擎、构建环境、测试框架和构件存储库。该团队可以选择实施 AWS CodeStar、AWS CodePipeline、AWS CodeCommit、AWS CodeDeploy、AWS CodeBuild 和 AWS CodeArtifact 等软件以及 Jenkins、GitHub、Artifactory、TeamCity 和其他类似工具。一些组织可能将其称为 DevOps 团队，但 AWS 不鼓励这样做，而是鼓励将 DevOps 视为软件交付中人员、流程和工具的总和。

持续集成和持续交付的测试阶段

这三个 CI/CD 团队应在 CI/CD 管道的不同阶段将测试纳入软件开发生命周期。总体而言，应尽早开始测试。以下测试金字塔是迈克·科恩 (Mike Cohn) 在《成功与敏捷》 中提供的概念。它显示了各种软件测试以及相关的成本和运行速度。



CI/CD 测试金字塔

单元测试位于金字塔的底部。它们既是运行速度最快的，也是成本最低的。因此，单元测试应构成测试策略的重要部分。一个好的经验法则是大约 70%。单元测试应该具有近乎完整的代码覆盖率，因为在此阶段中找到的错误可以快速、低成本修复。

服务、组件和集成测试位于金字塔上单元测试的上方。这些测试需要详细的环境，因此满足基础设施要求的成本更高，运行速度更慢。性能和合规性测试是下一个级别。它们需要生产质量的环境，而且成本更高。用户界面和用户验收测试处于金字塔的顶端，也需要生产质量的环境。

所有这些测试都是确保高质量软件的完整策略的一部分。但是，为了加快开发速度，重点是金字塔下半部分的测试数量和覆盖范围。

以下各节将讨论 CI/CD 阶段。

设置源代码

在项目开始时，务必设置可以存储原始代码以及配置和架构更改的源代码。在源代码阶段，选择源代码存储库，例如托管在 GitHub 或 AWS CodeCommit 中的源代码存储库。

设置和运行构建过程

构建过程自动化对于 CI 流程至关重要。设置构建过程自动化时，第一项任务是选择正确的构建工具。有许多构建工具，例如：

- Ant、Maven 和 Gradle for Java
- Make for C/C++
- Grunt for JavaScript
- Rake for Ruby

最适合您的构建工具取决于项目的编程语言和团队的技能集。选择构建工具后，需要在构建脚本中明确定义所有依赖项以及构建步骤。对最终的构建构件进行版本化也是一种最佳实践，这样可以更轻松地部署和跟踪问题。

构建

在构建阶段，构建工具会将对源代码存储库的任何更改作为输入，构建软件，并运行以下类型的测试：

单元测试 – 测试代码的特定部分，以确保代码执行预期的功能。单元测试由软件开发人员在开发阶段执行。在此阶段，可以应用静态代码分析、数据流分析、代码覆盖率和其他软件验证过程。

静态代码分析 – 此测试是在构建和单元测试后不实际执行应用程序的情况下执行的。这种分析可以帮助发现编码错误和安全漏洞，还可以确保符合编码准则。

暂存

在暂存阶段，将创建镜像最终生产环境的完整环境。将执行以下测试：

集成测试 – 根据软件设计验证组件之间的接口。集成测试是一个迭代过程，有助于构建稳健的接口和促进系统完整性。

组件测试 – 测试在不同组件之间传递的消息及其结果。该测试的一个关键目标可能是组件测试的幂等性。测试可能包括极大的数据量、边缘情况和异常输入。

系统测试 – 端到端测试系统并验证软件是否满足业务需求。这可能包括测试用户界面 (UI)、API、后端逻辑和结束状态。

性能测试 – 确定系统在特定工作负载下执行时的响应能力和稳定性。性能测试还用于调查、测量、确认或验证系统的其他质量属性，例如可扩展性、可靠性和资源使用情况。性能测试的类型可能包括负载测试、压力测试和峰值测试。性能测试用于根据预定义的标准进行基准测试。

合规性测试 – 检查代码更改是否符合非功能规范和/或法规的要求。它确定您是否正在实施且满足定义的标准。

用户验收测试 – 验证端到端业务流程。此测试由终端用户在暂存环境中执行，并确认系统是否满足要求规范的要求。通常，客户在此阶段会采用 Alpha 和 Beta 测试方法。

生产

最后，在通过之前的测试后，将在生产环境中重复暂存阶段。在此阶段，可以通过在将新代码部署到整个生产环境之前，将新代码部署到一小部分服务器甚至一台服务器上，或部署到一个 AWS 区域中，以完成最终 Canary 测试。部署方法部分介绍了如何安全地部署到生产环境的具体细节。

下一节将讨论构建管道以合并这些阶段和测试。

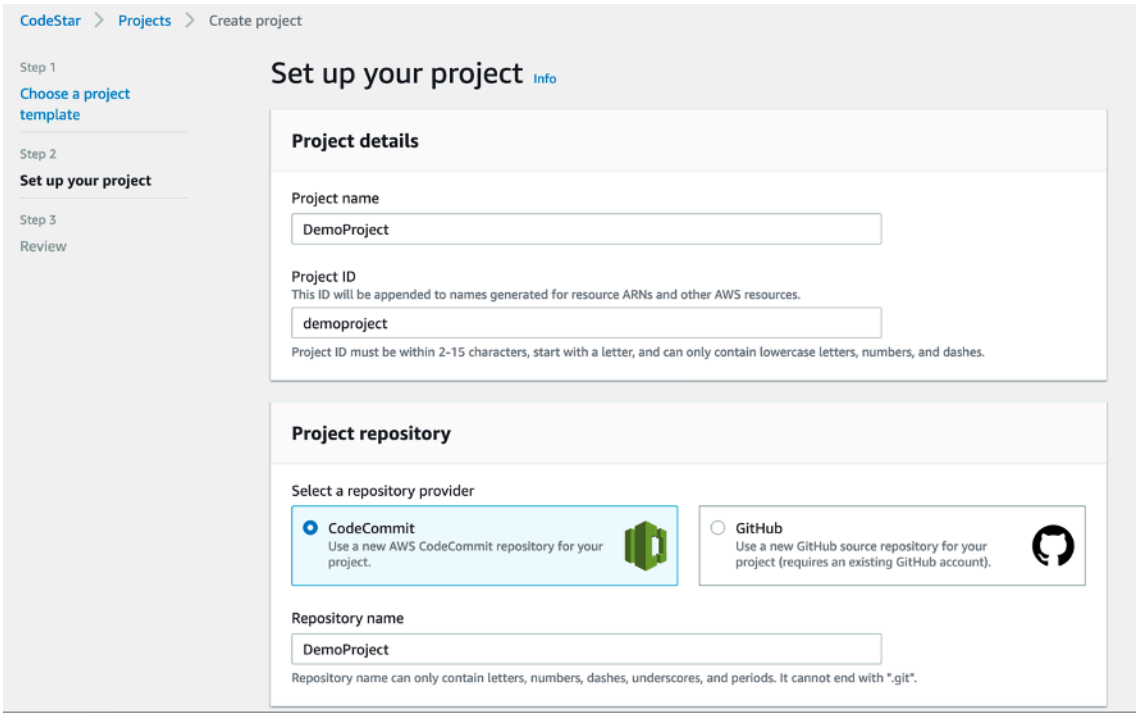
构建管道

本节讨论构建管道。首先建立一个仅包含 CI 所需组件的管道，之后过渡到包含更多组件和阶段的持续交付管道。本节还讨论如何考虑对大型项目使用 AWS Lambda 函数和手动批准，以及如何针对多个团队、分支和 AWS 区域进行规划。

从最简可行管道开始，以实现持续集成

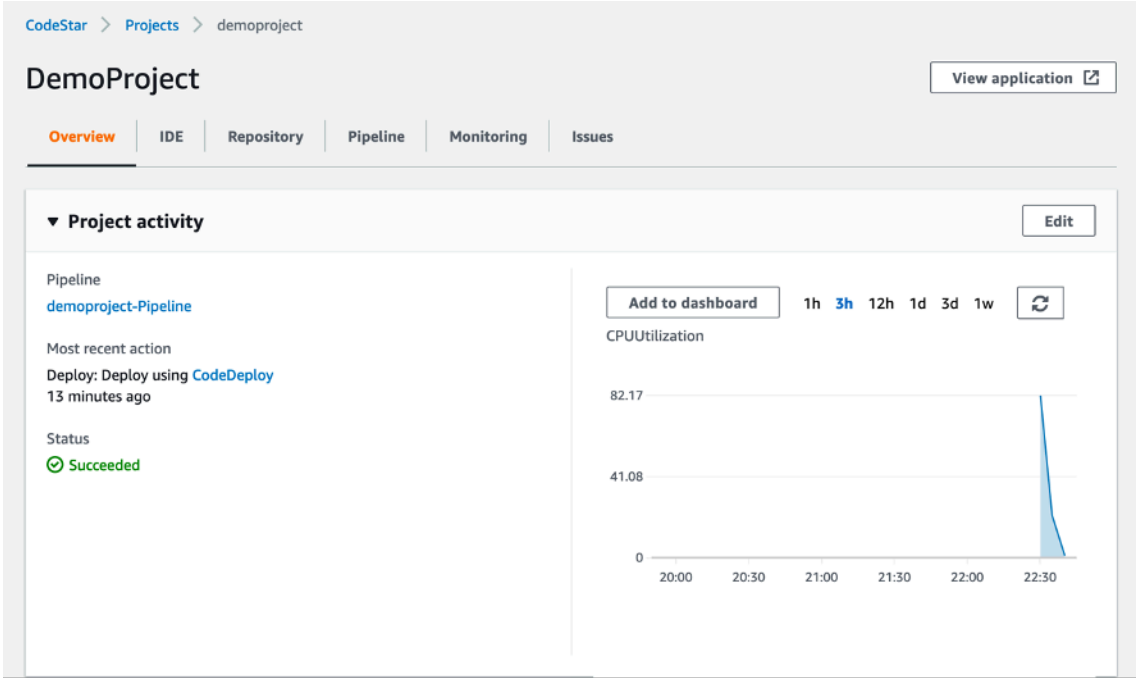
您的组织实现持续交付的旅程始于最简可行管道 (MVP)。正如实施持续集成和持续交付中所讨论，团队可以从一个非常简单的流程开始，例如实施一个管道，以执行代码样式检查或在不进行部署的情况下执行单个单元测试。

一个关键组件是持续交付编排工具。为了帮助您构建此管道，Amazon 开发了 AWS CodeStar。



AWS CodeStar 设置页面

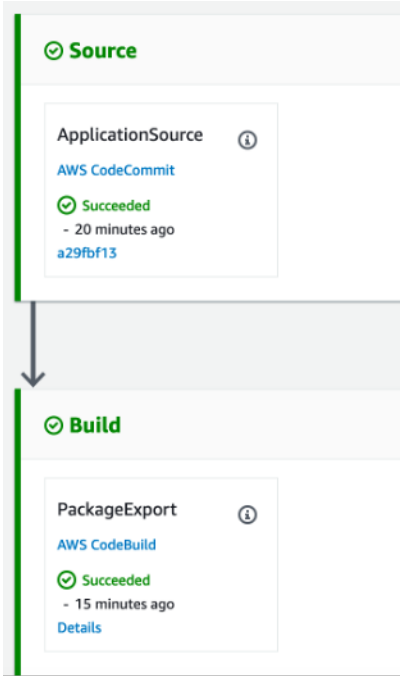
AWS CodeStar 将 AWS CodePipeline、AWS CodeBuild、AWS CodeCommit 和 AWS CodeDeploy 与集成的设置流程、工具、模板和控制面板结合使用。AWS CodeStar 提供了您在 AWS 上快速开发、构建和部署应用程序所需的一切。这让您可以更快地开始发布代码。已经熟悉 AWS Management Console并寻求更高级别控制的客户可以手动配置他们选择的开发工具，并可以根据需要预置各项 AWS 服务。



AWS CodeStar 控制面板

AWS CodePipeline 是一项 CI/CD 服务，可通过 AWS CodeStar 或通过 AWS Management Console使用，以进行快速、可靠的应用程序和基础设施更新。在每次代码发生更改时，AWS CodePipeline 都会根据您定义的发布流程模型构建、测试和部署代码。这使您能够快速而可靠地提供各种功能和更新。您可使用我们提供的针对常用第三方服务（如 GitHub）的预先构建的插件，或通过将您的自定义插件集成到您发布流程中的任何阶段，来轻松构建端到端解决方案。使用 AWS CodePipeline，您只需按实际使用量付费。无需预付费用或长期承诺。

AWS CodeStar 和 AWS CodePipeline 的步骤直接映射到源代码、构建、暂存和生产 CI/CD 阶段。虽然需要持续交付，但您可以从一个简单的两步管道开始，此管道检查源代码库并执行构建操作：

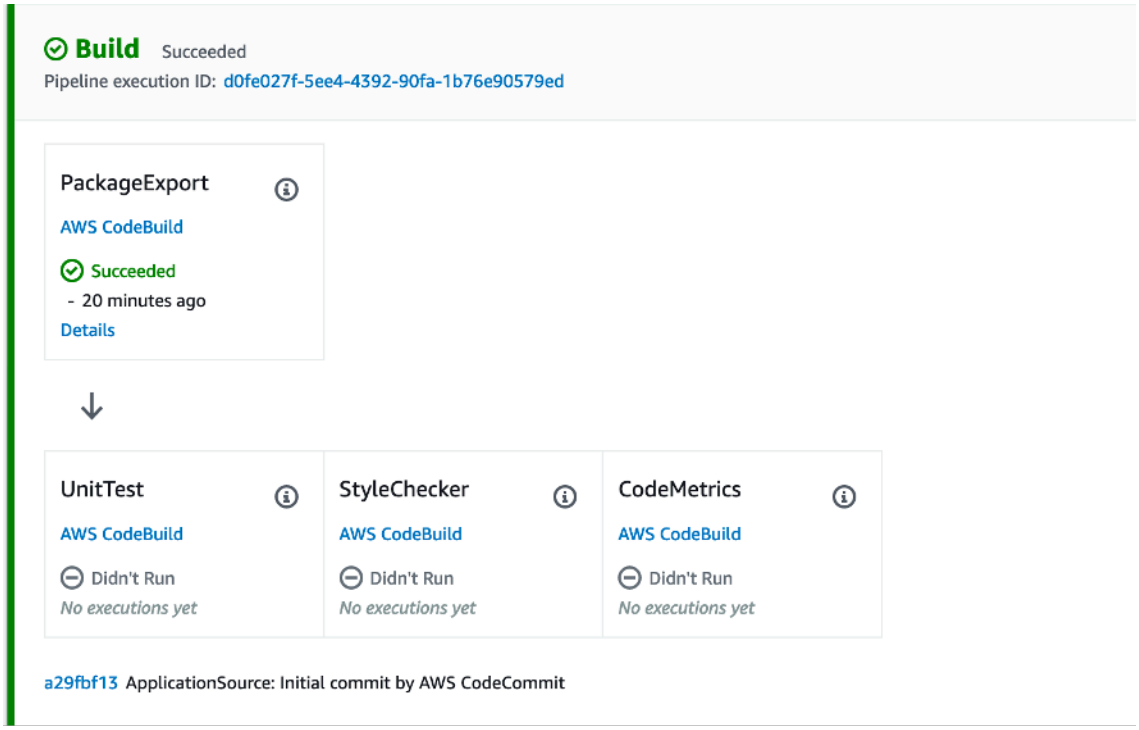


AWS CodePipeline — 源代码和构建阶段

对于 AWS CodePipeline，源代码阶段可以接受来自 GitHub、AWS CodeCommit 和 Amazon Simple Storage Service (Amazon S3) 的输入。自动执行构建流程是实施持续交付并迈向持续部署的关键性的第一步。消除人工参与生成构建构件过程可以减轻团队的负担，最大限度地减少因手动打包而引入的错误，并允许您更频繁地开始打包可使用的构件。

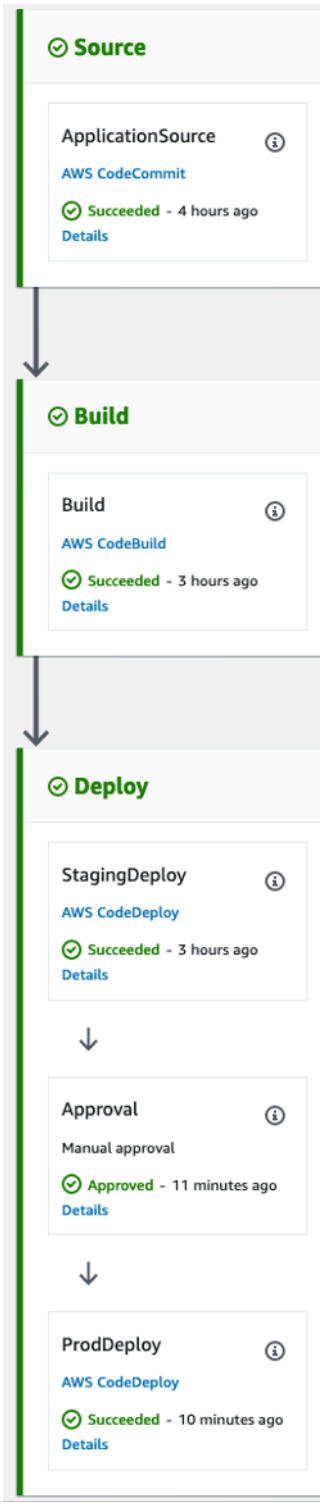
AWS CodePipeline 与 AWS CodeBuild 这一完全托管式构建服务无缝协作，以便更轻松地在管道中设置用于打包代码和运行单元测试的构建步骤。使用 AWS CodeBuild，您无需预置、管理或扩展自己的编译服务器。AWS CodeBuild 可以持续扩展并同时处理多项构建任务，因此您的构建任务不会在队列中等待。AWS CodePipeline 还与 Jenkins、Solano CI 和 TeamCity 等编译服务器集成。

例如，在接下来的构建阶段，三个操作（单元测试、代码样式检查和代码指标收集）并行运行。使用 AWS CodeBuild，可以将这些步骤添加为新项目，而无需进一步构建或安装编译服务器来处理负载。



AWS CodePipeline – 构建功能

图（AWS CodePipeline — 源代码和构建阶段）中所示的源代码和构建阶段以及支持流程和自动化可为团队向持续集成过渡提供支持。在这个成熟度级别上，开发人员需要定期关注构建和测试结果。他们还需要培育和维护一个正常运行的单元测试基地。这反过来又会增强整个团队对 CI/CD 管道的信心，并进一步推动对它的采用。



AWS CodePipeline 阶段

持续交付管道

实施持续集成管道并建立支持流程后，您的团队就可以开始向持续交付管道过渡。这种过渡要求团队自动执行构建和部署应用程序的过程。

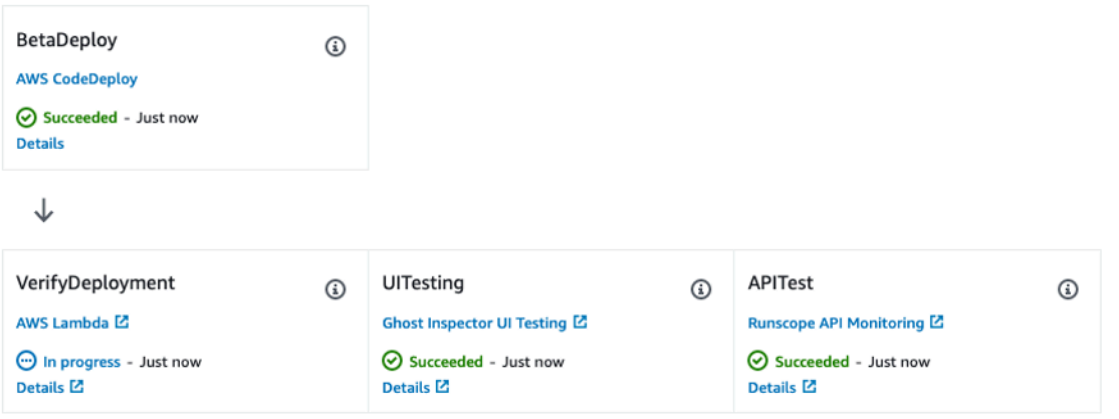
持续交付管道的特点是存在暂存和生产步骤，其中生产步骤是在人工批准后执行的。

与构建持续集成管道的方式相同，您的团队可以通过编写部署脚本逐步开始构建持续交付管道。

根据应用程序的需要，某些部署步骤可以由现有 AWS 服务抽象出来。例如，AWS CodePipeline 直接与以下服务集成：AWS CodeDeploy（自动将代码部署到 Amazon EC2 实例和本地运行的实例的服务）；AWS OpsWorks（可帮助您使用 Chef 操作应用程序的配置管理服务）；以及 AWS Elastic Beanstalk（用于部署和扩展 Web 应用程序和服务的服务）。

AWS 提供了详细的文档来介绍如何实施 AWS CodeDeploy 并将其与基础设施和管道集成。

在团队成功实现了应用程序部署自动化之后，可以通过各种测试来扩展部署阶段。例如，您可以添加其他与 Ghost Inspector、Runscope 等服务的开箱即用集成，如下图所示。



AWS CodePipeline – 部署阶段的代码测试

添加 Lambda 操作

AWS CodeStar 和 AWS CodePipeline 支持与 AWS Lambda 集成。这种集成支持实施一系列广泛的任务，例如在环境中创建自定义资源、与第三方系统（如 Slack）集成，以及对新部署的环境执行检查。

可以在 CI/CD 管道中使用 Lambda 函数来执行以下任务：

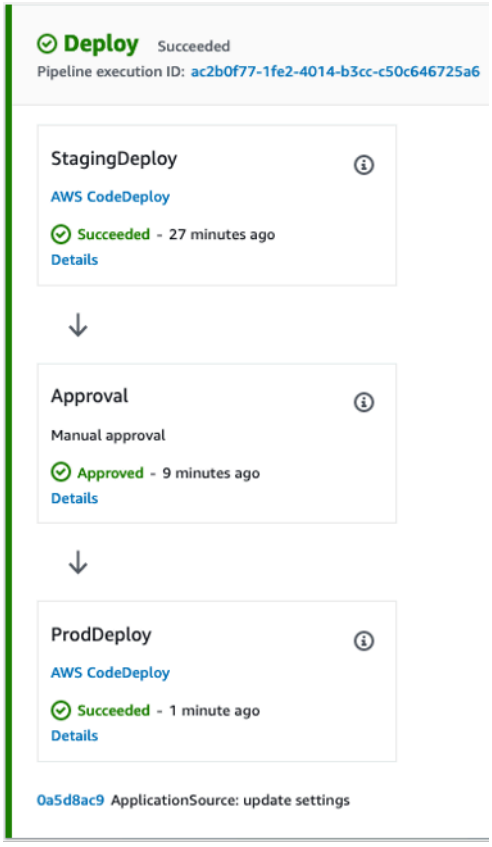
- 通过应用或更新 AWS CloudFormation 模板来实施对环境所做的更改。

- 使用 AWS CloudFormation 在管道的一个阶段按需创建资源，并在另一个阶段将其删除。
- 使用将交换规范名称记录 (CNAME) 值的 Lambda 函数，在 AWS Elastic Beanstalk 中部署零停机时间的应用程序版本。
- 部署到 Amazon Elastic Container Service (ECS) Docker 实例。
- 通过创建 AMI 快照在构建或部署之前备份资源。
- 将与第三方产品的集成添加到您的管道中，例如将消息发布到 Internet Relay Chat (IRC) 客户端。

手动批准

您可以在管道内的某个阶段中您希望管道处理停止的位置添加批准操作，以便拥有所需 AWS Identity and Access Management (IAM) 权限的人可以批准或拒绝该操作。

如果操作获得批准，管道处理将恢复。如果该操作被拒绝，或者没有人在管道到达该操作并停止的七天内批准或拒绝该操作，结果将与操作失败的结果相同，并且管道处理不会继续。



AWS CodeDeploy— 手动批准

在 CI/CD 管道中部署基础设施代码更改

AWS CodePipeline 可让您在管道的任何阶段选择 AWS CloudFormation 作为部署操作。然后，您可以选择您希望 AWS CloudFormation 执行的特定操作，例如创建或删除堆栈以及创建或执行更改集。堆栈是一个 AWS CloudFormation 概念，代表一组相关的 AWS 资源。虽然有许多方法可以预置基础设施即代码，但 AWS CloudFormation 是 AWS 推荐的综合工具，它是一种可扩展、完整的解决方案，可以将最全面的 AWS 资源集描述为代码。AWS 建议在 AWS CodePipeline 项目中使用 AWS CloudFormation，以跟踪基础设施更改和测试。

适用于无服务器应用程序的 CI/CD

还可以使用 AWS CodeStar、AWS CodePipeline、AWS CodeBuild 和 AWS CloudFormation 为无服务器应用程序构建 CI/CD 管道。无服务器应用程序将 Amazon Cognito、Amazon S3 和 Amazon DynamoDB 等托管式服务与事件驱动型服务以及 AWS Lambda 集成，以不需要管理服务器的方式部署应用程序。如果您是无服务器应用程序开发人员，则可以使用 AWS CodePipeline、AWS CodeBuild 和 AWS CloudFormation 的组合来自动构建、测试和部署在使用 AWS Serverless Application Model 构建的模板中表示的无服务器应用程序。有关更多信息，请参阅自动部署基于 Lambda 的应用程序的 AWS Lambda 文档。

还可以使用 AWS Serverless Application Model Pipelines (AWS SAM Pipelines) 创建遵循组织最佳实践的安全 CI/CD 管道。AWS SAM Pipelines 是 AWS SAM CLI 的一项新功能，能够让您在几分钟内获得 CI/CD 的益处，例如加快部署频率、缩短实施更改所需的时间以及减少部署错误。AWS SAM Pipelines 附带了一组适用于 AWS CodeBuild/CodePipeline 的原定设置管道模板，这些模板遵循 AWS 部署最佳实践。有关更多信息并要查看教程，请参阅博客 AWS SAM Pipelines 简介。

适用于多个团队、分支和 AWS 区域的管道

对于大型项目，多个项目团队处理不同组件的情况并不少见。如果多个团队使用单个代码存储库，则可以对其进行映射，以便每个团队都有其自己的分支。还应该有一个集成或发布分支进行项目的最终合并。如果使用面向服务的架构或微服务架构，则每个团队都可以拥有自己的代码存储库。

在第一种情况下，如果使用单个管道，则一个团队可能会通过阻塞管道来影响其他团队的进度。AWS 建议您为团队分支创建特定的管道，并为最终产品交付创建另一个发布管道。

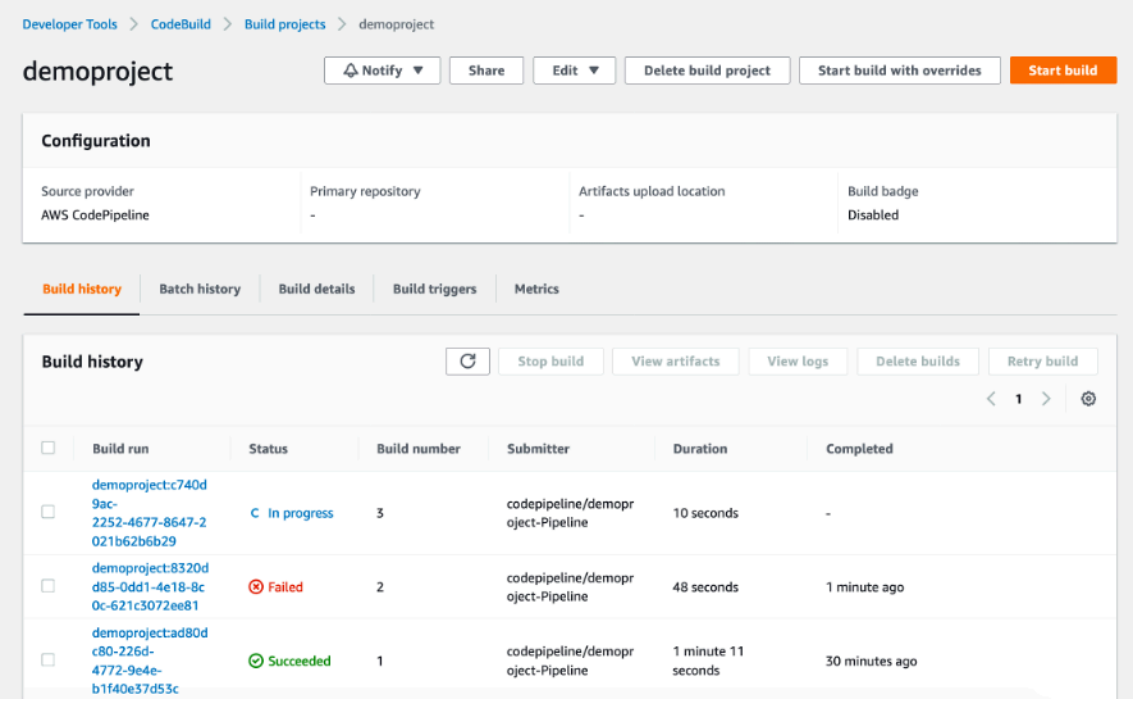
管道与 AWS CodeBuild 的集成

AWS CodeBuild 旨在使您的组织能够构建具有几乎无限规模的高可用性构建流程。AWS CodeBuild 为多种常用语言提供了快速入门环境，并能够运行您指定的任何 Docker 容器。

凭借与 AWS CodeCommit、AWS CodePipeline 和 AWS CodeDeploy 紧密集成的优势以及 Git 和 CodePipeline Lambda 操作，CodeBuild 工具非常灵活。

可以通过包含一个 `buildspec.yml` 文件来构建软件，该文件标识每个构建步骤，包括构建前和构建后操作或通过 CodeBuild 工具指定的操作。

可以使用 CodeBuild 控制面板查看每个构建操作的详细历史记录。事件以 Amazon CloudWatch Logs 日志文件的形式存储。



AWS CodeBuild 中的 CloudWatch Logs 日志文件

管道与 Jenkins 的集成

可以使用 Jenkins 构建工具创建交付管道。这些管道使用标准任务，而这些任务定义用于实施持续交付阶段的步骤。但是，对于较大的项目，这种方法可能不是最佳选择，因为管道的当前状态不会在 Jenkins 重启之间持续存在，实现手动批准并不简单，而且跟踪复杂管道的状态可能很复杂。

相反，AWS 建议您使用 AWS Code Pipeline 插件通过 Jenkins 实现持续交付。该插件允许使用类似 Groovy 的域特定语言来描述复杂的工作流，并可用于编排复杂的管道。AWS Code Pipeline 插件的功能可以通过使用附属插件来增强，例如 Pipeline Stage View 插件（用于直观展示在管道中定义的阶段的当前进度）或 Pipeline Multibranch 插件（对来自不同分支的构建进行分组）。

AWS 建议您将管道配置存储在 Jenkinsfile 中，并将其签入到源代码存储库中。这样，就可以跟踪对管道代码的更改，在使用 Pipeline Multibranch 插件时这会变得更加重要。AWS 还建议您将管道划分为

多个阶段。这会对管道步骤进行逻辑分组，并使 Pipeline Stage View 插件能够直观展示管道的当前状态。

下图显示一个 Jenkins 管道示例，其中包含由 Pipeline Stage View 插件直观展示的四个已定义阶段。

由 Pipeline Stage View 插件直观展示的 Jenkins 管道的四个已定义阶段

部署方法

在持续交付过程中推出新版软件时，您可以考虑多种部署策略和变体。本节讨论最常见的部署方法：一次部署全部（就地部署）、滚动、不可改变和蓝/绿。AWS 会指出 AWS CodeDeploy 和 AWS Elastic Beanstalk 支持其中哪些方法。

下表总结了每种部署方法的特征。

方法	部署失败带来的影响	部署时间	零停机时间	无 DNS 更改	回滚过程	代码部署到
就地部署	停机时间	🕒	×	✓	重新部署	现有实例
滚动	单个批处理服务中断。任何在故障之前成功的批处理将运行新应用程序版本。	🕒 🕒 †	✓	✓	重新部署	现有实例
附加批处理滚动部署（beanstalk）	如果第一个批处理失败，则影响最小；否则类似于滚动部署。	🕒 🕒 🕒 †	✓	✓	重新部署	新实例和现有实例
不可改变	最低	🕒 🕒 🕒 🕒	✓	✓	重新部署	新实例
流量拆分	最低	🕒 🕒	✓	✓	重新路由流量并终止新实例	新实例

方法	部署失败带来的影响	部署时间	零停机时间	无 DNS 更改	回滚过程	代码部署到
		🕒 🕒				
蓝/绿	最低	🕒 🕒 🕒 🕒	✓	X	切换回旧环境	新实例

一次部署全部（就地部署）

一次部署全部（就地部署）是一种可用于将新的应用程序代码部署到现有服务器机群的方法。此方法在一个部署操作中替换所有代码。它要求停机，因为机群中的所有服务器都会一起更新。无需更新现有的 DNS 记录。如果部署失败，恢复运营的唯一方法是再次在所有服务器上重新部署代码。

在 AWS Elastic Beanstalk 中，此部署称为一次部署全部，可用于单个负载均衡的应用程序。在 AWS CodeDeploy 中，此部署方法称为部署配置 `AllAtOnce` 的就地部署。

滚动部署

通过滚动部署，机群分成若干部分，这样就不会一次升级整个机群。在部署过程中，两个软件版本（新版本和旧版本）在同一个机群上运行。此方法可实现零停机时间更新。如果部署失败，则只有机群的已更新部分受到影响。

滚动部署方法的一种变体称为金丝雀发布 (Canary release)，首先是在极小比例的服务器上部署新的软件版本。这样，您就可以在少数服务器上观察软件在生产环境中的运行情况，同时最大限度地减少重大更改带来的影响。如果 Canary 版本部署的错误率提高，则会回滚软件。否则，使用新版本的服务器的百分比将逐渐增加。

AWS Elastic Beanstalk 已遵循滚动部署模式，并提供两个部署选项：滚动部署和附加批处理滚动部署。这些选项允许应用程序在服务器停止服务之前先纵向扩展，从而在部署期间保留完整的功能。AWS CodeDeploy 通过使用类似于 OneAtATime 和 HalfAtATime 等模式的就地部署变体来实现此模式。

不可改变和蓝/绿部署

不可改变模式通过使用新的应用程序代码配置或版本来启动一组全新的服务器，以指定应用程序代码的部署。这种模式利用了云功能，此功能通过简单的 API 调用创建新的服务器资源。

蓝/绿部署策略是一种不可改变部署，它还要求创建另一个环境。新环境启动并通过所有测试后，流量将立即转移到这一新部署。至关重要的是，旧环境（即“蓝”环境）保持空闲状态，以防需要回滚。

AWS Elastic Beanstalk 支持不可改变和蓝/绿部署模式。AWS CodeDeploy 也支持蓝/绿模式。有关 AWS 服务如何实现这些不可改变模式的更多信息，请参阅 AWS 上的蓝/绿部署白皮书。

数据库架构更改

现代软件通常具有数据库层。通常使用关系数据库，该数据库同时存储数据及其结构。在持续交付过程中，经常需要修改数据库。处理关系数据库中的更改需要特别注意，除了部署应用程序二进制文件时遇到的挑战之外，还会带来其他挑战。通常，在升级应用程序二进制文件时，您会停止应用程序，对其进行升级，然后重新启动。您确实不必担心应用程序状态，它是在应用程序之外处理的。

升级数据库时，您的确需要考虑状态，因为数据库包含的状态很多，但逻辑和结构相对较少。

应用更改之前和之后的数据库架构应视为不同版本的数据库。可以使用 Liquibase 和 Flyway 等工具来管理版本。

通常，这些工具采用以下方法的某种变体：

- 将表添加到存储数据库版本的数据库中。
- 保持跟踪数据库更改命令，并将它们集中放入版本化的更改集中。对于 Liquibase，这些更改将存储在 XML 文件中。Flyway 采用的方法略有不同，其中，更改集作为单独的 SQL 文件处理，或者偶尔作为单独的 Java 类来进行更复杂的转换。
- 当要求 Liquibase 升级数据库时，它会查看元数据表并确定要运行哪些更改集，以使数据库保持最新版本。

最佳实践汇总

以下是 CI/CD 的一些最佳实践注意事项。

应执行以下操作：

- 将您的基础设施视为代码。
  - 对基础设施代码使用版本控制。
  - 利用错误跟踪/票证系统。
  - 在应用更改之前，让同伴对其进行审核。
  - 建立基础设施代码模式/设计。
  - 测试基础设施更改，例如代码更改。
- 将开发人员纳入不超过 12 名自我维系的成员的综合团队。
- 让所有开发人员频繁地向主干提交代码，并且没有长时间运行的功能分支。
- 在整个组织中始终采用诸如 Maven 或 Gradle 之类的构建系统，并实现构建过程标准化。
- 让开发人员构建单元测试，以实现 100% 的代码库覆盖率。
- 确保单元测试在持续时间、数量和范围方面占整体测试的 70%。
- 确保单元测试是最新的，不会被忽略。应修复而不是绕过单元测试失败。
- 将持续交付配置视为代码。
- 建立基于角色的安全控制（也即，谁可以做什么以及何时做）。
  - 监控/跟踪所有可能的资源。
  - 提示服务、可用性和响应时间。
  - 捕获、学习和改进。
  - 与团队中的每个人共享访问权限。
  - 在生命周期中规划指标和监控。
- 保留和跟踪标准指标。
  - 构建的数量。
  - 部署的数量。
  - 更改投入生产的平均时间。
  - 从第一个管道阶段到每个阶段的平均时间。
  - 已投入生产的更改数量。

  - 平均构建时间。
- 为每个分支和团队使用多个不同的管道。

不应执行以下操作：

- 拥有长时间运行的分支以及大型复杂的合并。
- 进行手动测试。
- 拥有手动批准流程、关卡、代码审查和安全审查。

总结

持续集成和持续交付为组织的应用程序团队提供了理想的方案。开发人员只需将代码推送到存储库即可。此代码将经过集成、测试、部署、再次测试、与基础设施合并、通过安全和质量审查，并以极高的信心做好部署准备。

使用 CI/CD 时，代码质量得到提高，软件更新可以快速交付，并且非常有信心不会发生重大变化。任何版本的影响都可以与来自生产和运营的数据相关联。它也可以用于规划下一个周期，这是组织云转型中至关重要的 DevOps 实践。

延伸阅读

有关本白皮书中讨论的主题的更多信息，请参阅以下 AWS 白皮书：

- AWS 上的部署选项概览
- 在 AWS 上实施蓝/绿部署
- 通过将 Jenkins 与 AWS CodeBuild 和 AWS CodeDeploy 集成来设置 CI/CD 管道
- AWS 上的微服务
- AWS 上的 Docker：在云环境下运行容器

贡献者

以下是对本文做出贡献的个人和组织：

- Amrish Thakkar，AWS 首席解决方案构架师
- David Stacy，AWS 专业服务 DevOps 高级顾问
- Asif Khan，AWS 解决方案构架师
- Xiang Shen，AWS 高级解决方案构架师

文档修订

要获得有关此白皮书更新的通知，请订阅 RSS 源。

更新-历史记录-更改	更新-历史记录-描述	更新-历史记录-日期
初次发布	白皮书首次发布	2021 年 10 月 27 日
初次发布	白皮书首次发布	2017 年 6 月 1 日

声明

客户负责对本文档中的信息进行独立评估判断。本文档：(a) 仅供参考；(b) 代表当前提供的 AWS 产品和实践，如有更改，恕不另行通知；并且 (c) AWS 及其附属机构、供应商或许可方不做任何承诺或保证。AWS 产品或服务“按原样”提供，不提供任何形式的保证、陈述或条件，无论是明示还是暗示。AWS 对其客户的责任和义务由 AWS 协议决定，本文档与 AWS 和客户之间签订的任何协议无关，亦不影响任何此类协议。