IT运维标准化服务

简介

术语定义：

介绍专有名词

持续集成：是在共享代码库上进行协作，将不同的代码变更合并到版本控制系统以及自动创建和测试构建的过程。

持续交付：是在持续集成过程完成后自动将构建部署到非生产环境的过程。

持续部署：在没有操作人员的干预下，通过部署到服务于真实用户的工作系统来完成闭环。

通常采用3种执行环境（包括应用运行所需的所有资源），依照重要程度一次递增的顺序列出这些环境。保持环境平等，环境同步，与生产环境的匹配程度越接近，维持高可用性以及顺利交付软件的概率就越高。

开发环境（dev）：用于集成来自多个开发人员的更新、测试基础设施改动，以及检查明显故障。

测试环境：

预发布/测试环境（stage）：用于手动和自动化测试以及进一步审查变更和软件更新。

生产环境（prod）：用于为真实用户实现服务。生产环境通常包括广泛的措施以确保高性能和强健的安全性。生产中断是一种必须立即解决的紧急情况。

应用程序：

数据库实例：

数据库：

资源：所有物理服务器功能组件（CPU、磁盘、总线...）[1]

利用率：资源忙于服务工作的平均时间[2]

饱和度：资源具有无法服务的额外工作的程度，通常会排队

error : 错误事件的计数

约定：

应用程序运行用户：非root 权限用户

应用程序部署路径：

应用程序日志路径：

操作系统版本：

数据备份目录：

服务目标

介绍服务保障的基本目标，如基本可用性（在某前提基础上的可用性）、响应时间、备份数据

服务范围

介绍服务所涵盖的范围，如生产环境系统、数据库、应用服务

运维团队

介绍团队架构组成、岗位及对应岗位职责

网络运维

系统运维：实施部署，相关服务组件交付

应用运维

数据库运维

安全运维

云运维

虚拟化运维

DevOps

服务内容

介绍所包含的服务类别及对应的核心/关键 项/指标，系统实施方案等

监控管理

目的：识别资源瓶颈或错误，快速解决常见的性能问题。

监控策略

网络监控

系统监控

应用监控

日志管理

系统日志

应用日志

系统管理

系统初始化策略

应用管理

应用运行环境约定

数据库管理

数据备份与恢复

容量规划和性能管理

安全控制和漏洞管理

故障响应流程

服务级别协议（SLA）

运维流程规范

明确运维工作的流程和步骤，确保标准的执行和一致性。包括以下流程规范：

改进与发展计划

沟通和协调

附件

工具链

监控指标、阈值

数据备份列表、存储位置、保留周期

可用性最低架构要求

概念

术语

工具

|  |  |  |  |
| --- | --- | --- | --- |
| 事项 | | 类型 | 服务范围（生产环境） |
| 监控服务 |  | 基础（设施）服务 | 1. 监控系统构建、版本更新、日常维护 2. 资源指标添加、告警规则配置 |
| 性能监控 | 系统监控 | 监控cpu、内存、存储使用情况 |
| 网络监控 | 监控流量、带宽使用、延时、丢包 |
| 应用监控（APM） | 监控应用响应时间、事务速率、错误率 |
| 可用性监控 | 服务检查 | 定期检查web、数据库服务是否可访问 |
| 业务监控 |  | 根据业务需求监控特定指标 |
| 日志服务 |  | 基础（设施）服务 | 1. 日志系统构建、日常维护 2. 收集、存储、分析日志数据 3. 关键字告警配置 |
| 系统日志 |  | 收集系统、内核、用户认证日志 |
| 应用日志 |  | 收集应用服务日志 |
| 数据库服务 |  | 基础（设施）服务 |  |
| 数据库性能优化 |  | 参数优化、查询优化，提供数据库性能 |
| 设计数据结构和关系 |  | 合理设计业务数据结构 |
| SQL 语句执行 |  | 执行业务方提交的SQL 语句 |
| 容器编排服务 |  | 基础（设施）服务 |  |
| kubernetes 集群维护 |  | kubernetes 集群版本升级，负载均衡配置，服务配置 |
| 集群管理工具维护 |  | 管理工具版本升级 |
| 网络服务 |  | 基础（设施）服务 | 网段划分，vpn网络维护，网络监控 |
| 云服务 |  | 基础（设施）服务 | 资源创建、回收，域名解析，成本控制 |
| 数据备份和恢复 |  | 基础（设施）服务 |  |
| 业务数据 |  | 定期备份业务数据，以便丢失时恢复 |
| 数据库数据 |  | 定期备份数据库数据，以便丢失时恢复 |
| 安全服务 |  | 基础（设施）服务 |  |
| 系统维护 |  | 基础（设施）服务 |  |
| 操作系统 |  | 系统环境初始化，软件版本更新 |
| 基础服务系统 |  | 服务软件安装部署，配置调试，版本更新 |

金丝雀发布实践

金丝雀发布（Canary Release）是一种软件发布策略，用于降低部署新版本软件时的风险。在这种策略中，新版本先在生产环境中的有限用户群体上部署，以便在整个系统上广泛推广之前测试新更改的性能和稳定性。这个术语来源于矿工携带金丝雀进入矿井的做法，如果空气中有有害气体，金丝雀会首先受到影响，从而提醒矿工。

实施金丝雀发布的步骤通常包括：

1. **准备金丝雀环境**：
   * 确保新版本的软件可以平滑部署到现有的生产环境中。
2. **选择金丝雀用户群**：
   * 定义用于测试新版本的用户群体，可以基于地理位置、用户行为、设备类型等因素。
3. **小规模部署**：
   * 将新版本部署到选定的金丝雀用户群体中。比例可以从非常小（比如1%的用户）开始。
4. **监控和评估**：
   * 使用监控工具密切关注新版本对性能、稳定性和用户体验的影响。
   * 设定好评估标准，例如错误率、响应时间和用户反馈。
5. **分析反馈**：
   * 收集和分析用户反馈，以及系统中的日志和错误报告。
6. **扩大或回滚**：
   * 如果金丝雀版本表现良好，可以逐步扩大新版本的覆盖范围，直到全部用户都更新到新版本。
   * 如果出现问题，应迅速回滚到旧版本以最小化对用户的影响。
7. **全面部署**：
   * 当确认新版本稳定后，可以将其推送给所有用户。

金丝雀发布需要有足够的监控和自动化工具来支持这个流程，并且它通常与持续集成/持续部署（CI/CD）流程相结合。这种策略的关键优势在于它可以及时发现问题并将其影响降到最低，同时还可以为最终的全面部署提供实时反馈。然而，金丝雀发布也需要精心规划和执行，以确保能够快速响应任何问题。

开发人员维护生产环境

开发人员维护生产环境是近年来越来越多采纳的一种做法，尤其是在采用DevOps文化的公司中。这种做法的基本原则是“你构建它，你运行它”，即让开发人员对整个软件生命周期负责，包括开发、测试、部署、维护和监控在内。

当开发人员参与到生产环境的维护中时，以下是一些关键的实践：

1. **持续集成和持续部署（CI/CD）**：
   * 自动化的CI/CD管道有助于在开发人员提交代码后快速、可靠地构建、测试和部署到生产环境。
2. **监控和日志**：
   * 开发人员应配置和使用监控工具来跟踪生产环境的性能和健康状况。
   * 应用程序和基础设施的日志应该集中存储和管理，以便快速定位和解决问题。
3. **告警和通知**：
   * 设置基于阈值的告警，当发生异常时能够及时通知到责任人。
4. **问题排查和故障恢复**：
   * 开发人员应具备足够的权限和工具来快速响应生产问题，并能够执行必要的故障恢复操作。
5. **文档和知识共享**：
   * 维护良好的文档，记录系统架构、配置和常见问题的解决方案。
   * 促进团队内部知识共享，确保多人了解系统的关键部分。
6. **安全性**：
   * 开发人员需要了解和遵循最佳的安全实践，包括使用安全的编码标准、管理好敏感数据等。
7. **性能优化**：
   * 定期审查性能指标，识别瓶颈并进行优化。
8. **灾难恢复计划**：
   * 制定并测试灾难恢复计划，确保在严重故障时能够快速恢复服务。
9. **用户支持和反馈**：
   * 直接处理用户的问题和反馈，可以帮助开发人员更好地理解用户需求和体验。
10. **自动化和工具化**：
    * 开发工具和脚本来自动化常见的维护任务，减少手工操作的错误和时间。

开发人员维护生产环境能够带来许多好处，包括更快的故障响应时间、对产品的更深入理解、以及更高的系统稳定性和用户满意度。但这也要求开发人员拥有跨领域的技能和更高的责任心。同时，组织需要确保开发人员有足够的支持和培训，以及合适的工具和权限来执行他们的任务。

pipeline 循环

在软件开发中，"pipeline 循环"通常指的是持续集成/持续部署（CI/CD）的流程，它是自动化地将代码从版本控制系统（如Git）通过各种测试和部署阶段最终发布到生产环境的一系列步骤。这个循环确保了软件的快速迭代和高质量，以下是这个流程的关键步骤：

1. **代码提交（Commit）**：
   * 开发人员将代码变更提交到版本控制系统的共享仓库。
2. **自动构建（Build）**：
   * 持续集成服务器监听版本控制系统的变化，当检测到新的提交时，自动开始构建流程。
3. **单元测试（Unit Testing）**：
   * 在构建过程中自动运行单元测试来验证代码更改不会破坏现有的功能。
4. **代码分析（Code Analysis）**：
   * 自动代码质量分析，包括检查代码风格、潜在的错误和性能问题。
5. **集成测试（Integration Testing）**：
   * 自动化测试来验证各个组件或服务之间的交互是否按预期工作。
6. **部署到测试环境（Deploy to Test Environment）**：
   * 自动将代码部署到一个或多个测试环境中，进行更深入的测试。
7. **端到端测试（E2E Testing）**：
   * 在测试环境中自动执行端到端的测试，模拟用户操作来验证整个系统的行为。
8. **负载和性能测试（Load & Performance Testing）**：
   * 自动测试应用程序在高负载和压力条件下的性能。
9. **审批和门控（Approval and Gating）**：
   * 根据需要，流程中可以包括手动审批步骤，以决定是否将变更推送到生产环境。
10. **部署到生产环境（Deploy to Production）**：
    * 在通过所有测试和审批后，自动将代码部署到生产环境。
11. **监控和反馈（Monitoring & Feedback）**：
    * 持续监控生产环境的状态，并将任何问题或性能指标反馈给开发团队。

这个循环是迭代的，每次代码提交都会触发这一系列自动化步骤，确保快速、频繁且可靠地交付软件。CI/CD pipeline 的实现通常依赖于各种工具，如Jenkins、GitLab CI、Travis CI、CircleCI、Azure DevOps等，这些工具可以帮助自动化测试、构建、部署等各个阶段的任务。

pipeline 手动审批权限实现

在CI/CD pipeline中实现手动审批通常是为了确保在代码自动部署到生产环境之前有一个质量检查的门槛，或者确保关键的业务决策者对发布有最终的控制权。以下是实现手动审批过程的一些步骤：

1. **选择工具**：
   * 首先，你需要选择一个支持手动审批步骤的CI/CD工具。大多数现代CI/CD工具例如Jenkins, GitLab CI, CircleCI, Azure DevOps等都支持这一功能。
2. **配置流水线**：
   * 在你的CI/CD流水线配置中，你可以定义一个特定的审批步骤或作业。通常，这是在自动化测试通过之后，代码部署到生产之前的步骤。
3. **定义审批角色和权限**：
   * 定义谁有权限批准部署。这通常是通过CI/CD工具的角色和权限管理功能配置的，确保只有授权的用户可以批准或拒绝变更。
4. **设置通知**：
   * 设置通知机制以告知审批者需要他们的输入。这可以通过电子邮件、聊天工具（如Slack）、或者直接在CI/CD工具中的通知系统实现。
5. **手动审批操作**：
   * 审批者可以在CI/CD工具的用户界面中，查看待审批的变更，并选择批准或拒绝部署。
6. **审批日志**：
   * 出于审计的目的，所有的审批操作都应该被记录下来。大多数CI/CD工具会自动进行这种记录。
7. **后续步骤**：
   * 如果审批通过，流水线将继续执行后续的部署到生产环境的步骤。
   * 如果审批被拒绝，流水线可以配置为发送通知、记录原因、并停止进一步的操作。

例如，在Jenkins中，可以使用“Input”步骤来请求手动审批，而在GitLab CI中，则可以使用when: manual关键字来定义手动作业。在Azure DevOps中，可以使用环境的"Check"功能来实现审批流程。

这样的手动审批步骤可以有效地作为质量保证的最后关卡，但也会增加部署的时间。因此，组织需要在自动化和控制之间找到合适的平衡点。

jenkins 配置手动审批权限

在Jenkins中，你可以使用"Input"步骤来要求在Pipeline中进行手动审批。这通常在"Jenkinsfile"中定义，也就是用于描述Pipeline的文本文件。这里是如何配置的基本步骤：

1. **编辑Jenkinsfile**： 打开你的Jenkinsfile并加入一个input步骤在适当的阶段，通常是在自动测试通过之后和部署到生产环境之前。

例如：

pipeline {

agent any

stages {

stage ('Build') {

steps {

// 添加构建步骤

}

}

stage ('Test') {

steps {

// 添加测试步骤

}

}

stage('Deploy to Staging') {

steps {

// 添加部署到预发布环境的步骤

}

}

stage('Approval') {

steps {

script {

input message: 'Approve deployment?', ok: 'Deploy to Production'

}

}

}

stage ('Deploy to Production') {

steps {

// 添加部署到生产环境的步骤

}

}

}

}

1. **配置权限**： 你需要确保只有授权的用户才能批准部署。在Jenkins中，你可以通过“角色策略插件”来管理用户权限。你可以为不同的用户分配不同的角色，并限制谁可以执行输入步骤。
2. **使用脚本**： 如果你需要更复杂的权限控制，你可以在input步骤中使用脚本来查询外部系统，或者根据某些条件来决定是否允许用户进行审批。

例如：

stage('Approval') {

steps {

script {

def user = input(

id: 'userInput', message: 'Approve deployment?',

parameters: [[$class: 'ChoiceParameterDefinition',

choices: 'user1\nuser2\nuser3',

description: 'Select user',

name: 'USER']]

)

if (!user.equals('user1')) {

error("Only user1 is allowed to approve")

}

}

}

}

1. **审批操作**： 当Pipeline运行到需要审批的步骤时，它会暂停并等待用户的输入。授权的用户需要登录Jenkins，导航到对应的Pipeline运行，并进行审批操作。
2. **审批记录**： Jenkins会自动记录下哪个用户进行了审批操作，这对于跟踪和审计非常有用。

请记住，实际的配置可能会根据你的具体需求和Jenkins的版本而有所不同。另外，确保Jenkins实例安全是非常重要的，因此请确保你的Jenkins实例和插件都保持更新，并且正确配置了安全设置。

CI/CD 环境管理最佳实践

CI/CD（持续集成/持续部署）环境管理是软件开发中一个重要的部分，它确保了软件从开发到部署的各个阶段能够顺畅、高效、安全地流转。以下是一些CI/CD环境管理的最佳实践：

1. **环境一致性**：
   * 确保开发、测试、预发布和生产环境尽可能地保持一致。使用基础设施即代码（Infrastructure as Code, IaC）工具如Terraform、Ansible或CloudFormation管理环境配置可以帮助实现这一点。
2. **安全性**：
   * 保护你的CI/CD流程免受未授权访问。使用角色基权限管理（RBAC）确保只有授权用户才能执行构建或部署。
   * 对所有的敏感数据（如API密钥、密码）使用密钥管理系统，不要直接存储在代码库或配置文件中。
3. **自动化和自服务**：
   * 自动化所有可自动化的任务，并提供自服务门户，让开发者可以自己启动环境或部署，而无需运维团队的手动干预。
4. **版本控制**：
   * 所有环境配置都应该被版本控制，这样你可以追踪变更、快速回滚以及审计使用。
5. **监控和日志记录**：
   * 对所有环境进行适当的监控和日志记录，确保你可以快速发现并解决问题。集中日志管理和实时监控告警是非常关键的。
6. **构建一次，部署多次**：
   * 构建的工件（如二进制文件、Docker镜像）应该在整个流程中保持不变，确保你部署到生产环境的是与你在测试环境中测试过的相同的东西。
7. **使用蓝/绿或金丝雀发布策略**：
   * 通过蓝/绿部署或金丝雀发布来减少部署风险。通过这种方式，你可以将流量逐渐转移到新版本上，而不是一次性替换所有的实例。
8. **依赖管理**：
   * 清晰地定义并锁定依赖关系，确保从开发到生产的所有环境中依赖的一致性。
9. **灾难恢复和备份**：
   * 定期备份配置和数据，并确保你有恢复计划以防万一。
10. **文档和培训**：
    * 提供详尽的文档和培训，让团队成员理解环境管理流程和操作指南。
11. **持续改进**：
    * 定期审查并改进CI/CD流程。收集反馈，使用度量来量化改进的影响，并调整流程以适应新的需求或解决发现的问题。

通过遵循这些最佳实践，你可以建立一个可靠、高效、安全的CI/CD环境，从而帮助你的团队更快地交付高质量的软件。