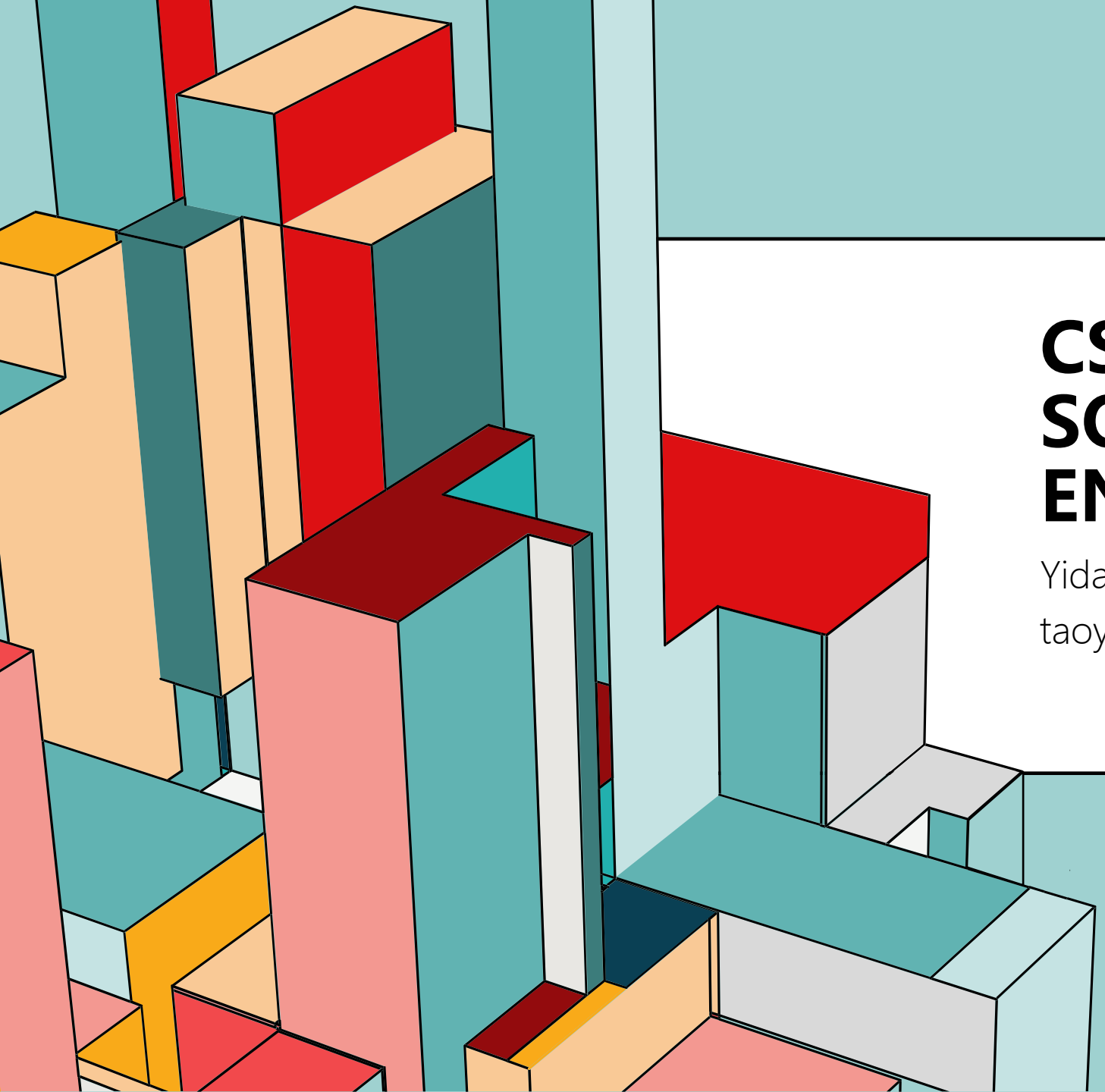


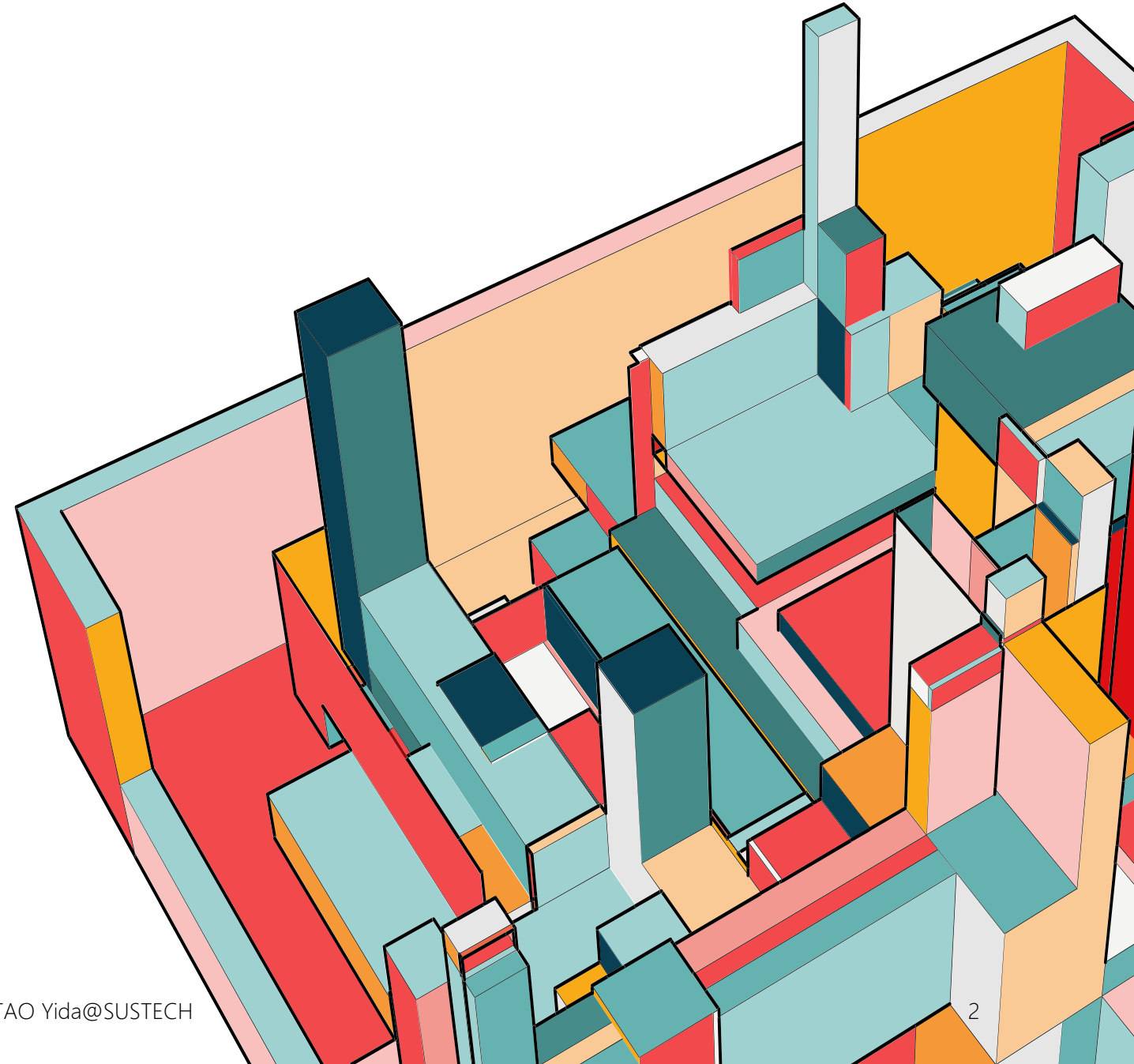
CS304 SOFTWARE ENGINEERING

Yida Tao

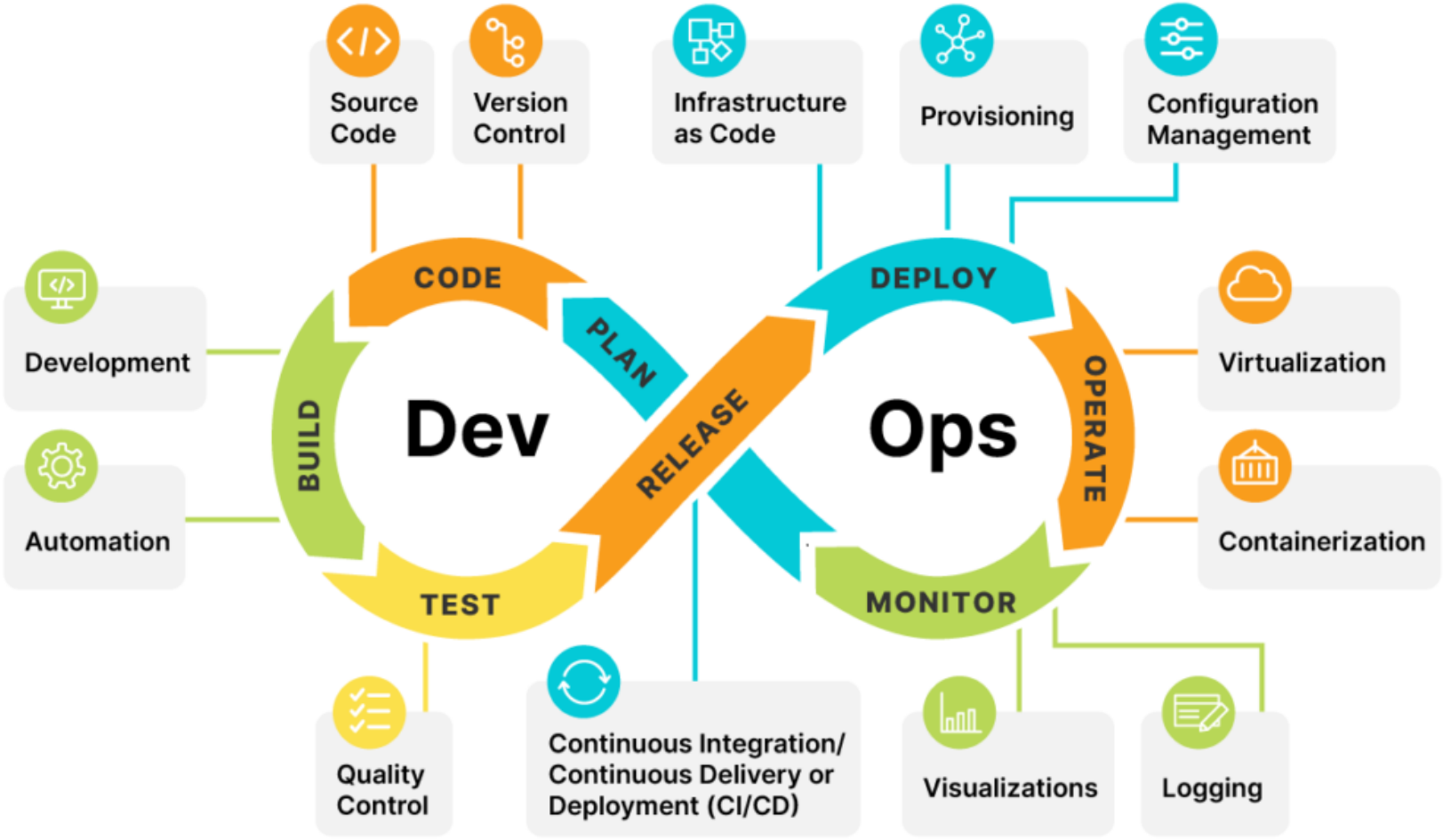
taoyd@sustech.edu.cn

LECTURE 12

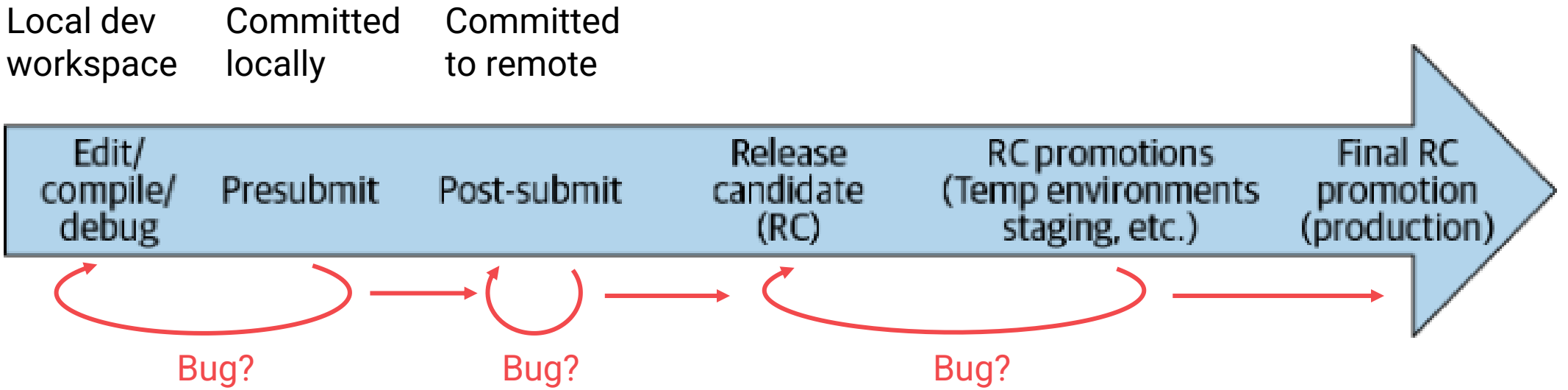
- CI/CD
- Deployment Strategy
- Cloud-native Applications
- Deployment Pipelines



WHERE ARE WE NOW?



LIFE OF A CODE CHANGE



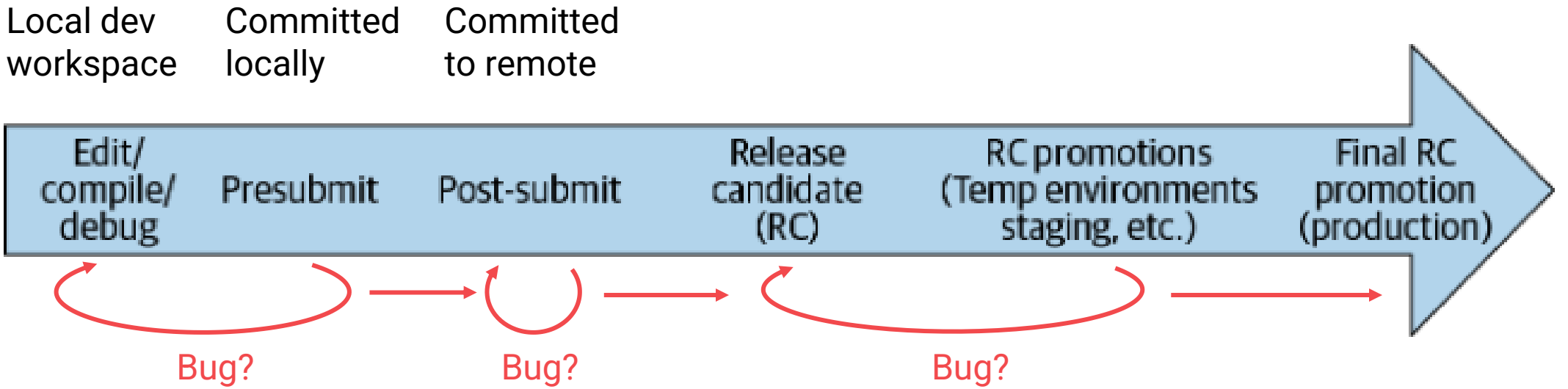
Automated testing error within the module

Integration error with another module

Bugs on environment configurations
Incompatible with upstream service dependency

Bugs reported by users

LIFE OF A CODE CHANGE



To minimize the cost of bugs, we need a feedback loop whenever we integrate a code change

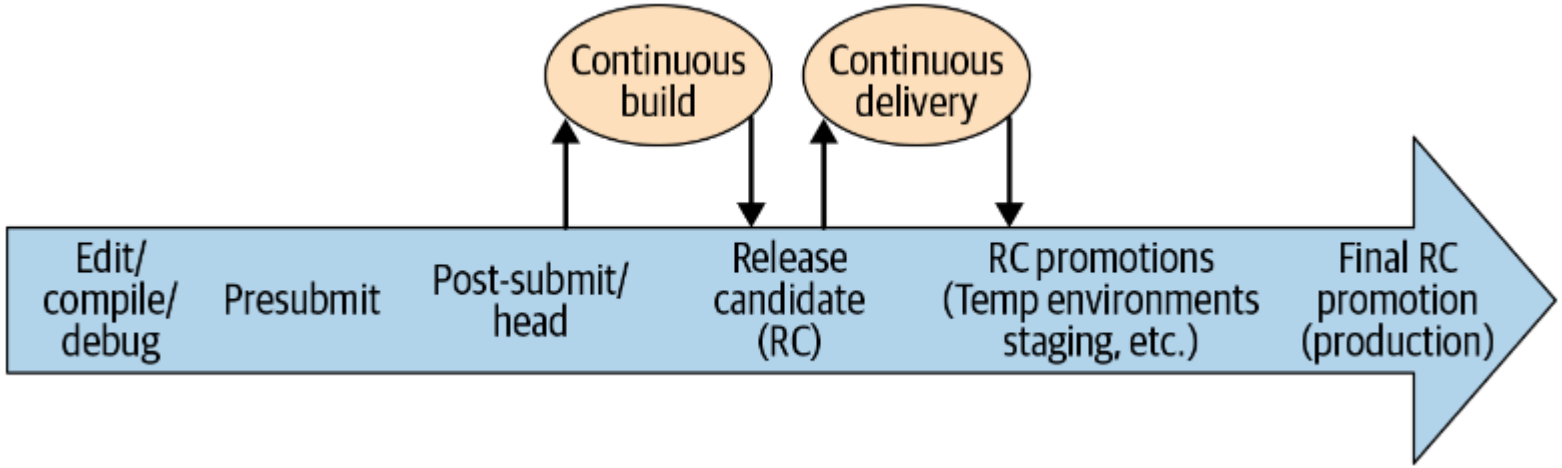
CONTINUOUS INTEGRATION (CI)

- CI is a software development practice where members of a team integrate their work frequently
- Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible

The fundamental goal of CI is to automatically catch problematic changes as early as possible.

CONTINUOUS INTEGRATION (CI)

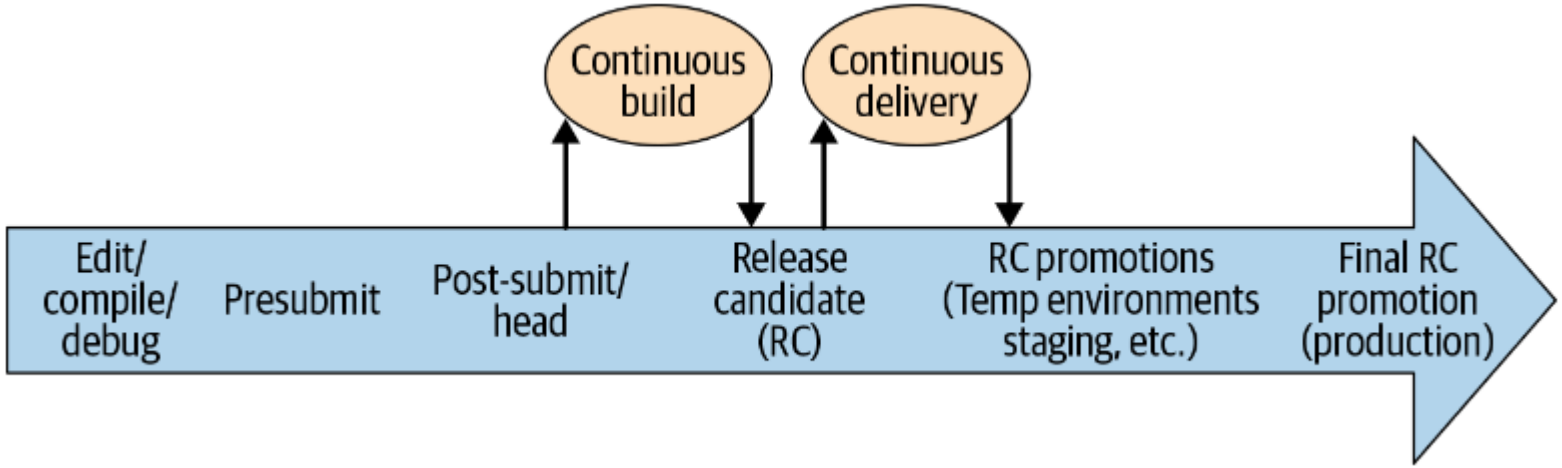
- CI, specifically, automates the build and release processes, with a Continuous Build and Continuous Delivery.
- Continuous testing is applied throughout the entire process (e.g., unit/integration/system test, etc.)



CONTINUOUS BUILD (CB)

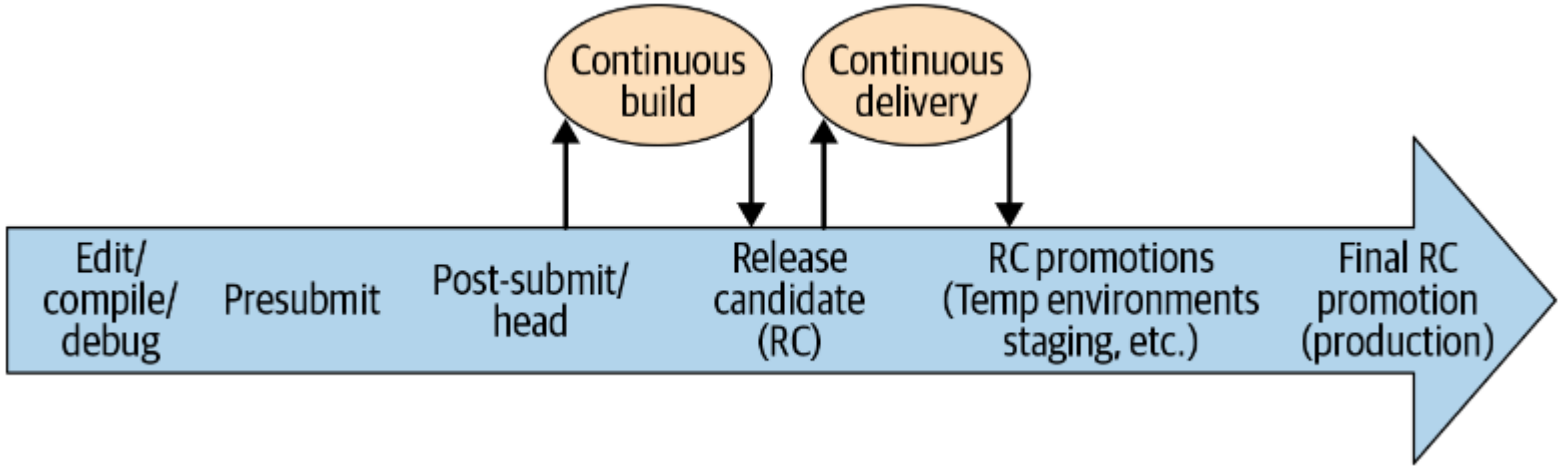
- CB integrates the latest code changes at HEAD
- CB runs automated build
- CB runs automated tests
- If the code passes all tests, CB marks it passing

"Breaking the build" or "failing the build" includes breaking compilation or breaking tests



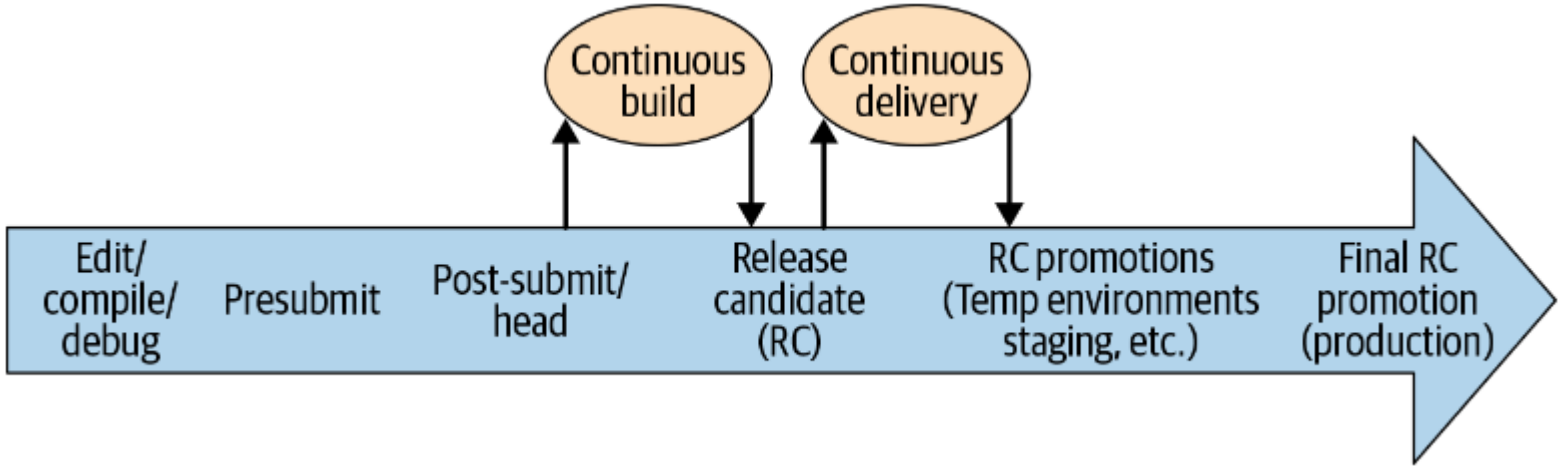
CONTINUOUS BUILD (CB)

- Release candidate (RC): a cohesive, deployable unit created by an automated process, assembled of code, configuration, and other dependencies that have passed the continuous build.



CONTINUOUS DELIVERY (CD)

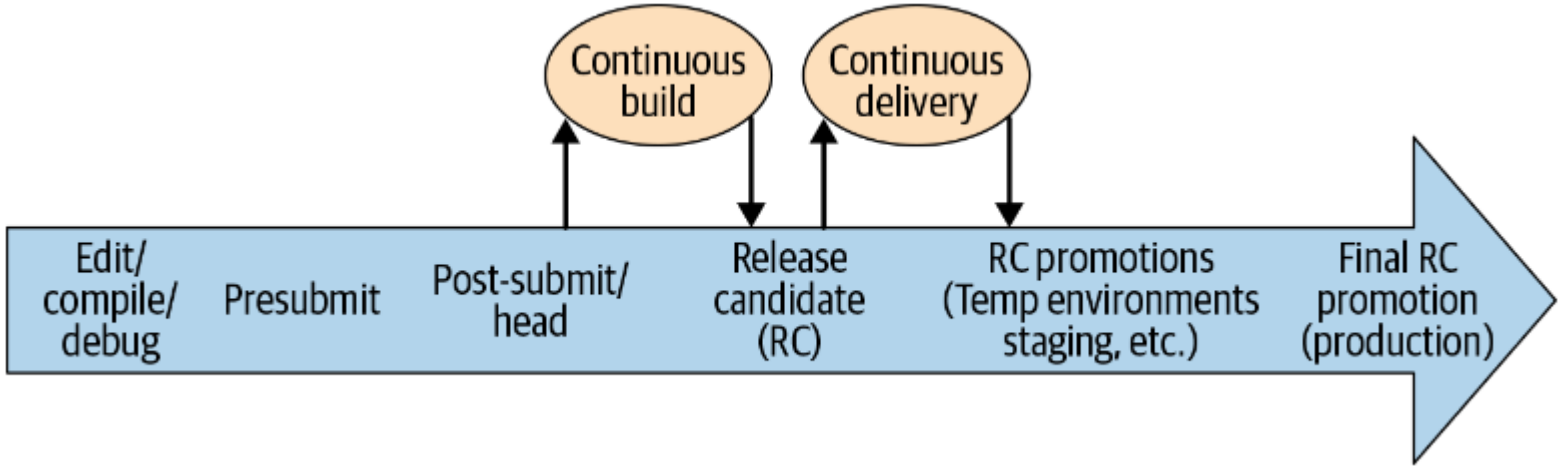
- CD: a continuous assembling of release candidates, followed by the promotion and testing of those candidates throughout a series of environments—sometimes reaching production and sometimes not.



CONTINUOUS TESTING

- Continuous testing is applied throughout the CI process
- Key objective: determining **when** to test **what**

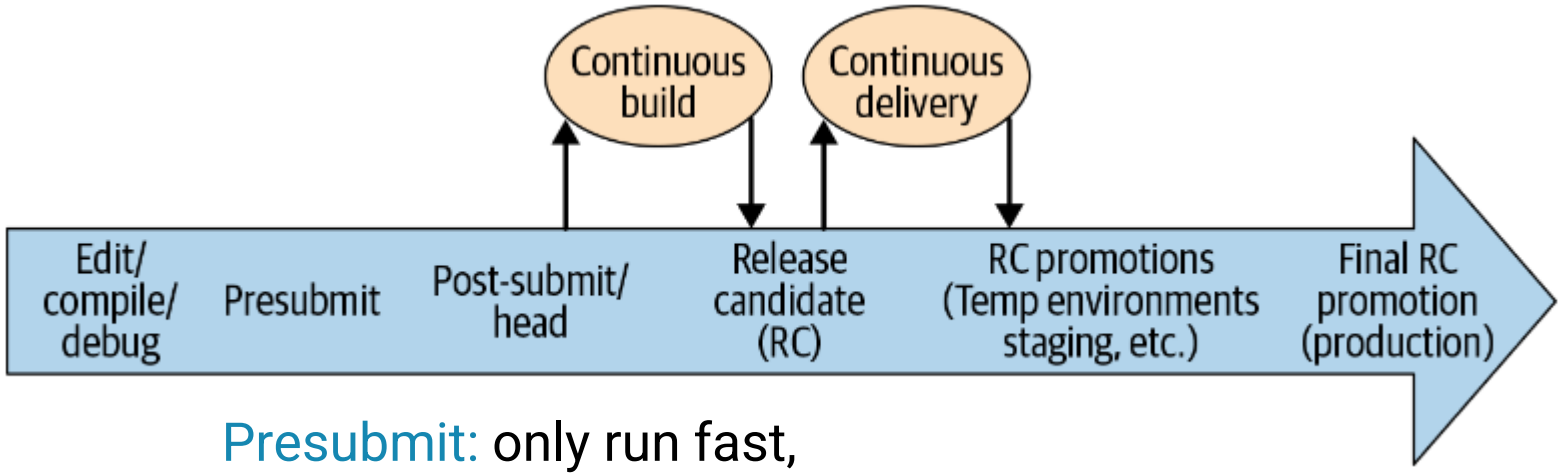
Why not just run all tests on presubmit?



CONTINUOUS TESTING - PRESUBMIT

Case study at Google

- Each team creates a **fast subset** of tests, often a project's unit tests, that can be run before a change is submitted and code reviewed (i.e., presubmit)
- Empirically, a change that passes the presubmit has a very high likelihood (95%+) of passing the rest of the tests

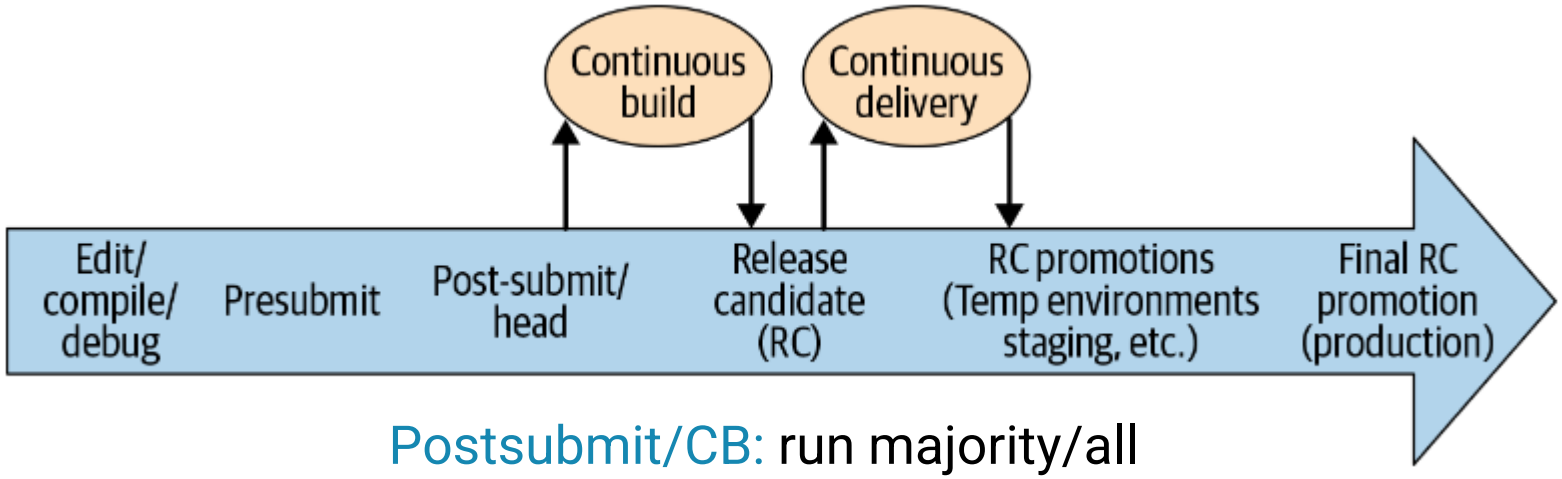


Presubmit: only run fast, reliable tests (e.g., unit tests)

CONTINUOUS TESTING - POSTSUBMIT/CB

Case study at Google

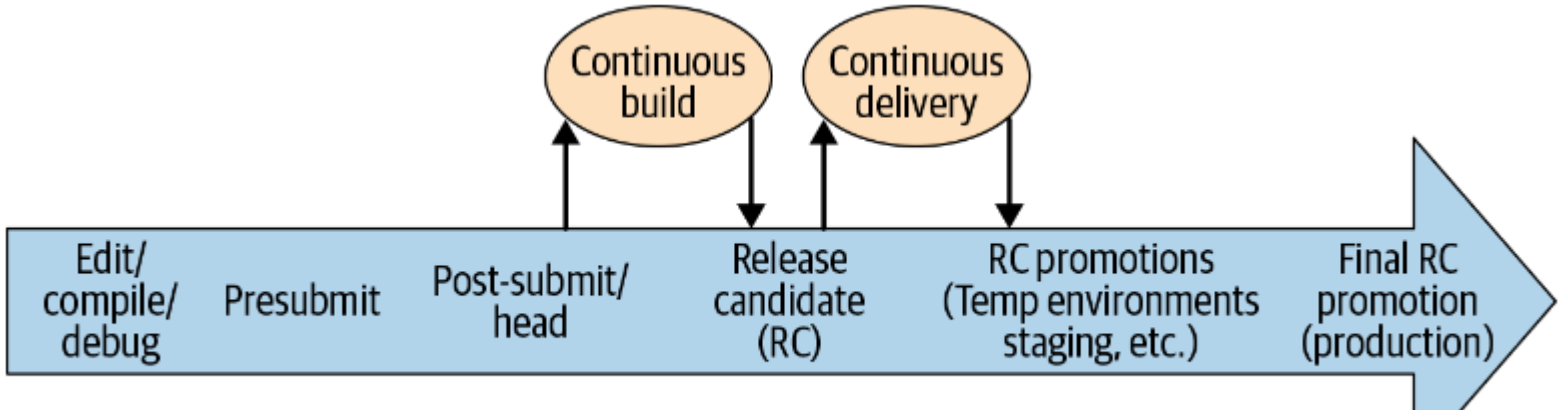
- After a change has been submitted, the Test Automation Platform (TAP) run all potentially affected tests, including larger and slower tests
- When a change causes a test to fail on TAP, engineers decide whether to fix it or roll back (TAP enables automatic rollback on high-confidence cases)



Postsubmit/CB: run majority/all automated tests

CONTINUOUS TESTING - RC PROMOTION/CD

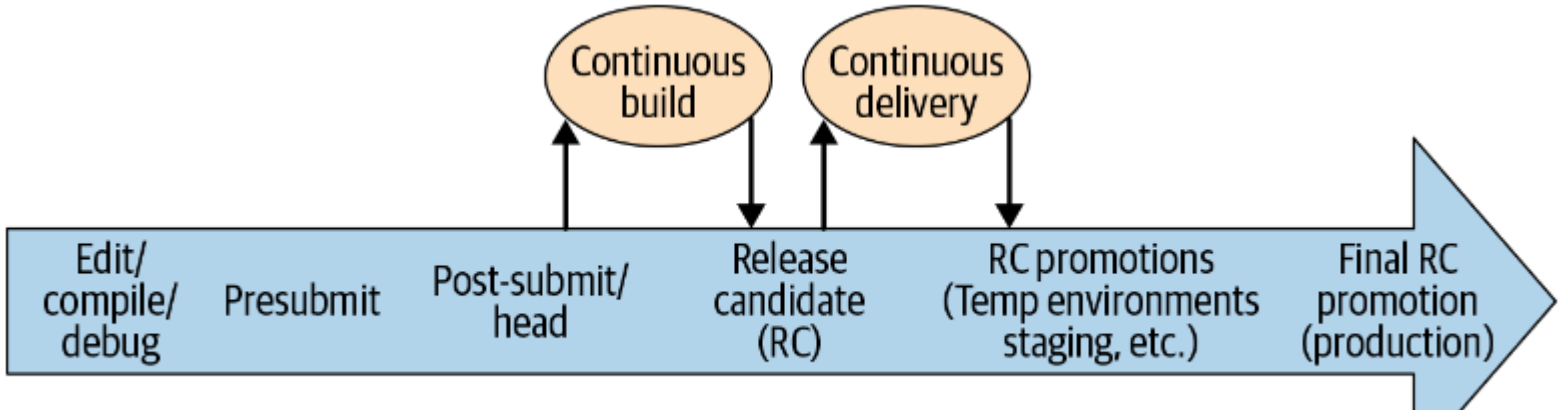
- Staging (类生产环境) is a test environment that exactly resembles a production environment
- It seeks to mirror an actual production environment as closely as possible and may connect to other production services and data, such as databases and servers.



RC promotion/CD: RC is tested at each deployment on a series of test environments (e.g., sandbox, staging)

CONTINUOUS TESTING - RC PROMOTION/CD

- The primary use of a staging environment is to test all the installation/configuration/migration scripts and procedures before they're applied to a production environment.
- Another important use of staging is performance testing, particularly load testing

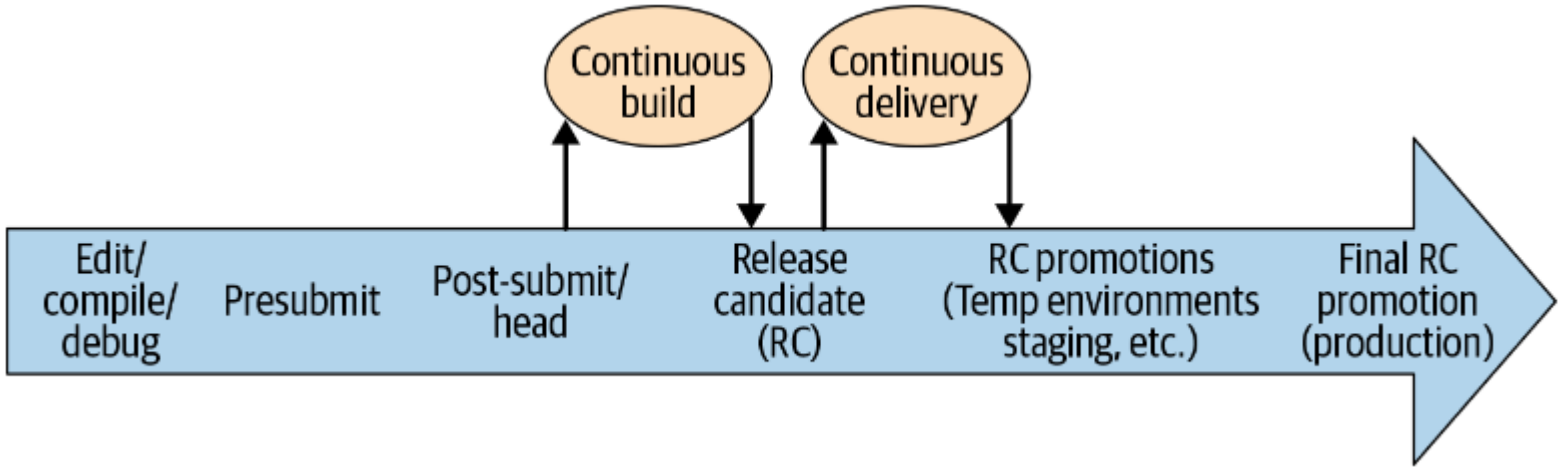


Ready to be deployed to the production server

RC promotion/CD: RC is tested at each deployment on a series of test environments (e.g., sandbox, staging)

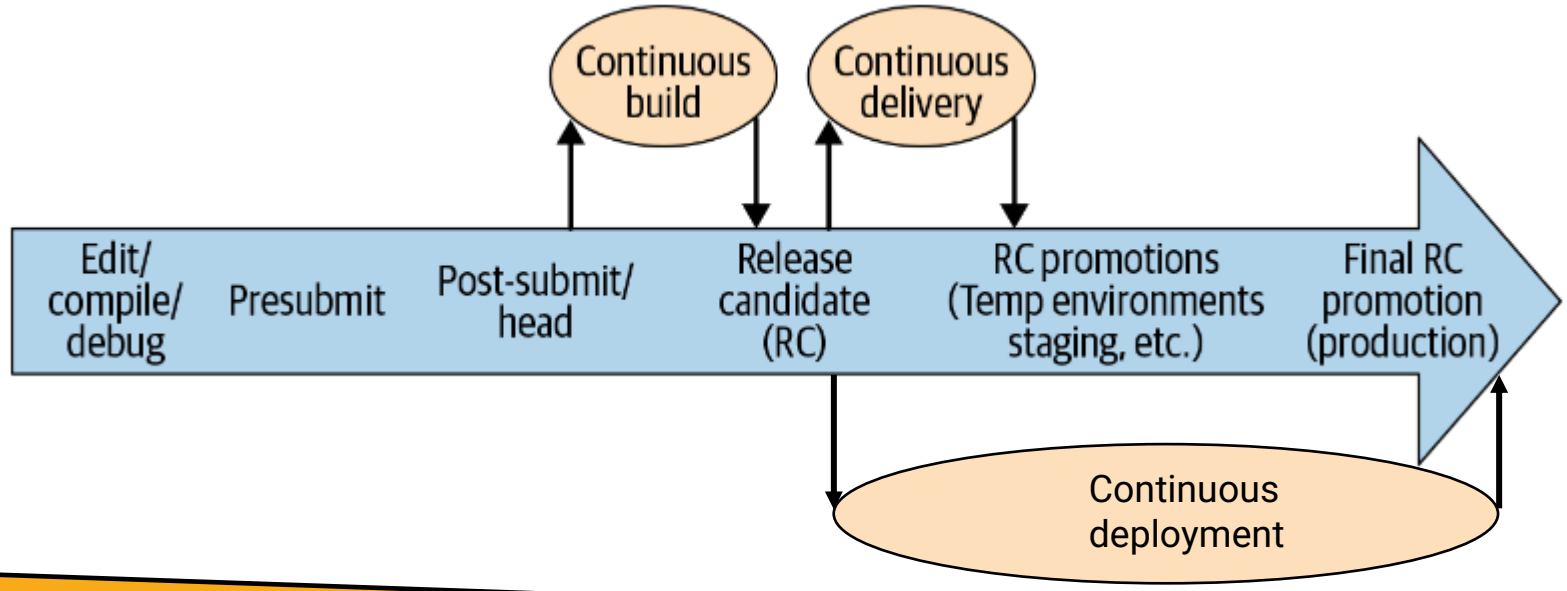
CONTINUOUS DEPLOYMENT

- CI/CD is (less often) referred to as continuous integration / continuous deployment
- Difference between continuous delivery and continuous deployment
- Continuous delivery automates deployment of a release to a staging or testing environment
  - Make sure that software changes are always in a releasable state (can be deployed at any time)
  - Decision of whether to deploy changes to production is left to the team (human intervention)



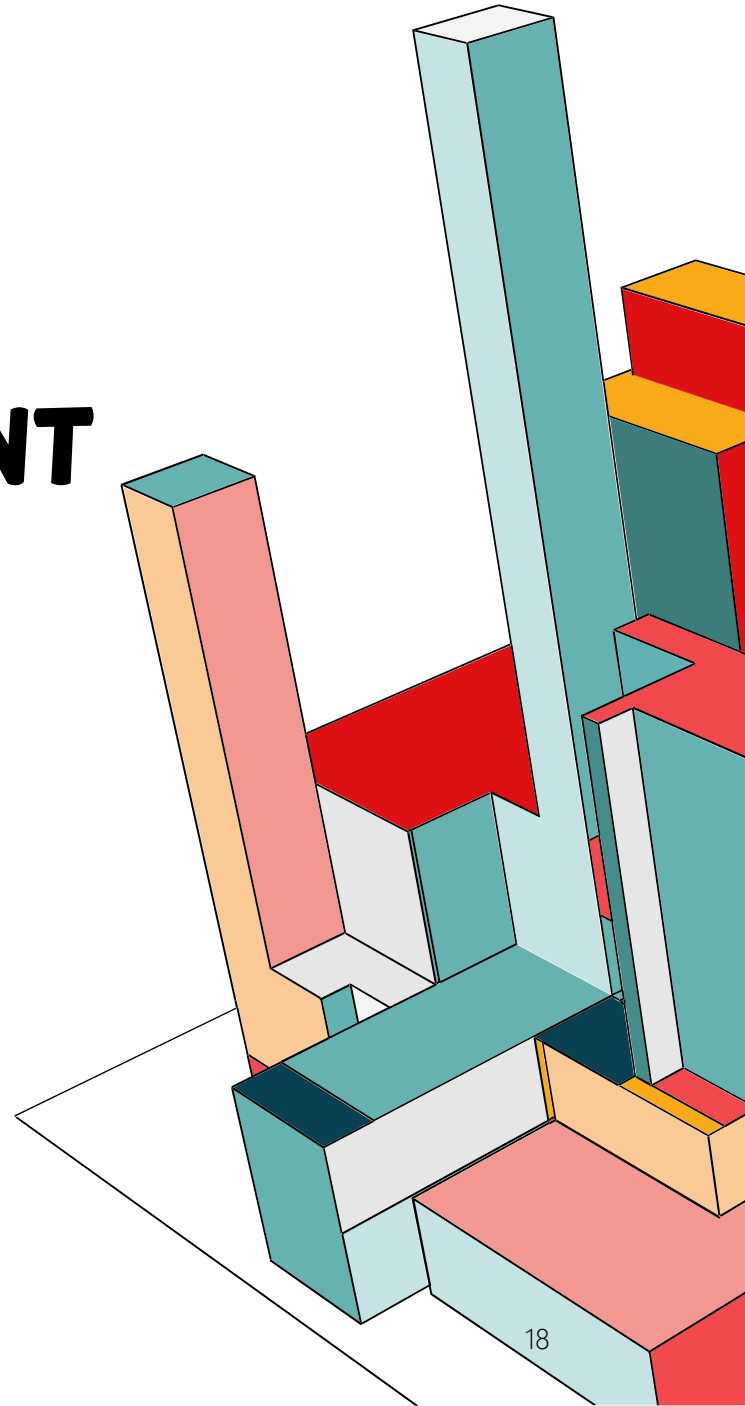
CONTINUOUS DEPLOYMENT

- CI/CD is (less often) referred to as continuous integration / continuous deployment
- Difference between continuous delivery and continuous deployment
- Continuous deployment: automates deployment of a release all the way to the production environment, no human intervention



RISKS OF CONTINUOUS DEPLOYMENT

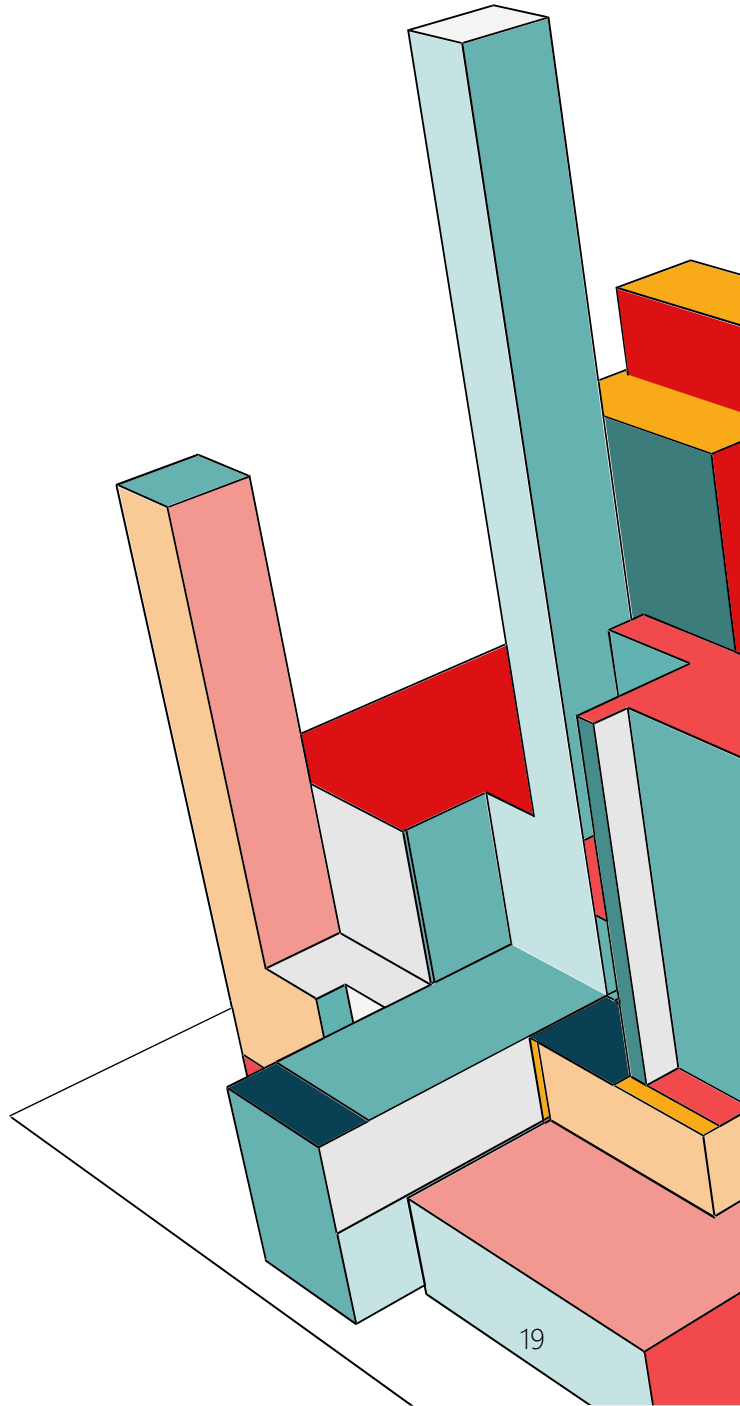
- New production version has problems and needed to be rolled back to previous versions
- Service is temporarily down during upgrading or rollback
- ……



DEPLOYMENT STRATEGIES

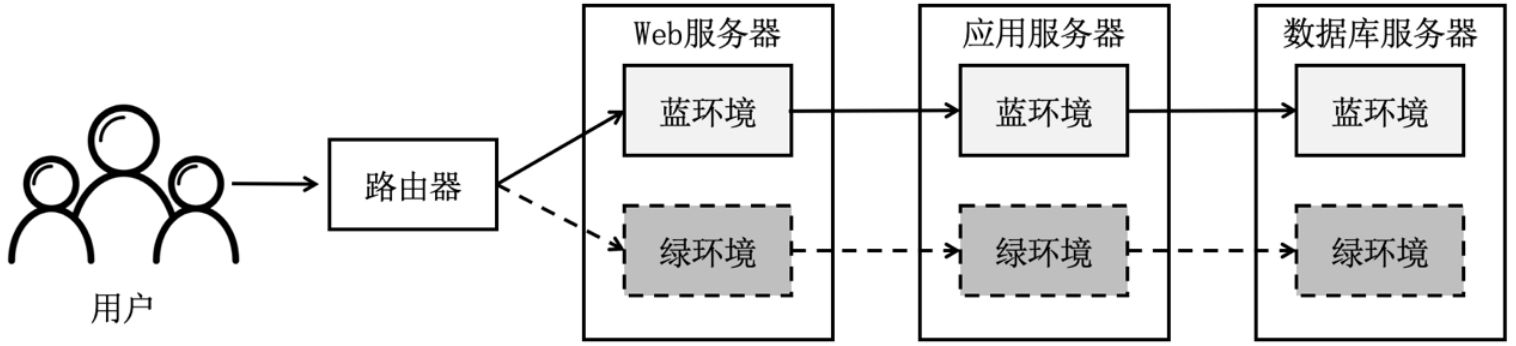
Goal: minimize the impact of new deployment on end users

- Blue-Green Deployment (蓝绿部署)
- Canary/Greyscale Release (金丝雀发布/灰度发布)



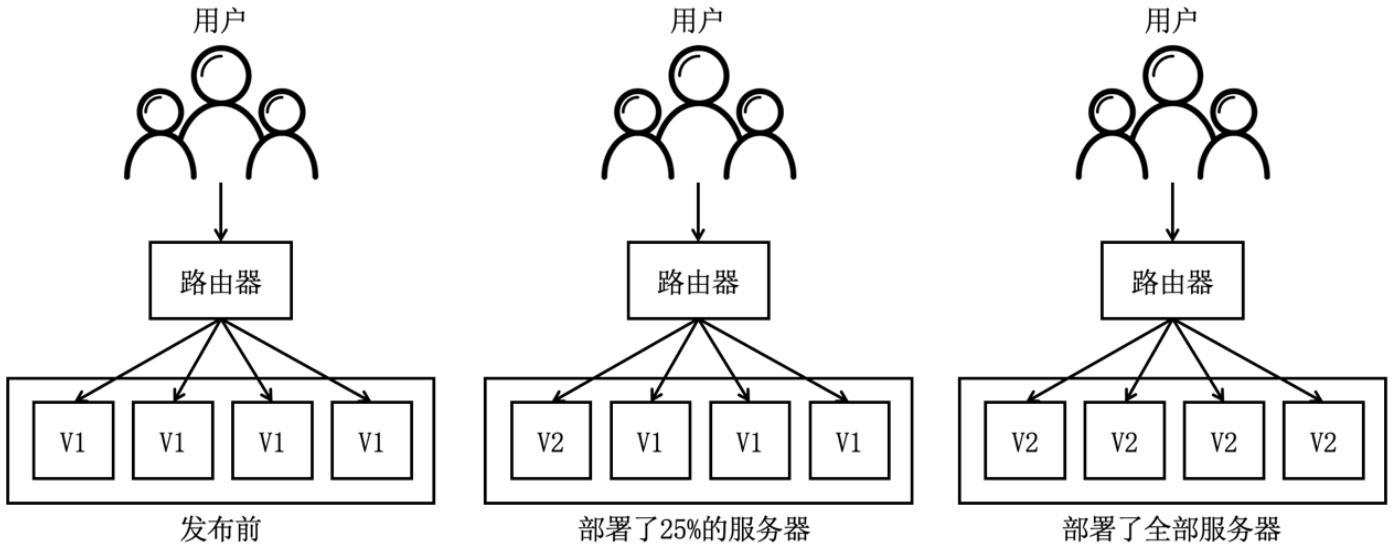
BLUE-GREEN DEPLOYMENT

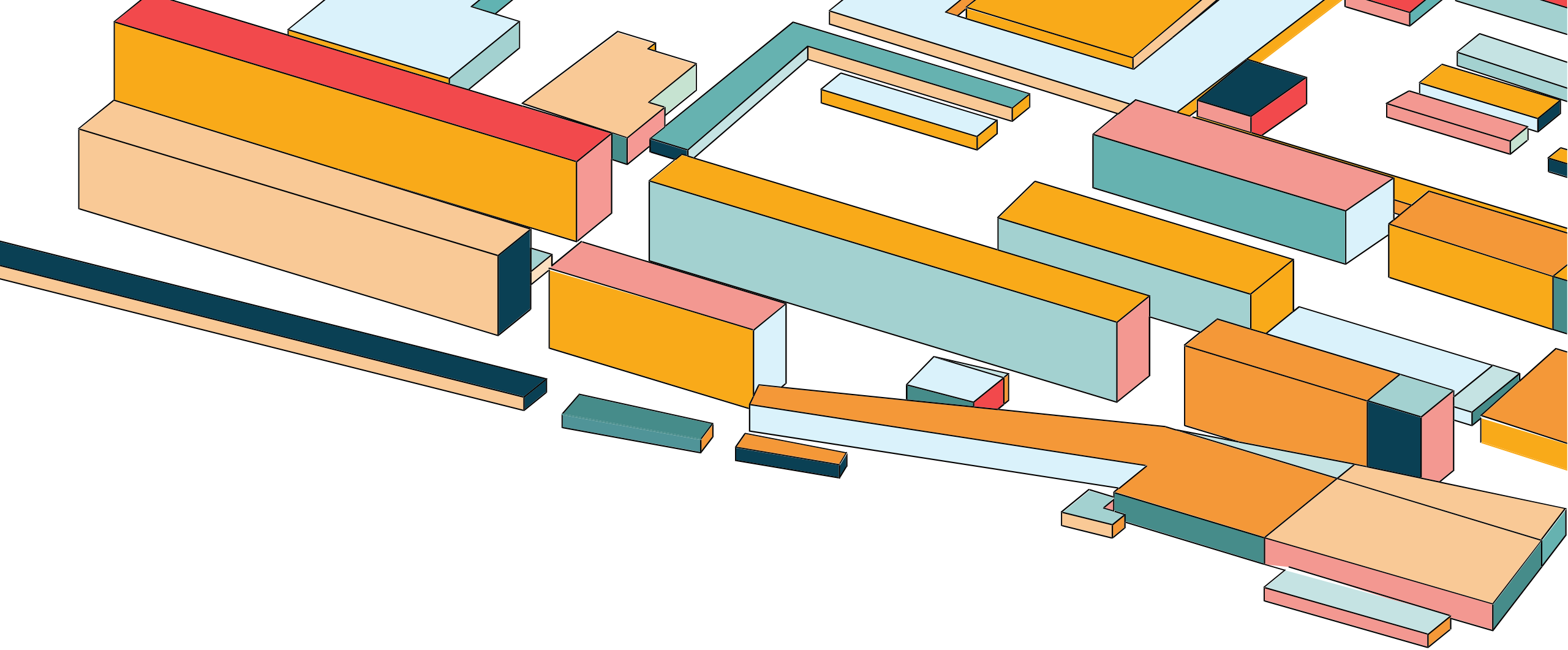
- A deployment strategy in which you create two separate, but identical environments.
- Blue environment is running the current/old application version
- Green environment is running the new application version.
- Once testing has been completed on the green environment, live application traffic is directed to the green environment.
- If error occurs, switch the traffic back to the blue environment



Canary release (greyscale release)

- A deployment strategy to reduce the risk of introducing a new software version in production
- Slowly rolling out the change to **a small subset of users (canary)** before rolling it out to the entire infrastructure and making it available to everybody.
- Once error occurs, simply stop the traffic to the server with new version





CLOUD NATIVE APPLICATIONS

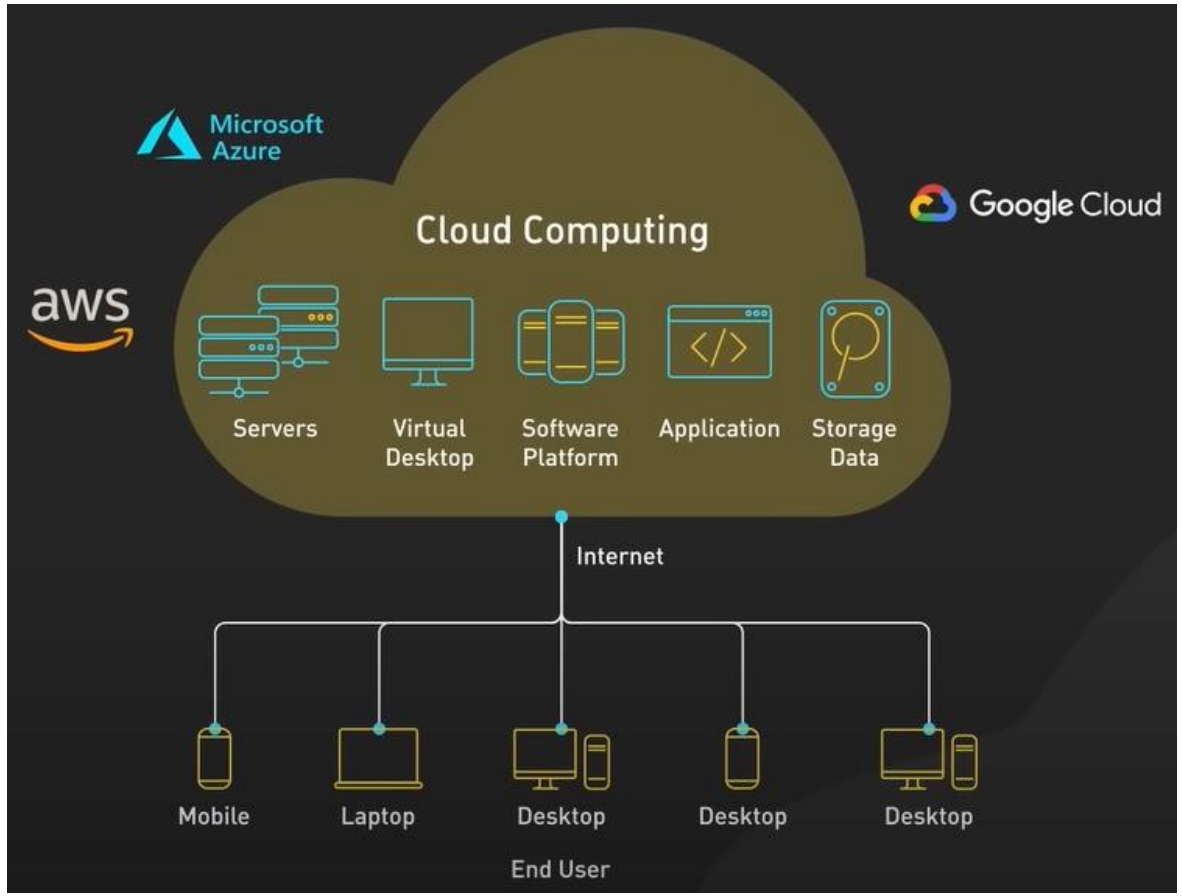
WHAT IS CLOUD NATIVE (云原生)?

https://www.youtube.com/watch?v=p-88GN1WVs8

CLOUD COMPUTING

Running applications on computing resources managed by cloud providers (e.g., aws), without having to purchase and manage hardware ourselves

https://www.youtube.com/watch?v=p-88GN1WVs8

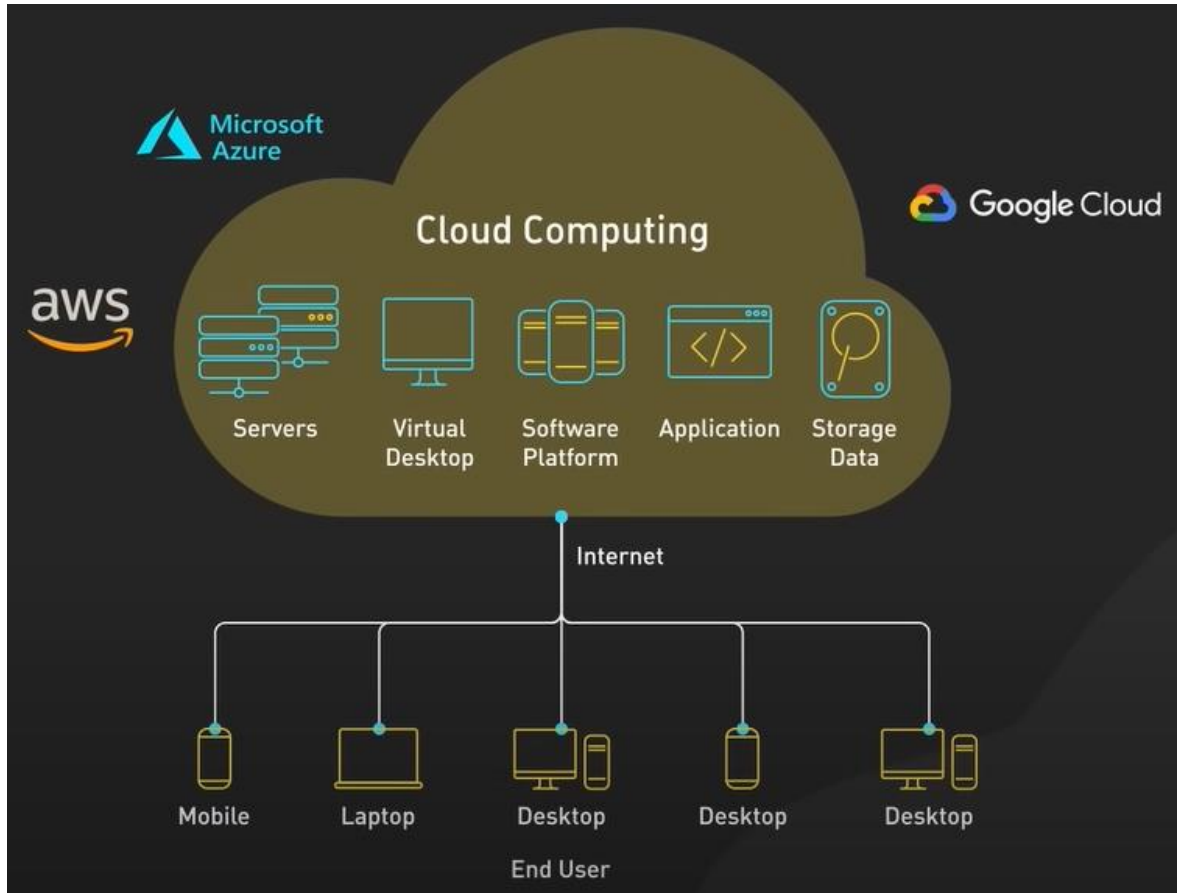


CLOUD COMPUTING

Still not cloud-native applications

- Free the team from managing hardware infrastructure
- Fast to support new computing resources
- Scaling up is effortless

https://www.youtube.com/watch?v=p-88GN1WVs8

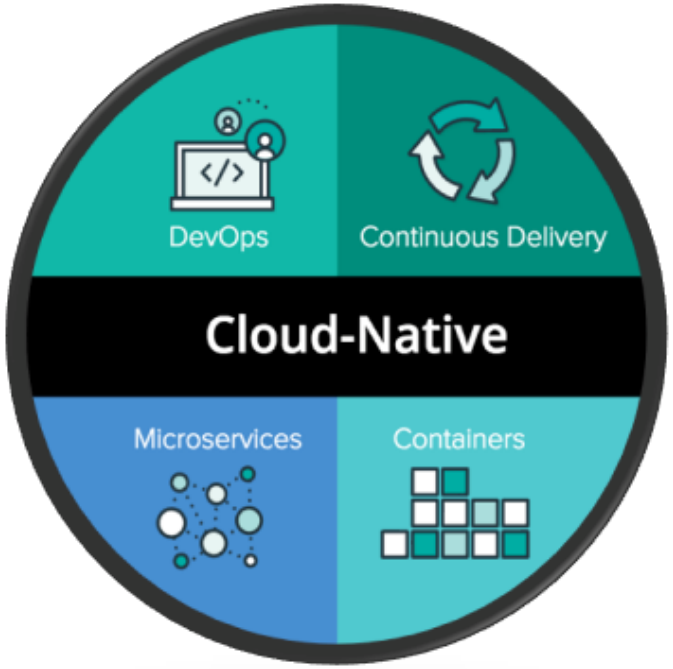


CLOUD NATIVE APPLICATIONS

Cloud native apps are designed and built to exploit the scale, elasticity, resiliency, and flexibility the cloud provides.

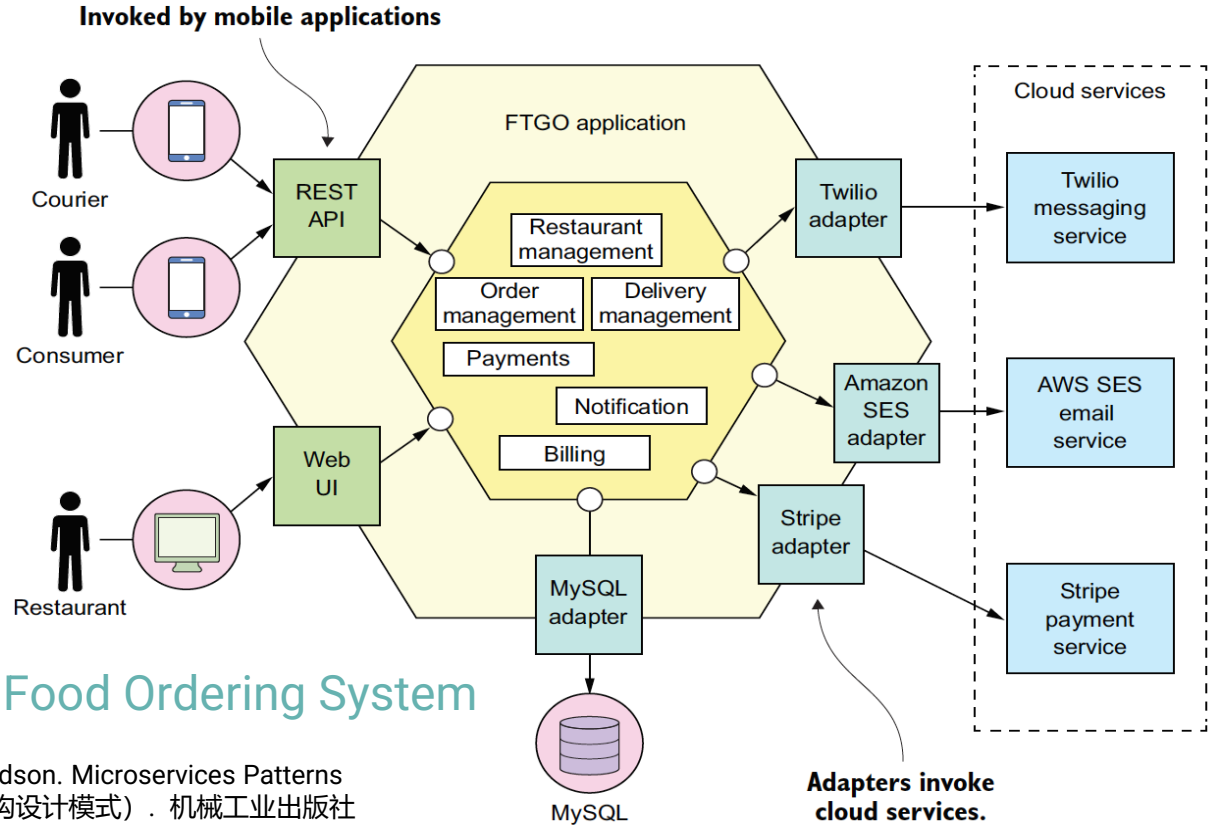
For an application to be considered as cloud-native, it should at least adopt these 4 approaches or technologies

- DevOps/Continuous Delivery
- Microservices
- Containers
- Cloud-native Open Standards



MICROSERVICES

Cloud-native applications consist of multiple small, interdependent services called microservices.



Online Food Ordering System

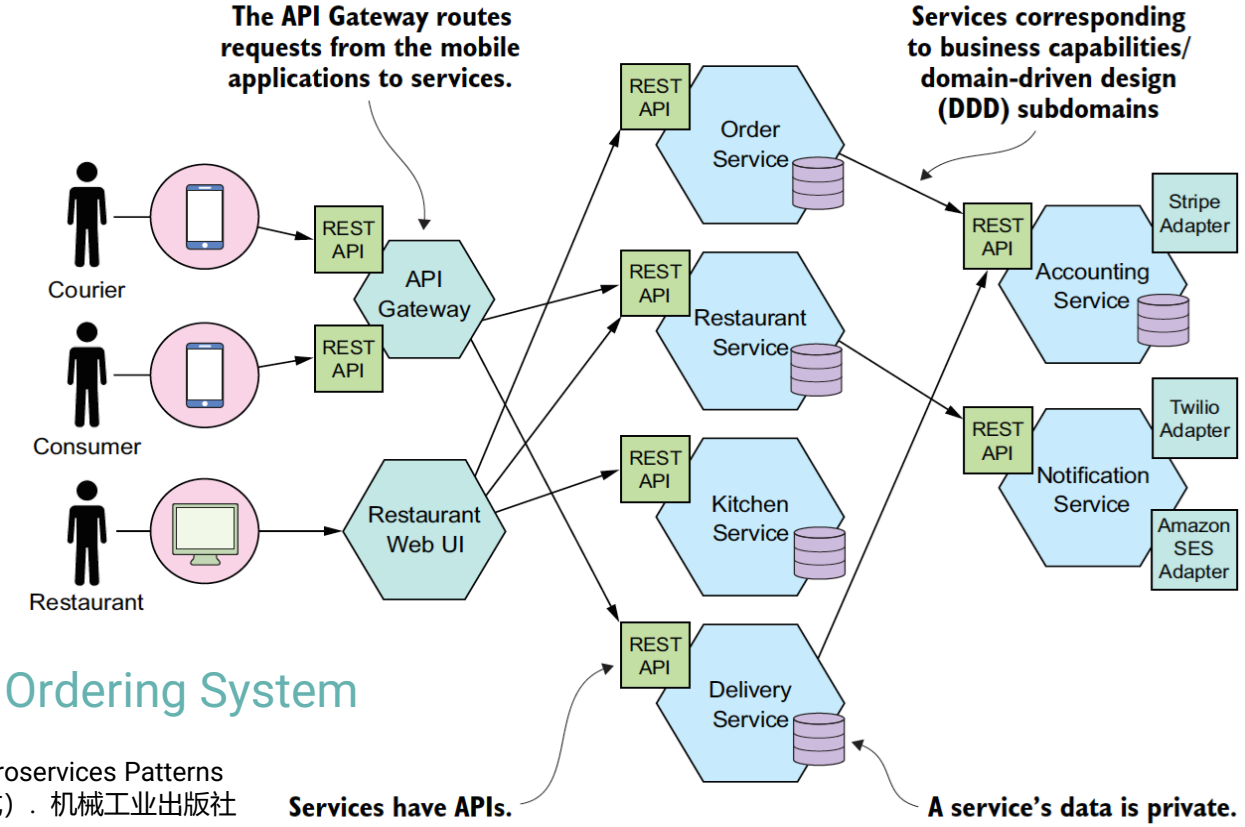
Chris Richardson. Microservices Patterns（微服务架构设计模式）. 机械工业出版社

Monolithic architecture

- The whole application is packaged into a single executable
- Less flexible for large team/code base
- Difficult to scale, wasting deployment resources

MICROSERVICES

Cloud-native applications consist of multiple small, interdependent services called microservices.



Online Food Ordering System

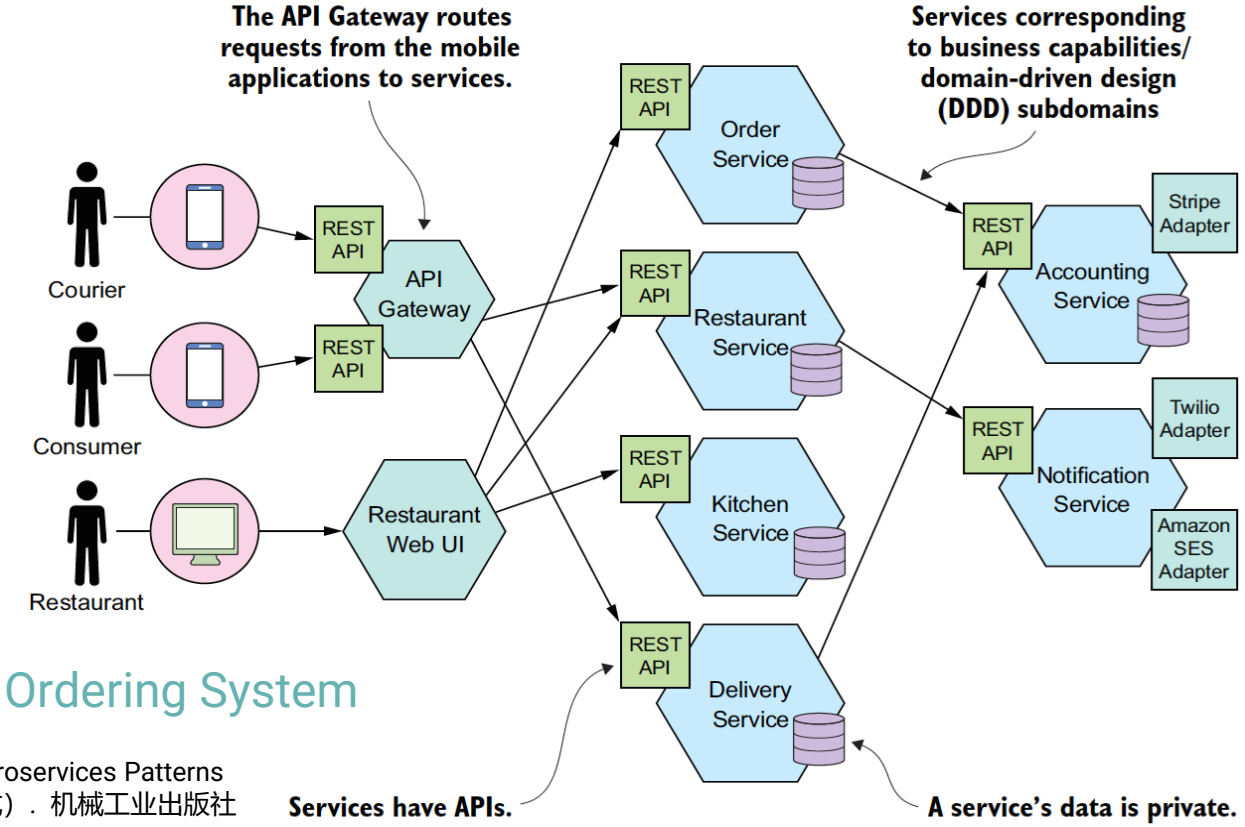
Chris Richardson. Microservices Patterns（微服务架构设计模式）. 机械工业出版社

Microservice Architecture

- Microservices allow a large application to be separated into smaller independent parts, with each part having its own responsibility
- Each service can be developed, managed, and deployed independently.

MICROSERVICES

Cloud-native applications consist of multiple small, interdependent services called microservices.



Microservice Architecture

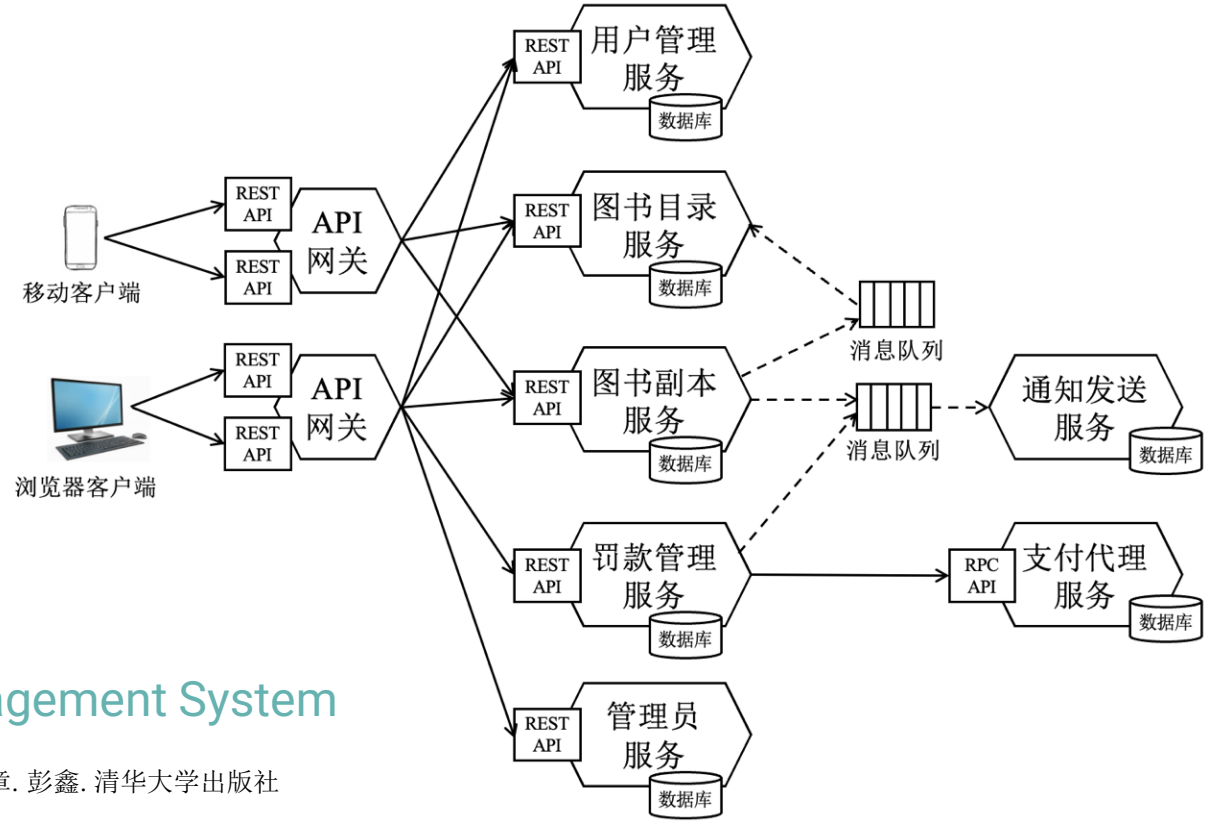
- Services communicate with each other by using well-defined APIs (e.g., REST)
- We could scale only the required microservice

Online Food Ordering System

Chris Richardson. Microservices Patterns（微服务架构设计模式）. 机械工业出版社

MICROSERVICES

Cloud-native applications consist of multiple small, interdependent services called microservices.



Library Management System

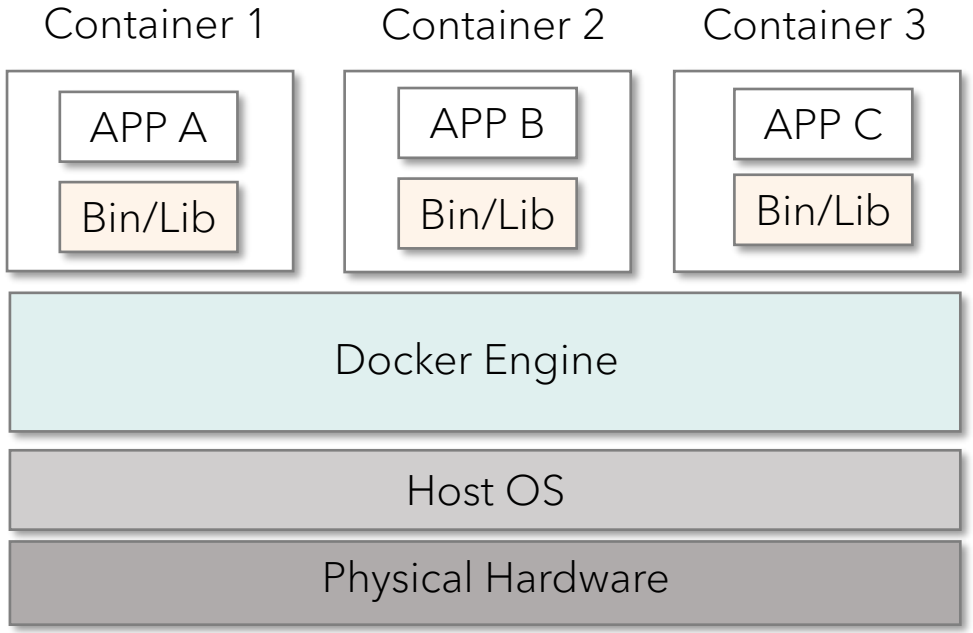
现代软件工程基础,第7章. 彭鑫. 清华大学出版社

Microservice Architecture

- Services communicate with each other by using well-defined APIs (e.g., REST)
- We could scale only the required microservice

CONTAINERS

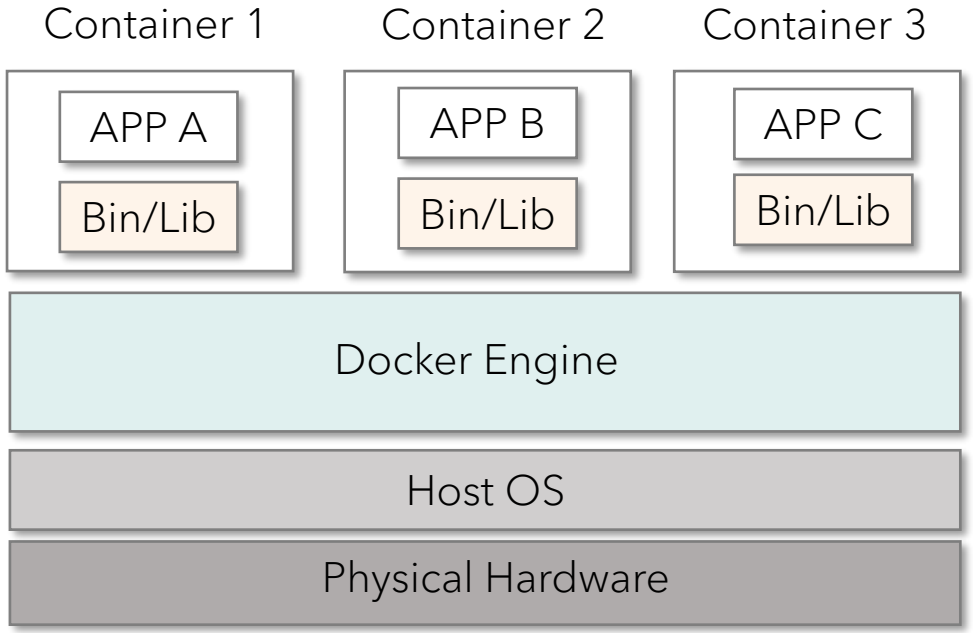
A container is a lightweight and portable unit of software that packages an application and its dependencies together in a single environment.

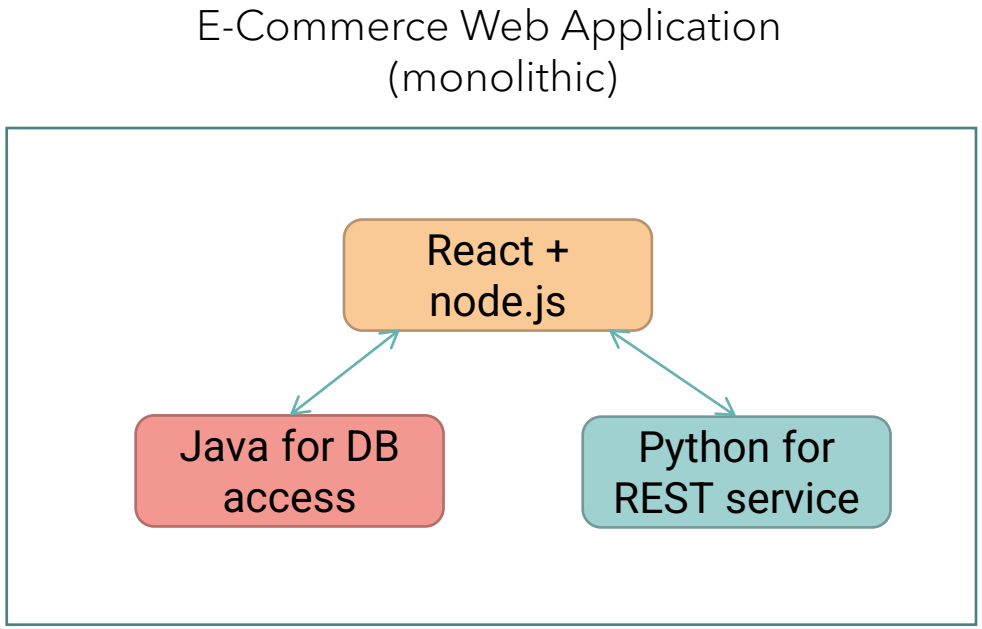


- A container consists of all the dependencies required to run an application, and isolates these dependencies from other containers on the same machine
- Developers can easily create, deploy, and run applications in a consistent and predictable manner across different servers or cloud platforms.

CONTAINERS

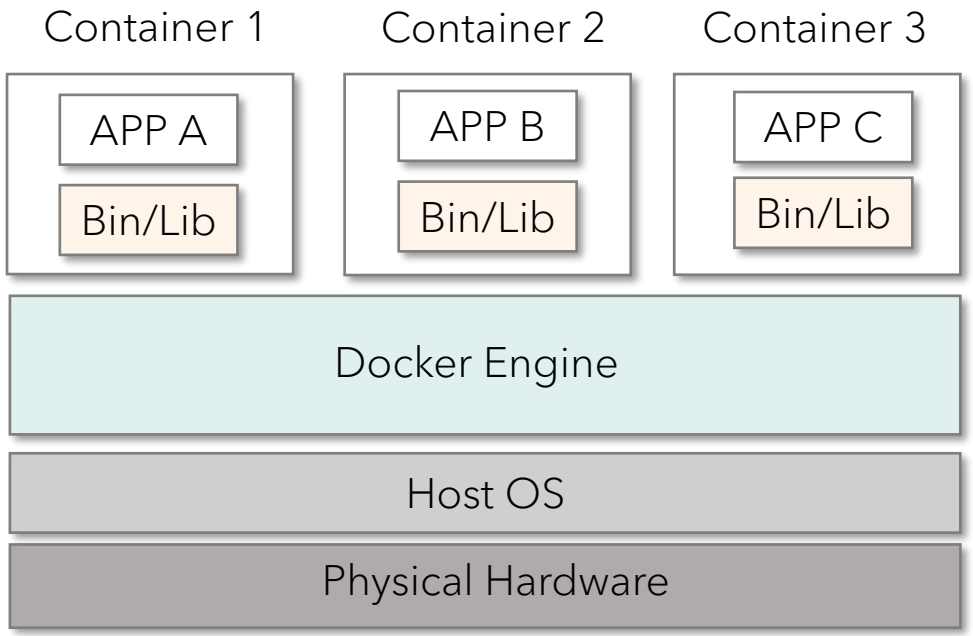
A container is a lightweight and portable unit of software that packages an application and its dependencies together in a single environment.





CONTAINERS

A container is a lightweight and portable unit of software that packages an application and its dependencies together in a single environment.

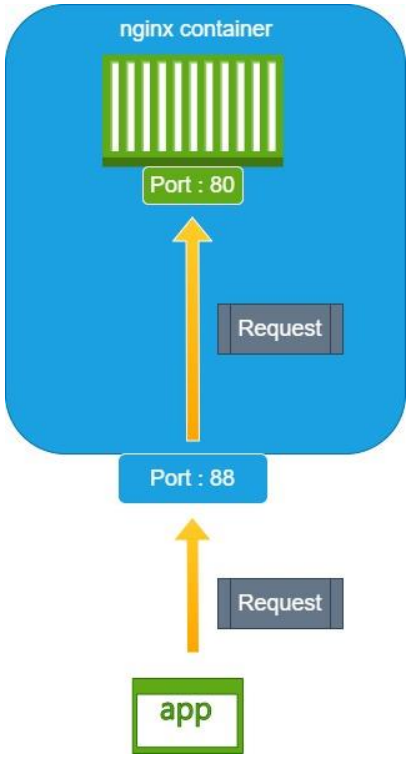


Container Communication

- Containers are assigned to unique IP addresses so that they could communicate with each other using standard network protocols (e.g., TCP/IP)

CONTAINERS

A container is a lightweight and portable unit of software that packages an application and its dependencies together in a single environment.



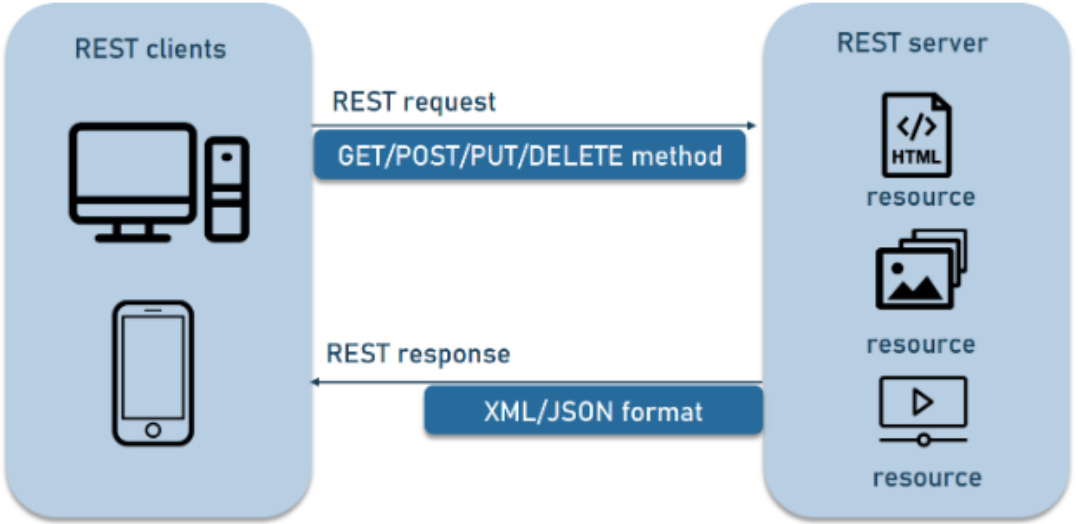
Container Communication

- Containers are assigned to unique IP addresses so that they could communicate with each other using standard network protocols (e.g., TCP/IP)
- Containers could expose **services** to the outside world by mapping container's port to the port on the host machine (port forwarding)

CONTAINERS - SERVICES

REST Services

- RESTful systems use HTTP verbs to manipulate resources identified by URIs (Uniform Resource Identifiers).
- RESTful APIs are stateless and cacheable, which makes them highly scalable and easy to manage.
- Clients can manipulate resources through simple and standardized interfaces.



https://www.altexsoft.com/blog/rest-api-design/

CONTAINERS - SERVICES

REST Services

- RESTful systems use HTTP verbs to manipulate resources identified by URIs (Uniform Resource Identifiers).
- RESTful APIs are stateless and cacheable, which makes them highly scalable and easy to manage.
- Clients can manipulate resources through simple and standardized interfaces.

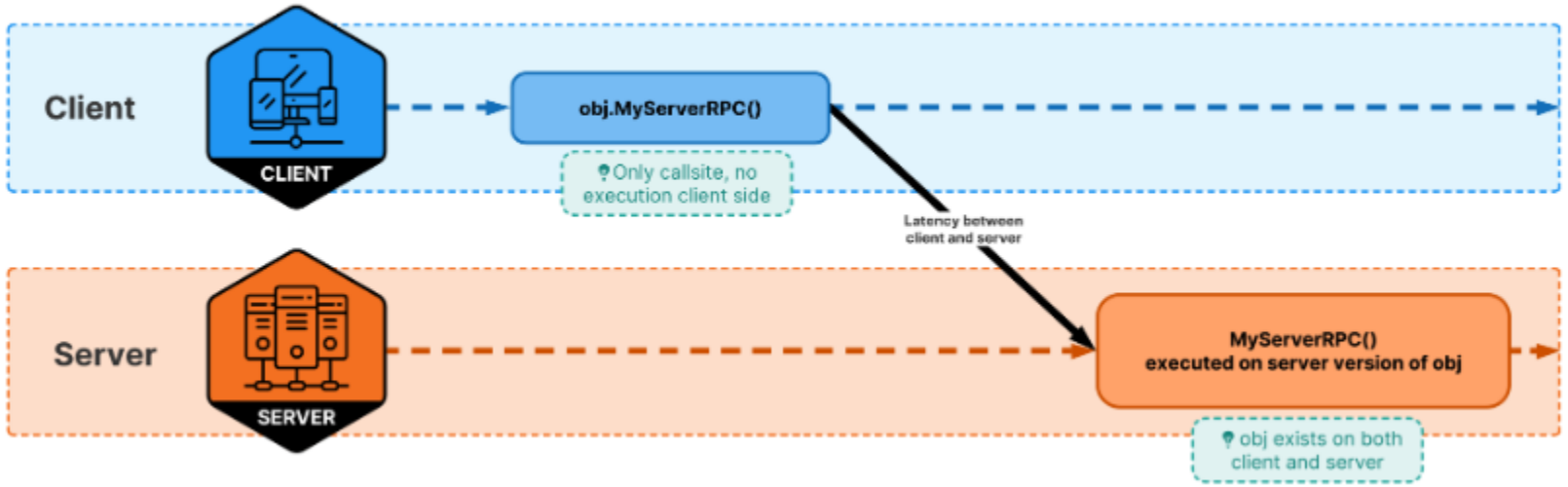
Sample API endpoints for an E-commerce application

`GET /products` - returns a list of all products
`GET /products/{id}` - returns details of a specific product
`POST /orders` - creates a new order
`GET /orders/{id}` - returns details of a specific order
`POST /customers` - creates a new customer
`GET /customers/{id}` - returns details of a specific customer

CONTAINERS - SERVICES

RPC Services

- RPC is a mechanism for invoking a function or method on a remote server, and receiving the results back.
- When calling an RPC, you call a method remotely on an object that can be anywhere in the world (i.e., RPC methods execute on a different machine)



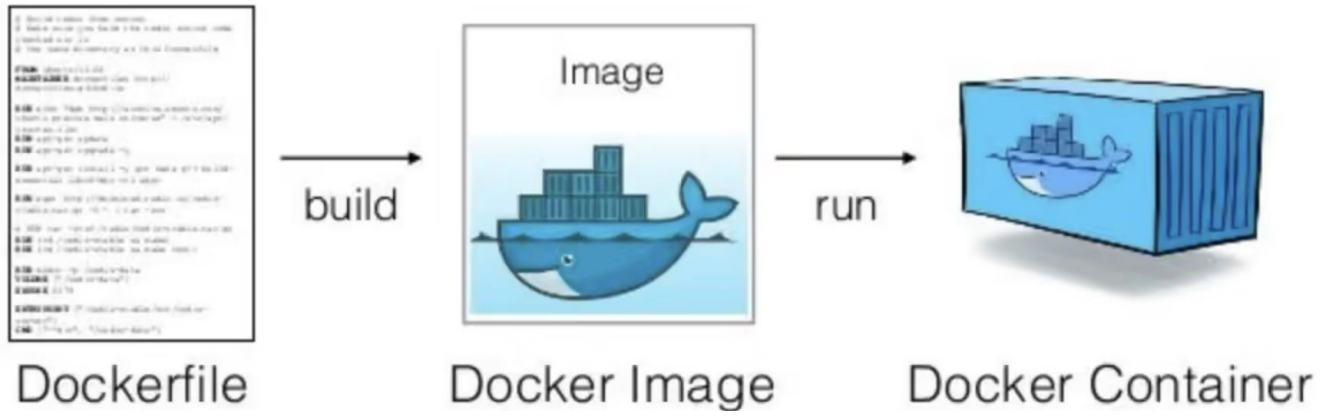
https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/index.html

DOCKER

Docker is one of the most popular platforms for building, deploying, and managing containers

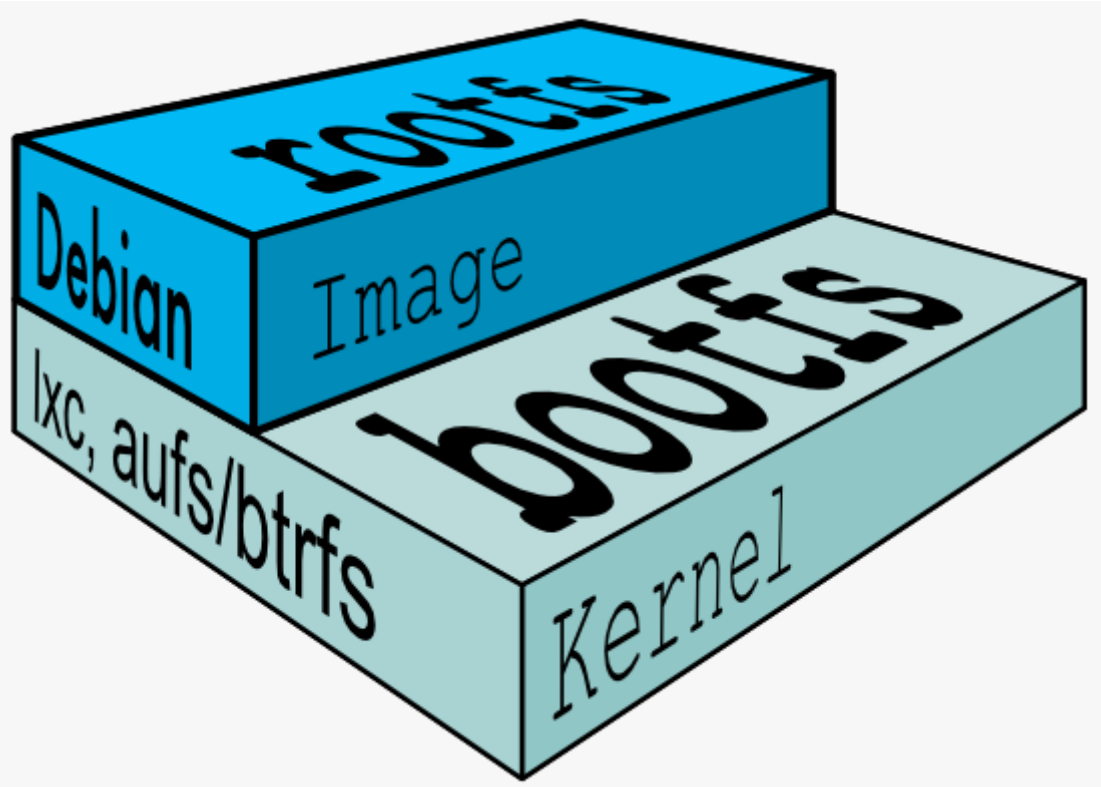
Docker image (镜像): a read-only template that acts as a set of instructions to create a container.

Dockerfile: a script that defines the instructions for building the image



Docker container: a running instance of a Docker image

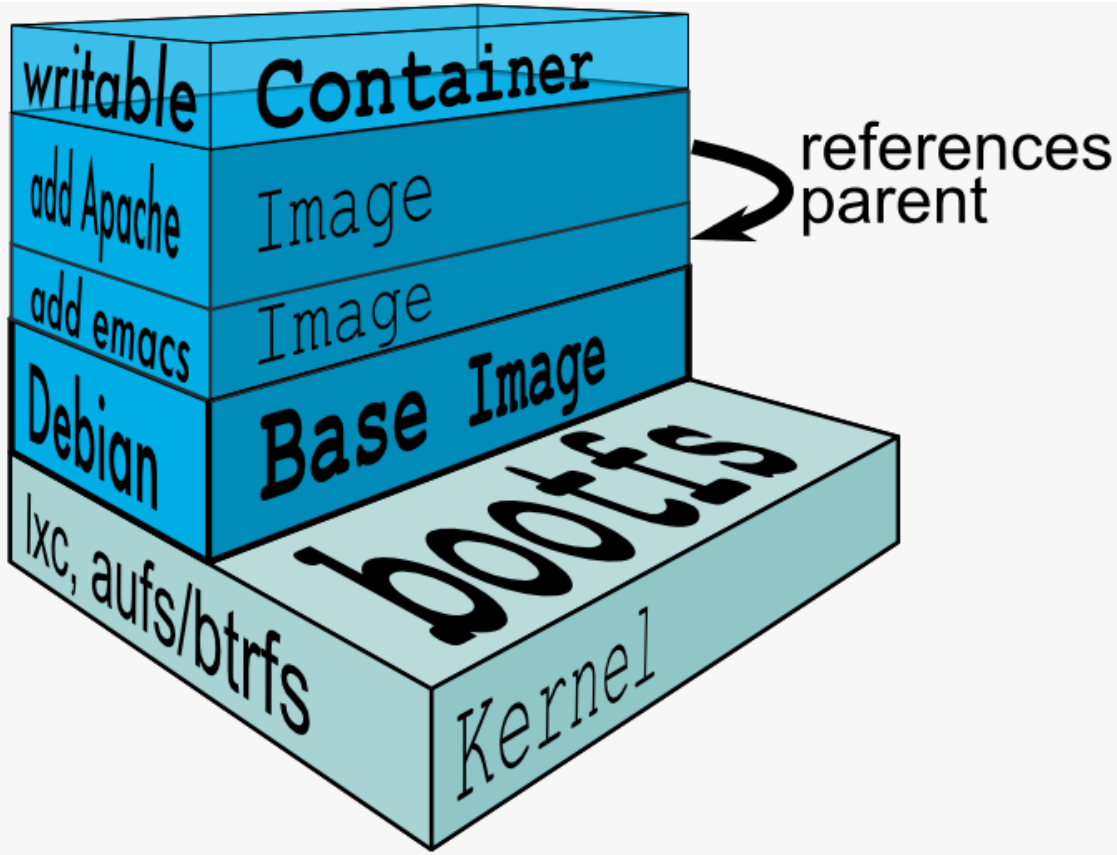
DOCKER IMAGES



- Docker images consist of many layers. Each layer is built on top of another layer to form a series of intermediate images

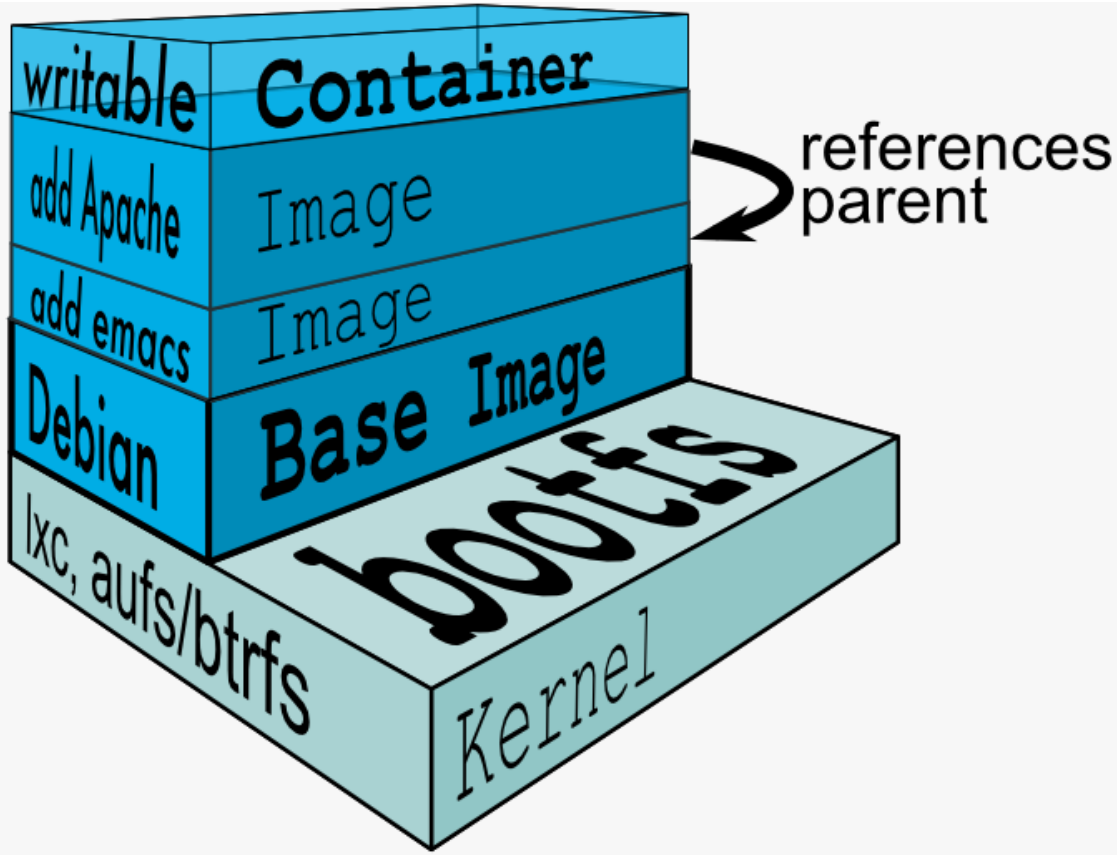
- Kernel: Host machine's kernel
- Base image:
  - Different Linux distribution, e.g., Debian, Ubuntu
  - Various pre-installed software images, e.g., OpenJDK
  - Or a blank image with nothing included

DOCKER IMAGES



- Images (read-only) can be layered on top of one another, specified by Dockerfile
- Each time Docker launches a container from an image, it adds a thin writable layer, known as the container layer, which stores all changes to the container throughout its runtime.

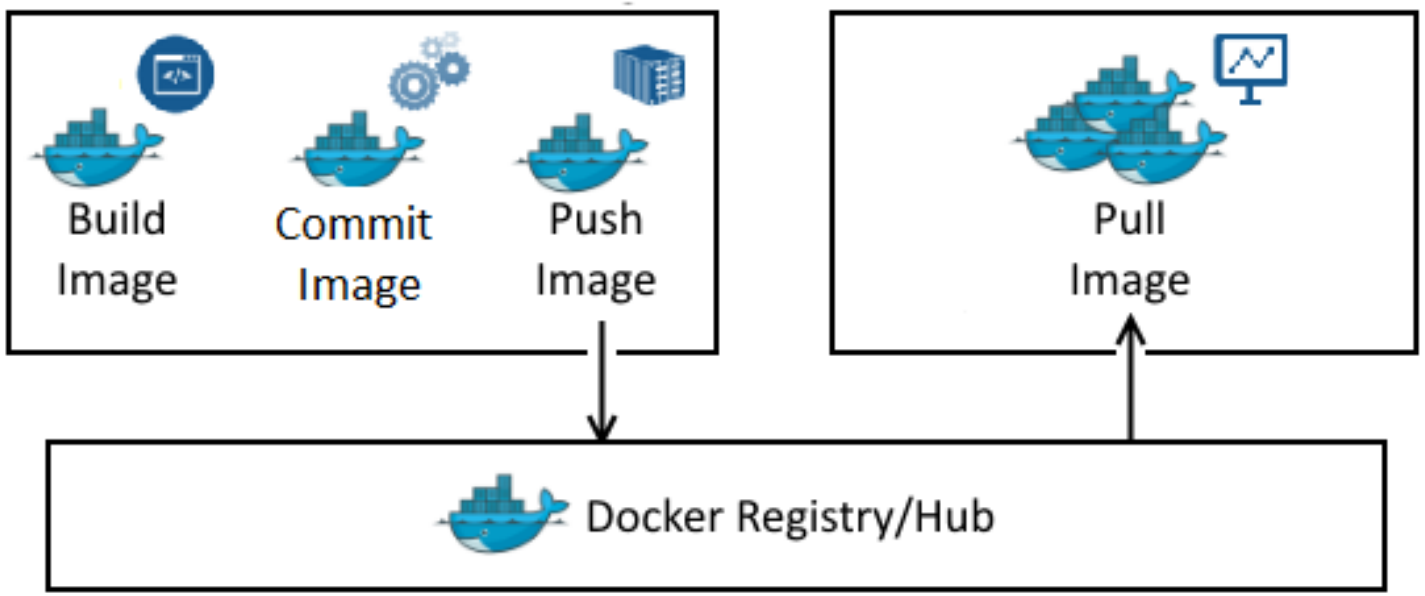
DOCKER IMAGES



- The top layer has read-write permissions, and all the remaining layers have read-only permissions
- Several containers may share access to the same underlying level of a Docker image, but write the changes locally and uniquely to each other

DOCKER IMAGES

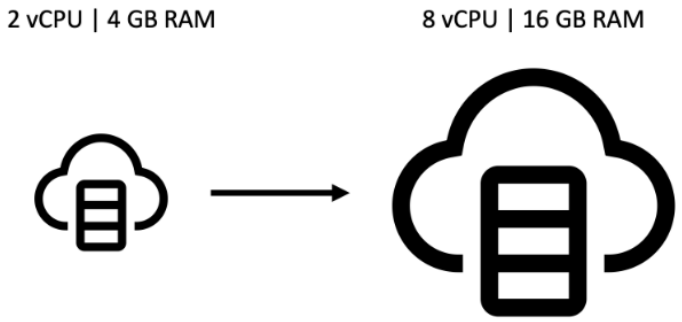
Docker image could be shared as we did for source code version control



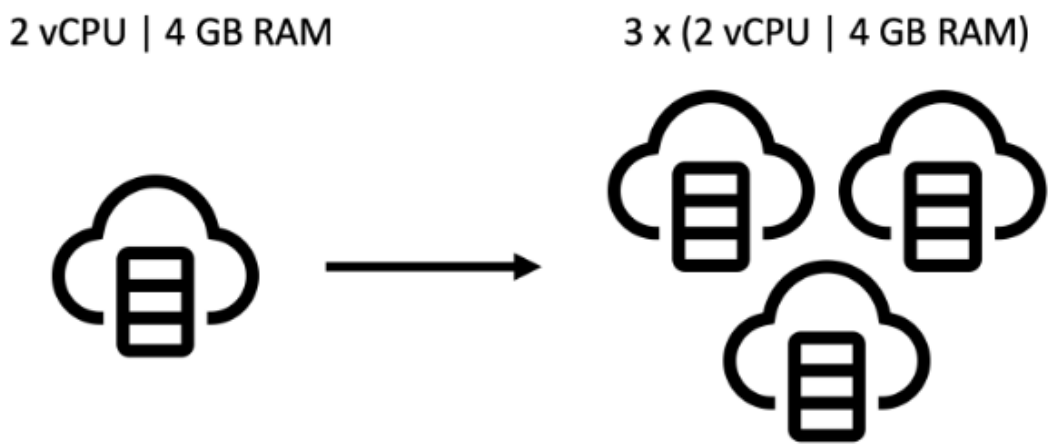
SCALE

Our application becomes much more popular. Many more users start to us our application

Scale up (Vertical scaling)



Scale out (Horizontal scaling)

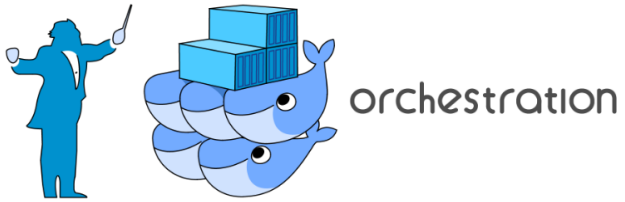


https://developer.ibm.com/articles/scale-up-and-scale-out-vms-vs-containers/

SCALE OUT - COMPLEXITIES

- Load Balancing: With more servers added, it's important to distribute incoming traffic across these servers to ensure that the load is evenly distributed
- Configuration: each new server or node needs to be configured properly to work with the rest of the infrastructure (e.g., network config, security config). As the number of servers grows, the complexity of configuring and managing them can increase.
- Monitoring/Troubleshooting: With more servers in the infrastructure, it becomes more difficult to monitor the system and identify issues.
- Maintenance: With more servers, there is more hardware and software that needs to be maintained and updated, which increases the workload and complexity of managing the infrastructure.

CONTAINER ORCHESTRATION



Use container orchestration to **automate and manage** tasks such as:

- Deployment
- Configuration
- Scheduling
- Resource allocation
- Container availability
- Load balancing and traffic routing
- Scaling or removing containers based on balancing workloads across your infrastructure
- Monitoring container health
- Keeping interactions between containers secure

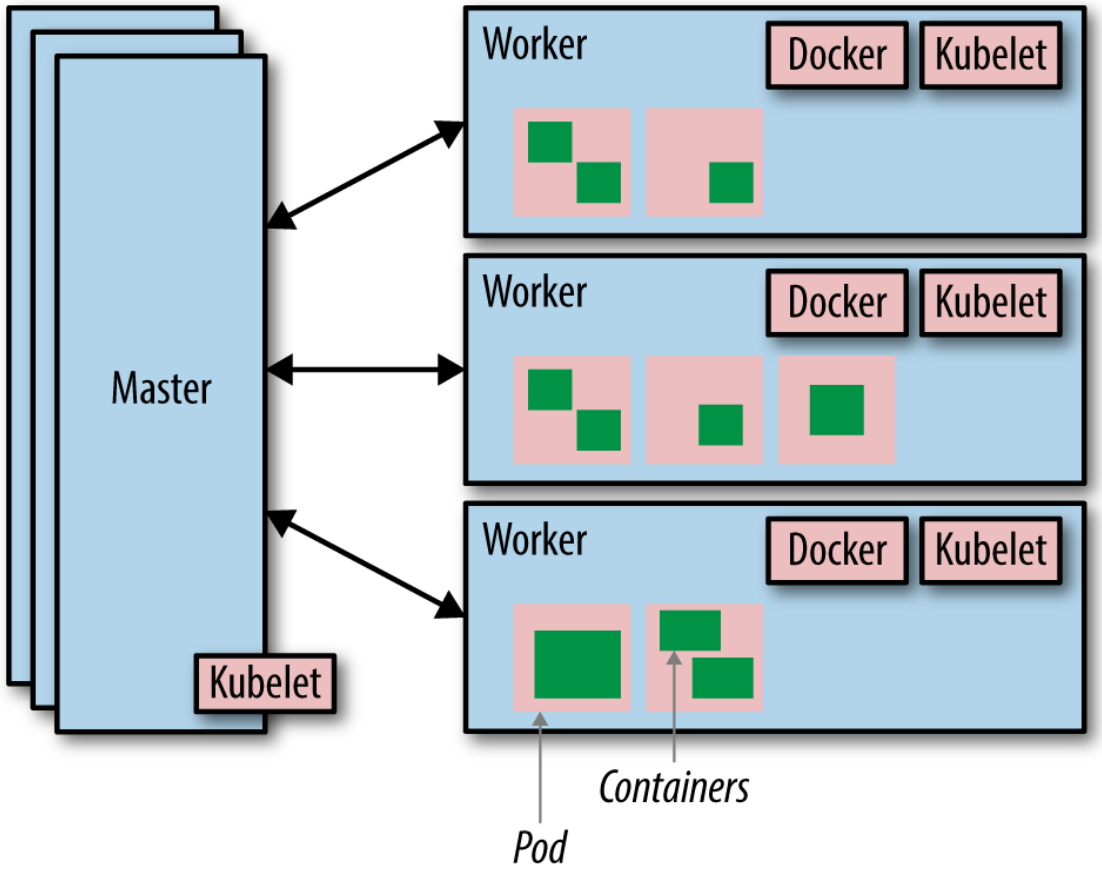
CONTAINER ORCHESTRATION TOOLS

Docker Swarm and Kubernetes (K8s) are both container orchestration tools that allow for the management and deployment of containerized applications **at scale**

kubernetes

KUBERNETES ARCHITECTURE

https://enabling-cloud.github.io/kubernetes-learning/

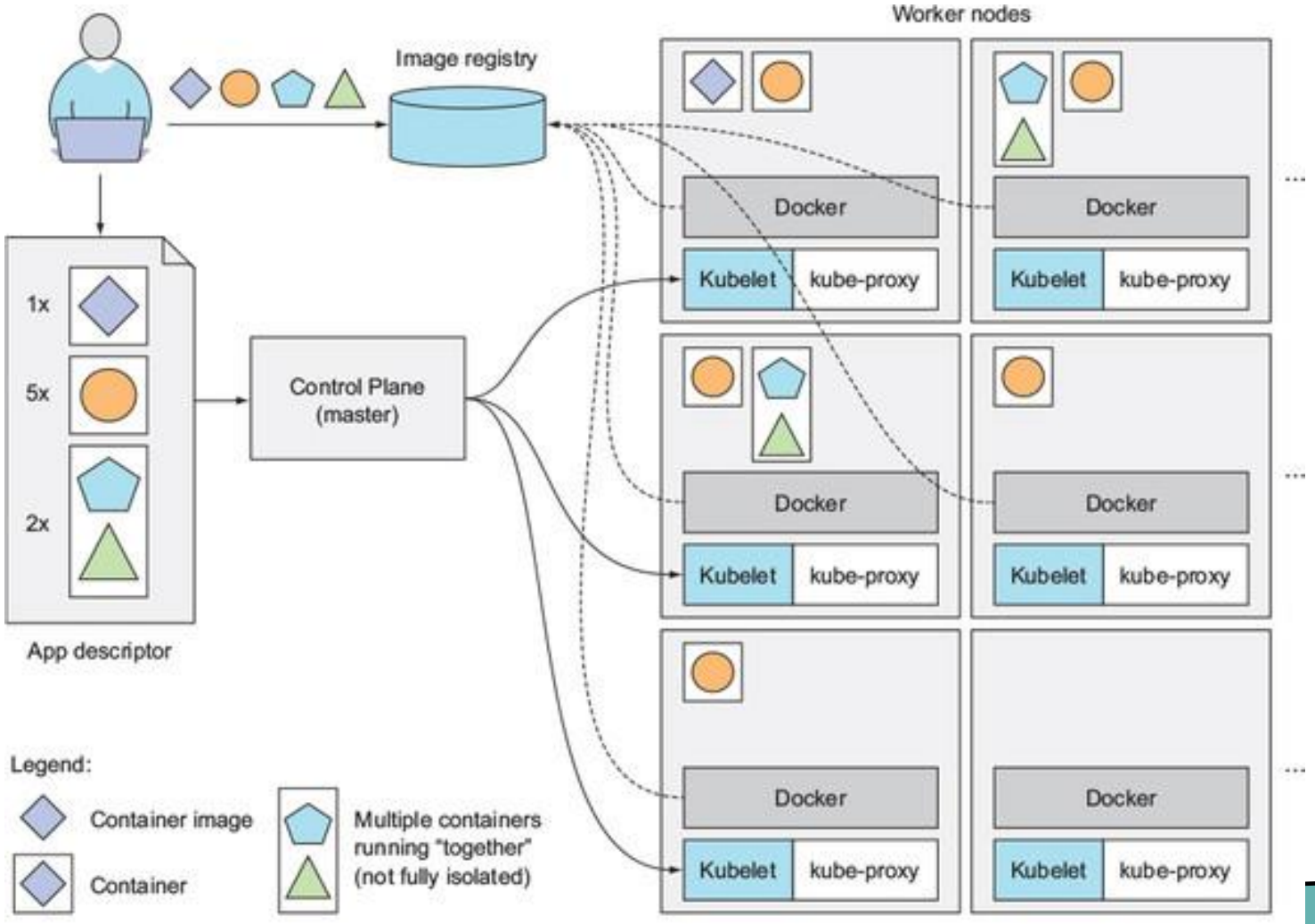


Cluster: A control plane and one or more compute machines, or worker nodes.

- **Control plane (master)**: The collection of processes that control k8s nodes. This is where all task assignments originate.
- **Worker nodes**: Worker nodes within the k8s cluster are used to run containerized applications
  - **Kubelet**: This service runs on nodes and reads the container manifests and ensures the defined containers are started and running.=
  - **Pod**: A group of one or more containers deployed to a single node. All containers in a pod share an IP address, hostname, and other resources.

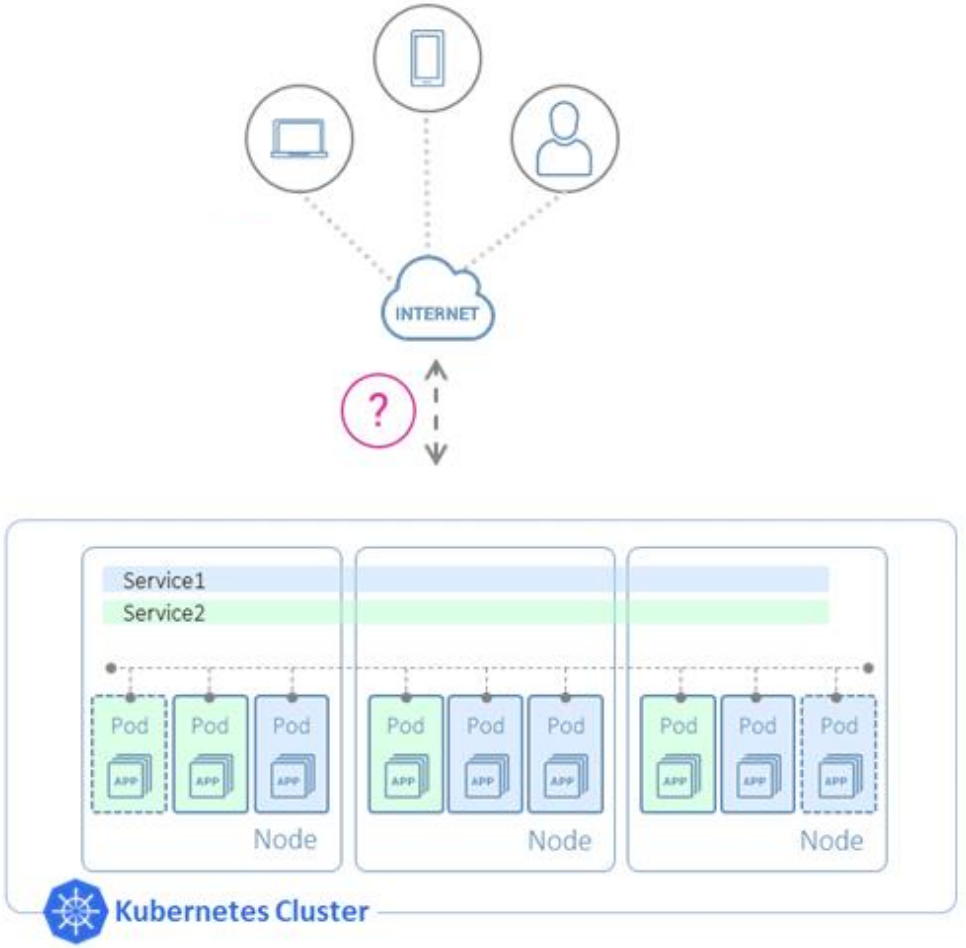
KUBERNETES ARCHITECTURE

https://enabling-cloud.github.io/kubernetes-learning/

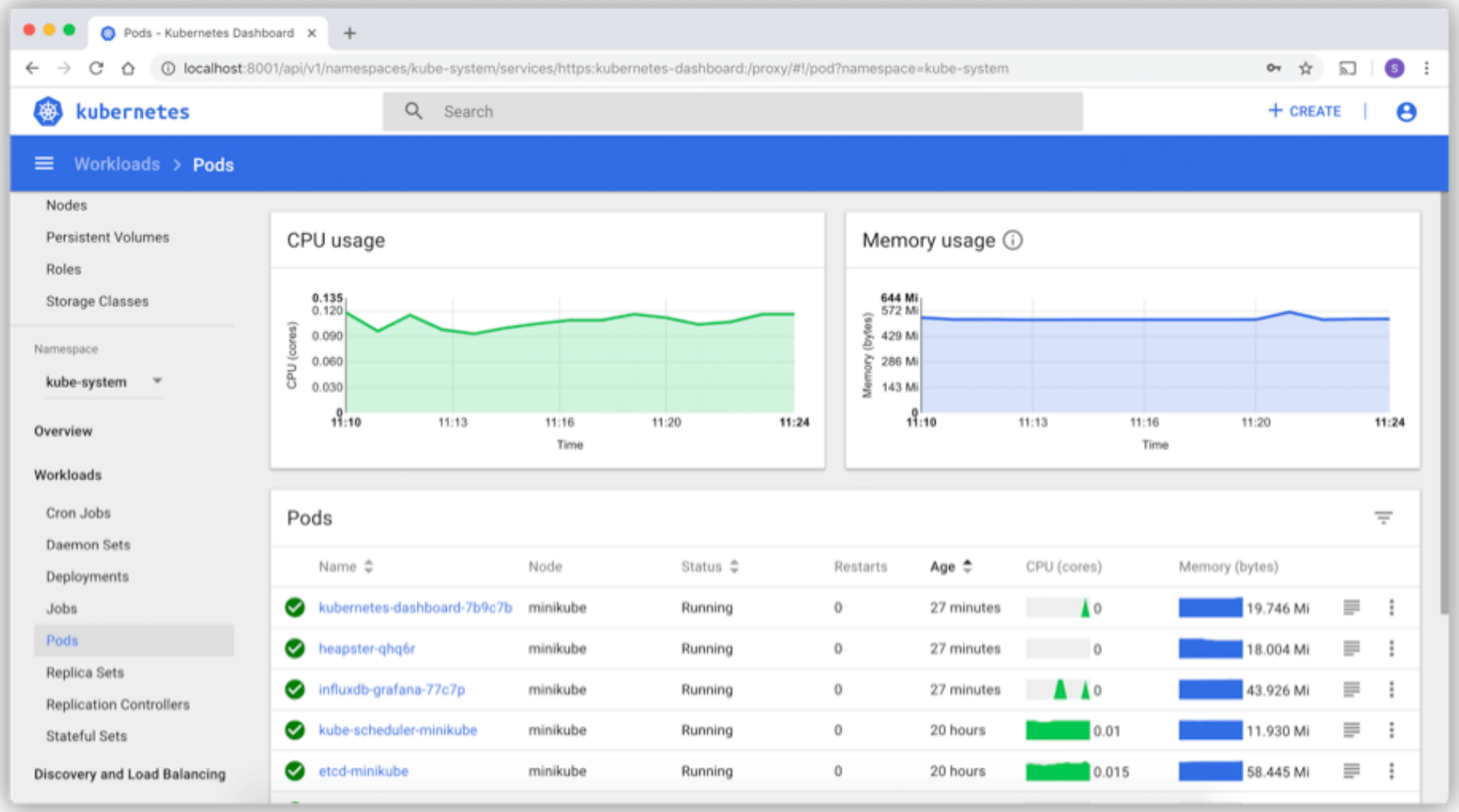


LOAD BALANCING FOR THE K8S CLUSTER

https://www.a10networks.com/blog/load-balancing-traffic-to-applications-in-kubernetes-cluster/

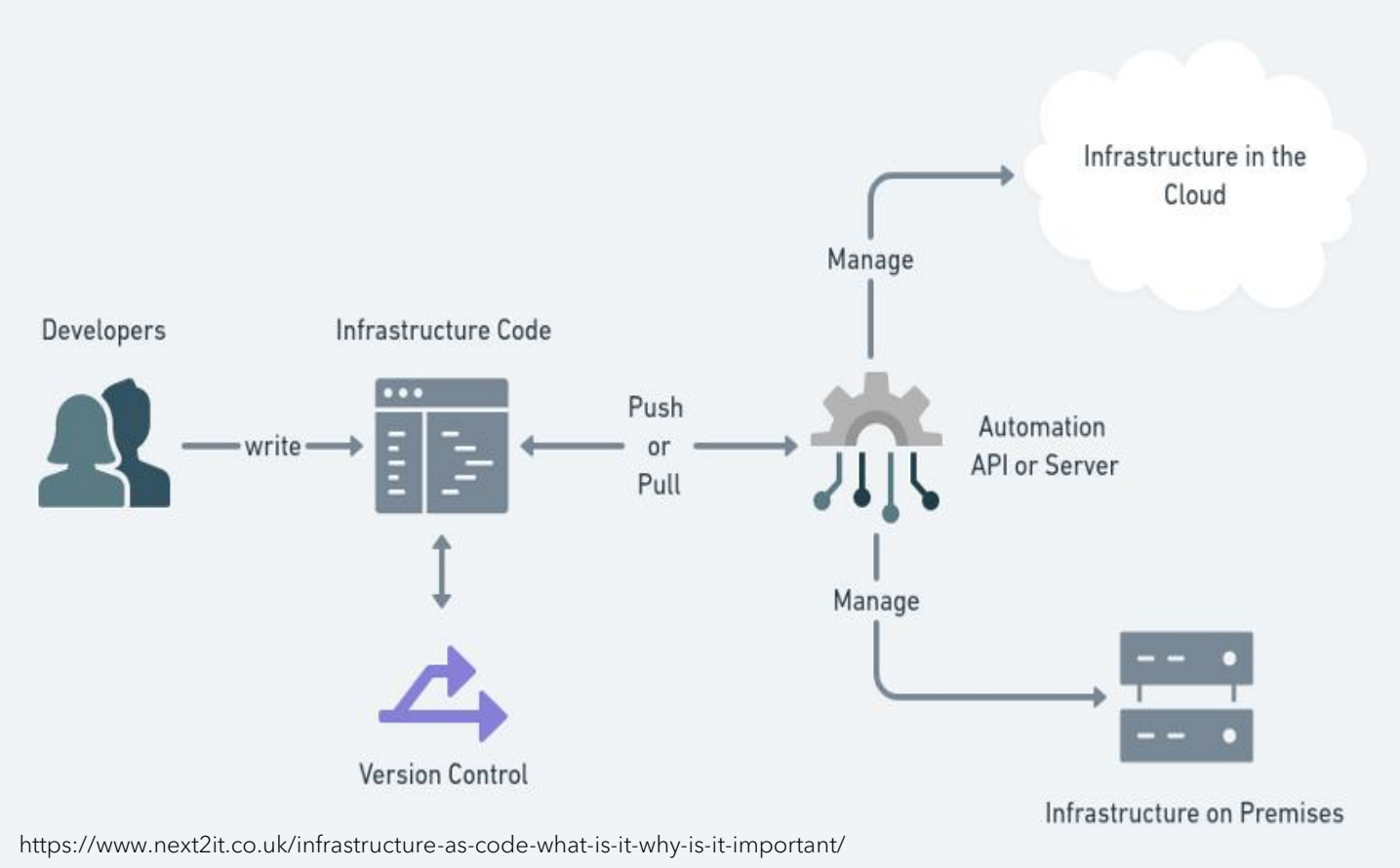


MONITORING THE K8S CLUSTER

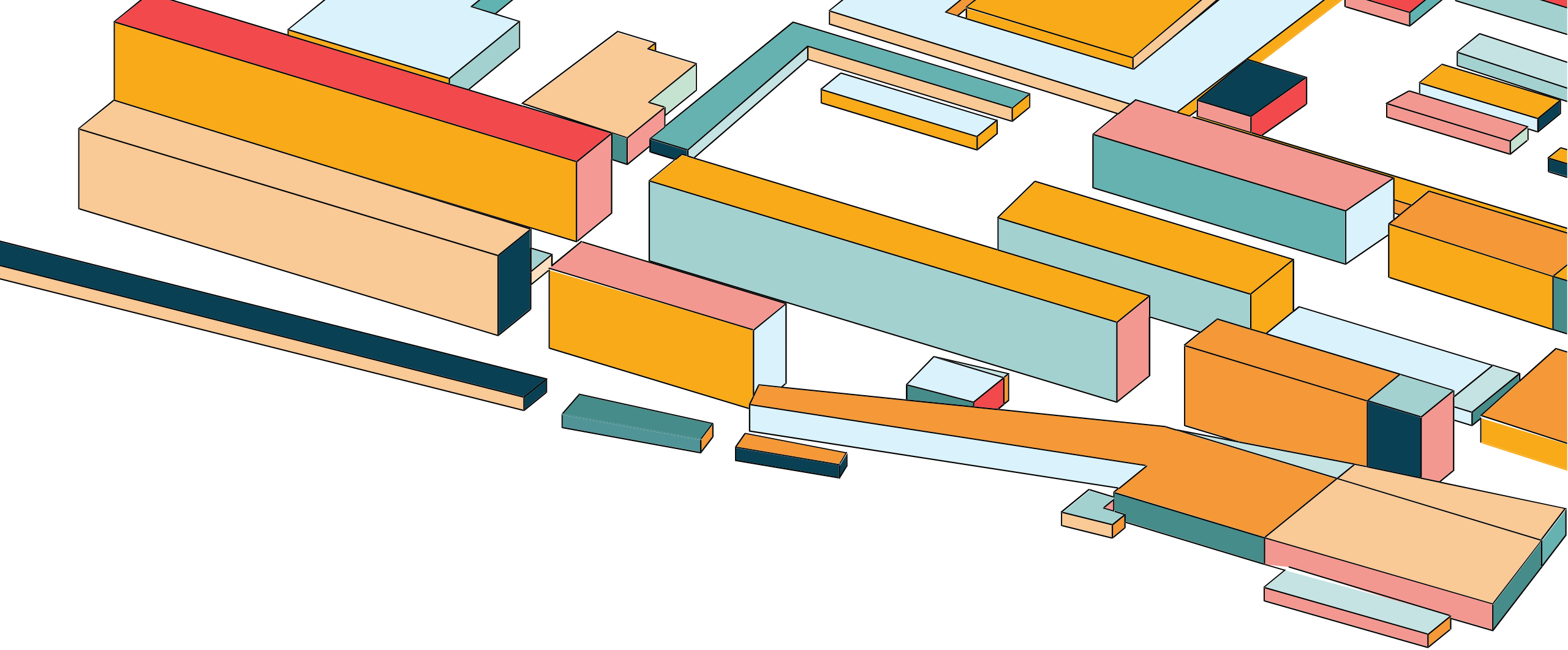


INFRASTRUCTURE AS CODE

- Infrastructure as code (IaC) is DevOps practice that involves managing and provisioning infrastructure resources such as servers, networks, storage using code instead of manual processes.
- IaC code is stored in version control systems such as Git, and then executed using automation tools such as Ansible or Terraform.
- Teams can define and maintain their infrastructure as code, just like software applications, and apply best practices such as version control, testing, and CI/CD.



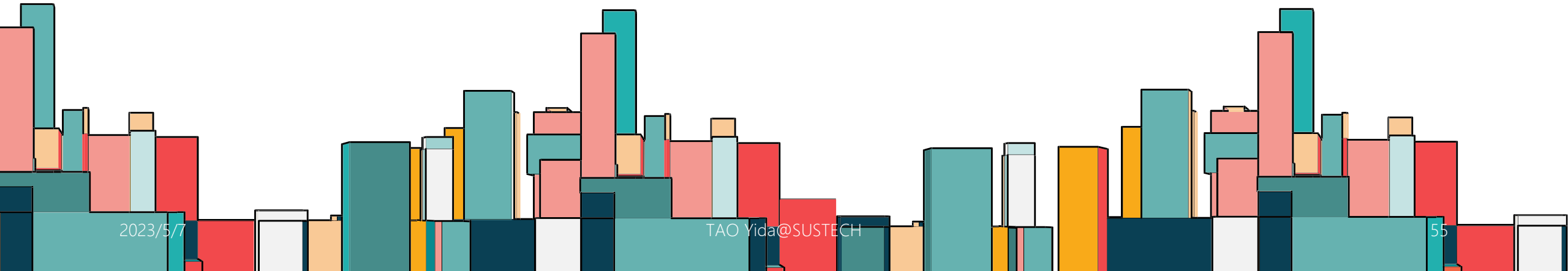
https://www.next2it.co.uk/infrastructure-as-code-what-is-it-why-is-it-important/



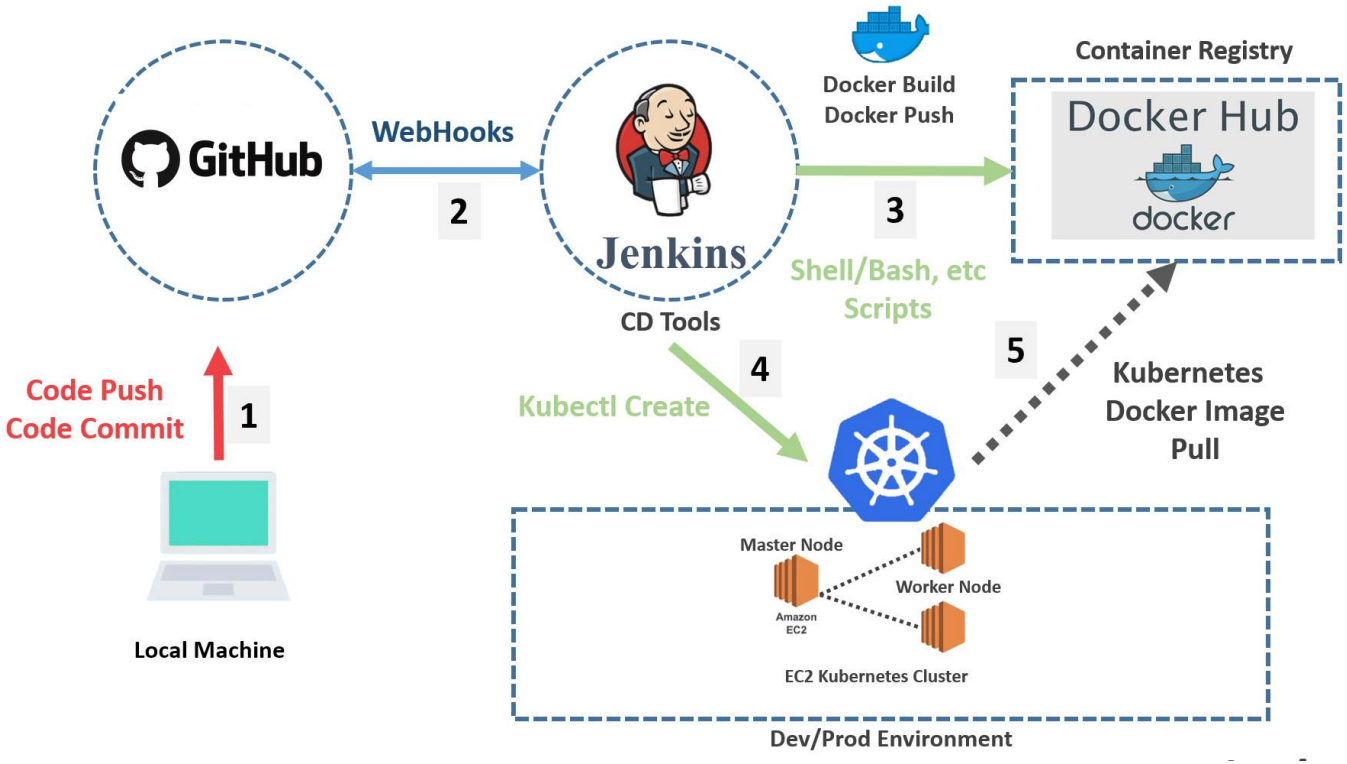
DEPLOYMENT PIPELINES

Jenkins

- Jenkins is an open source automation server.
- It helps automate building, testing, and deploying process
- It facilitates continuous integration and continuous delivery.

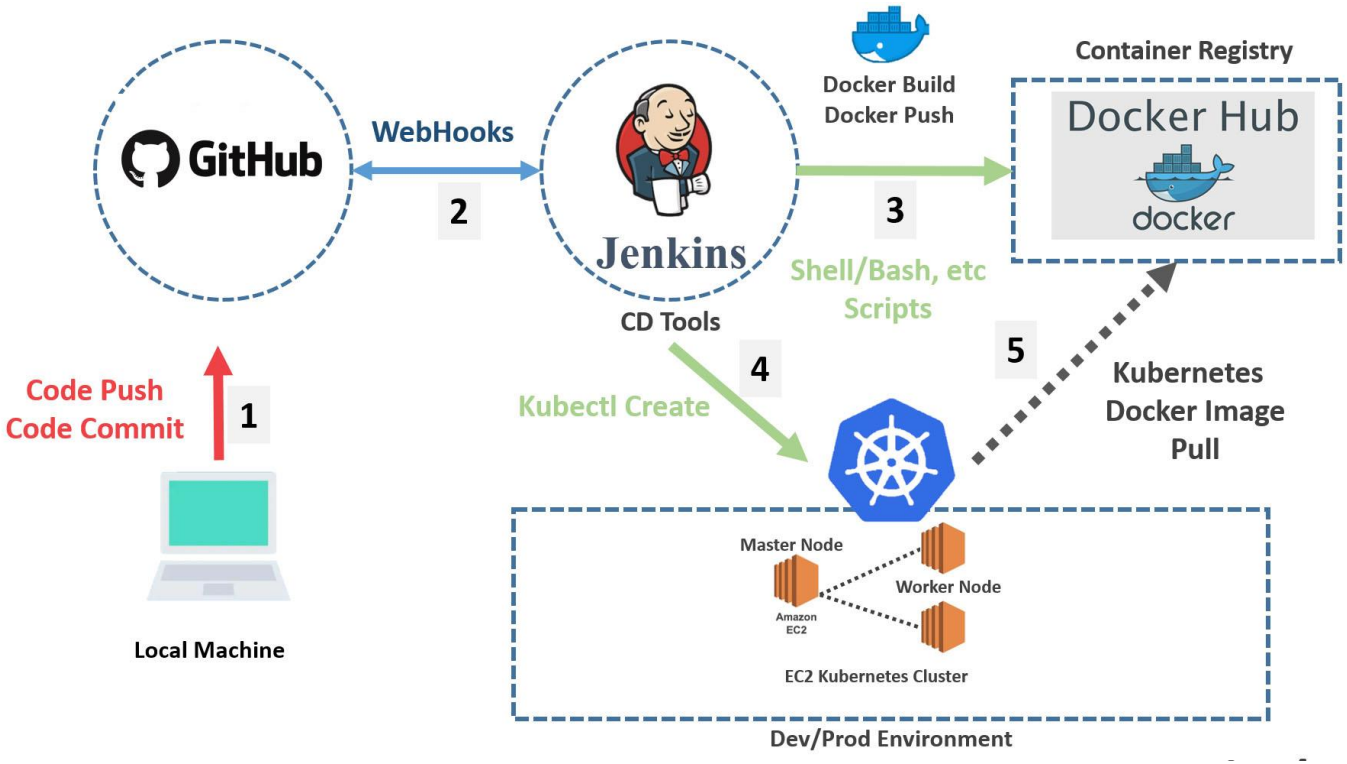


A TYPICAL JENKINS PIPELINE



Step 0: Configure a **webhook** on GitHub, which allows external services like Jenkins to be notified when certain events like push happen.

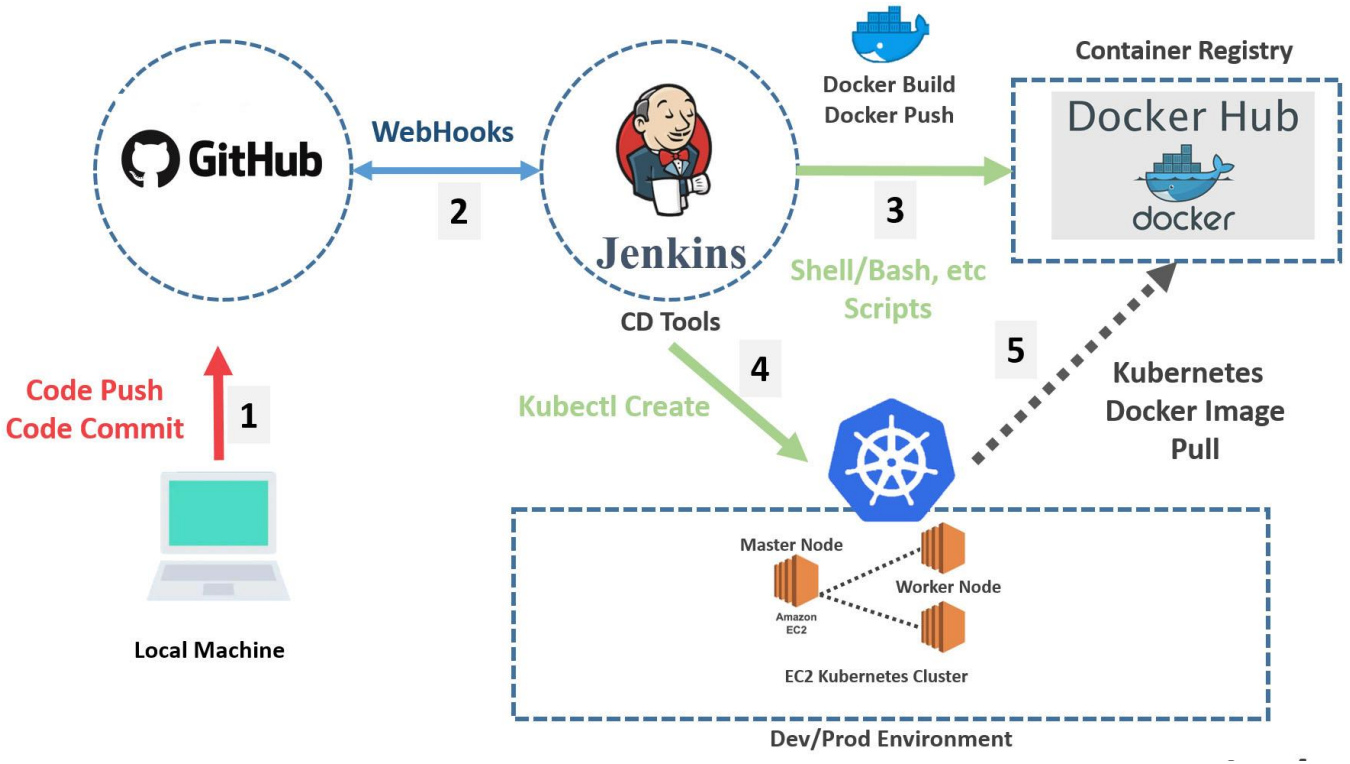
A TYPICAL JENKINS PIPELINE



Step 1: Dev team push local commits to GitHub

Step 2: The push triggers the webhook, and Jenkins will automatically download all the updates from GitHub

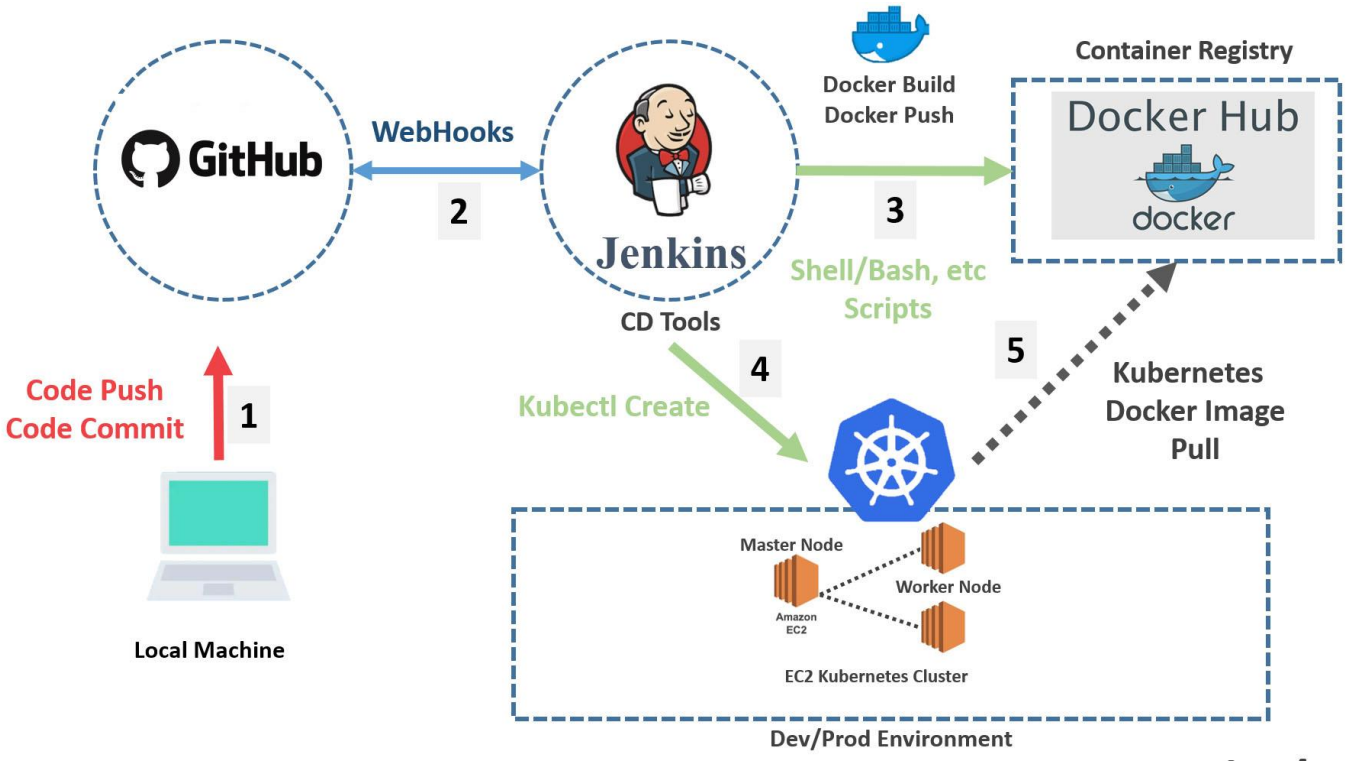
A TYPICAL JENKINS PIPELINE



Step 3: Jenkins could be configured to automatically execute a series of tasks/stages when triggered, for example:

- Build the project with Maven
- Test the project with JUnit
- Build the docker image
- Push the docker image to Docker Hub

A TYPICAL JENKINS PIPELINE



Step 4: Jenkins could also be configured for automatic deployment, e.g., to deploy the software to a K8s cluster

- Step 5: K8s pull docker images from the docker registry

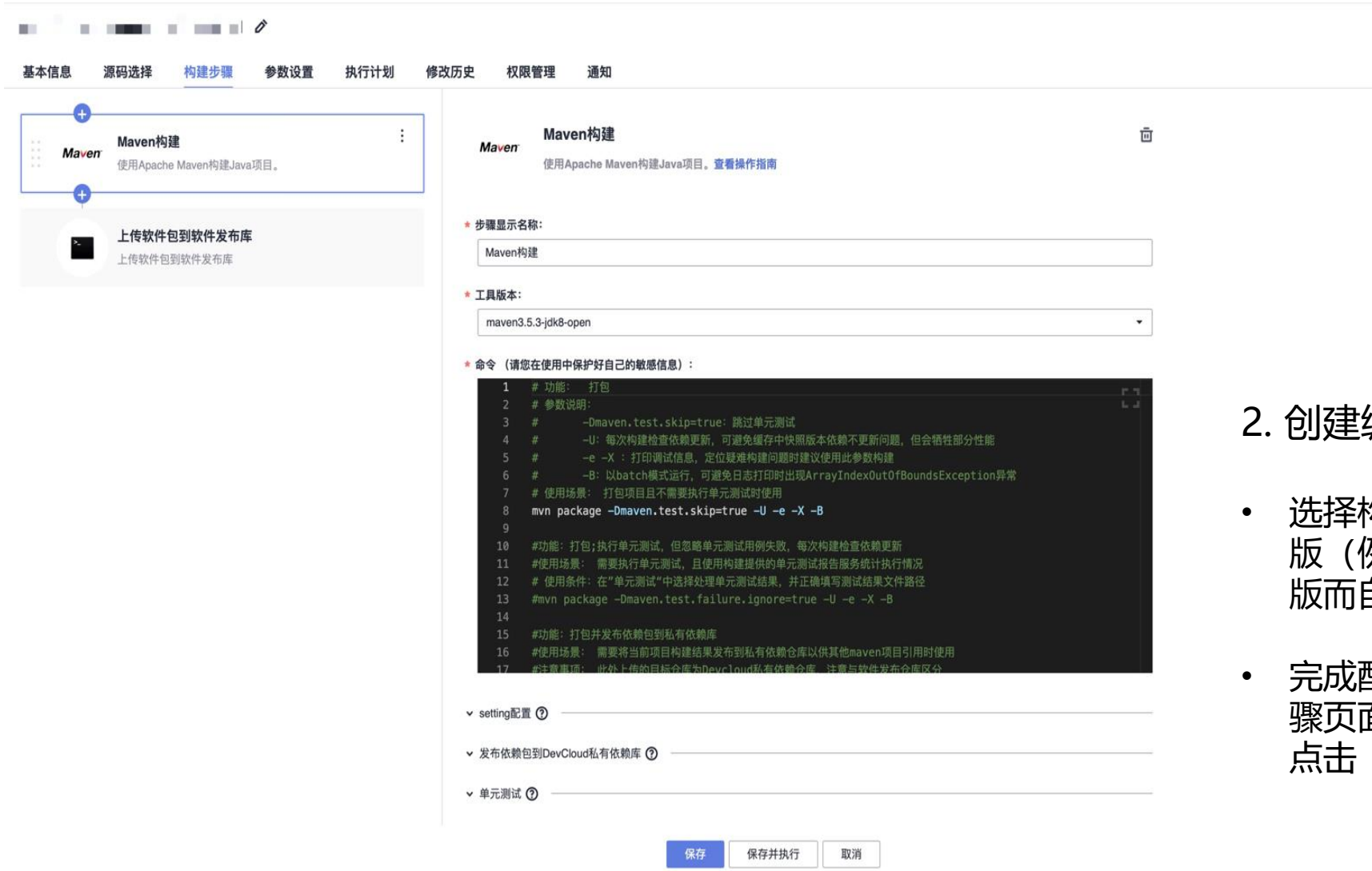
https://www.huaweicloud.com/devcloud/

华为DevCloud/CodeArts部署流水线

1. 创建代码检查(Lint)任务

- 点击“新建任务”，选择需要进行代码检查的代码仓库，点击“创建”
- 待代码检查任务创建成功后，将自动跳转到任务详情页面，然后点击“开始检查”启动任务。任务执行完毕后，就可以查看代码问题列表

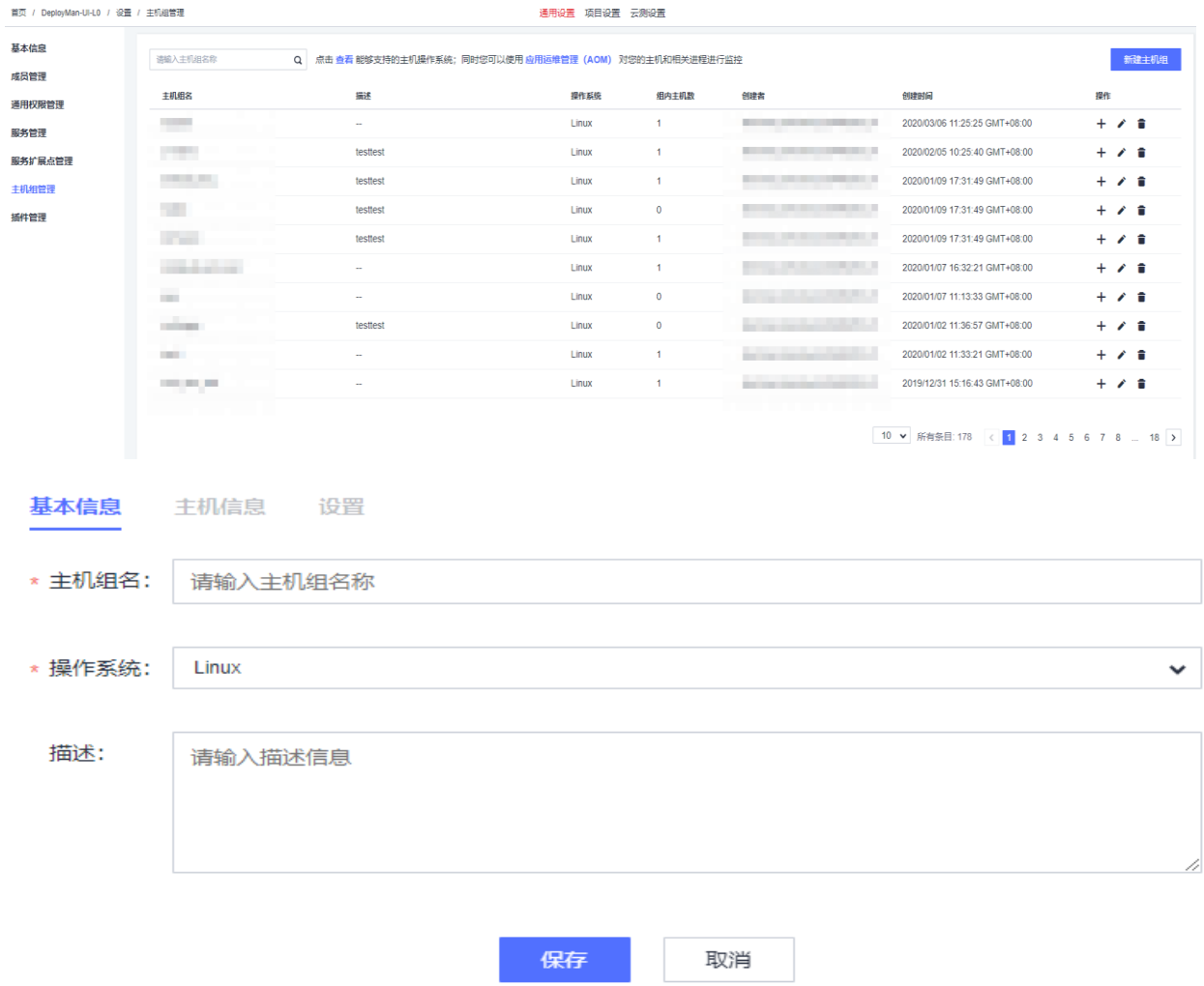
华为DevCloud/CodeArts部署流水线



2. 创建编译构建(Build)任务

- 选择构建模版：根据实际情况选择相应的模版（例如Maven模版），也可以不使用模版而自定义构建步骤
- 完成配置后，点击确定，自动跳转至构建步骤页面，根据情况编辑各步骤中的配置项，点击“新建”

华为DevCloud/CodeArts部署流水线



3. 创建部署(Deploy)任务

- 添加部署服务器/授信主机信息
- 新建部署任务，根据实际情况编辑各步骤中的配置项
- 启动任务，执行成功后可以在页面中查看部署日志，也可以登录主机查看部署结果

华为DevCloud/CodeArts部署流水线

4. 配置流水线

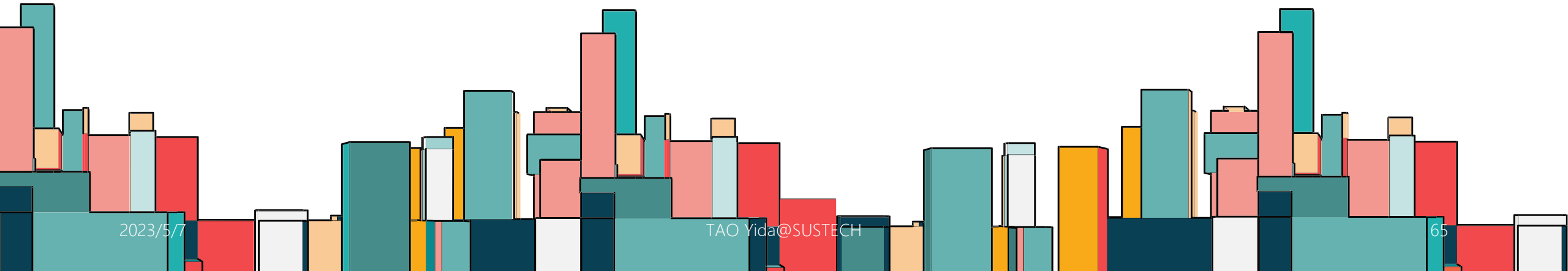
- 配置流水线信息，包括基本信息（即流水线名称）、代码源（即选择代码源、仓库与分支）等
- 添加之前创建的代码检查任务、编译构建任务和部署任务。
- 启动任务，待任务执行成功后可单击各任务查看其详情

腾讯云 | CODING

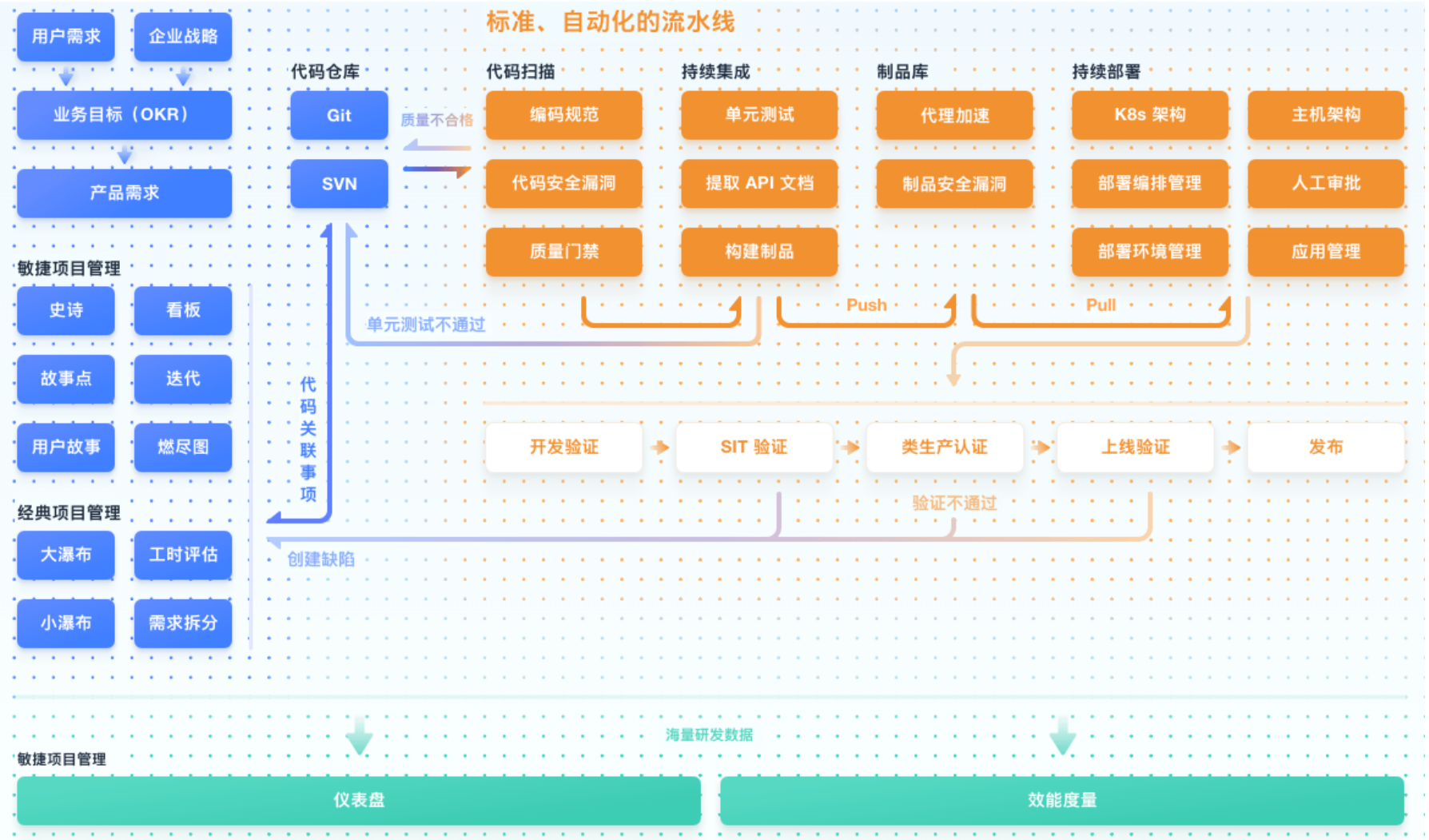
为研发团队打造的数字化软件工厂

提供一站式研发管理平台及云原生开发工具，让软件研发如同工业生产般简单高效，助力企业提升研发管理效能

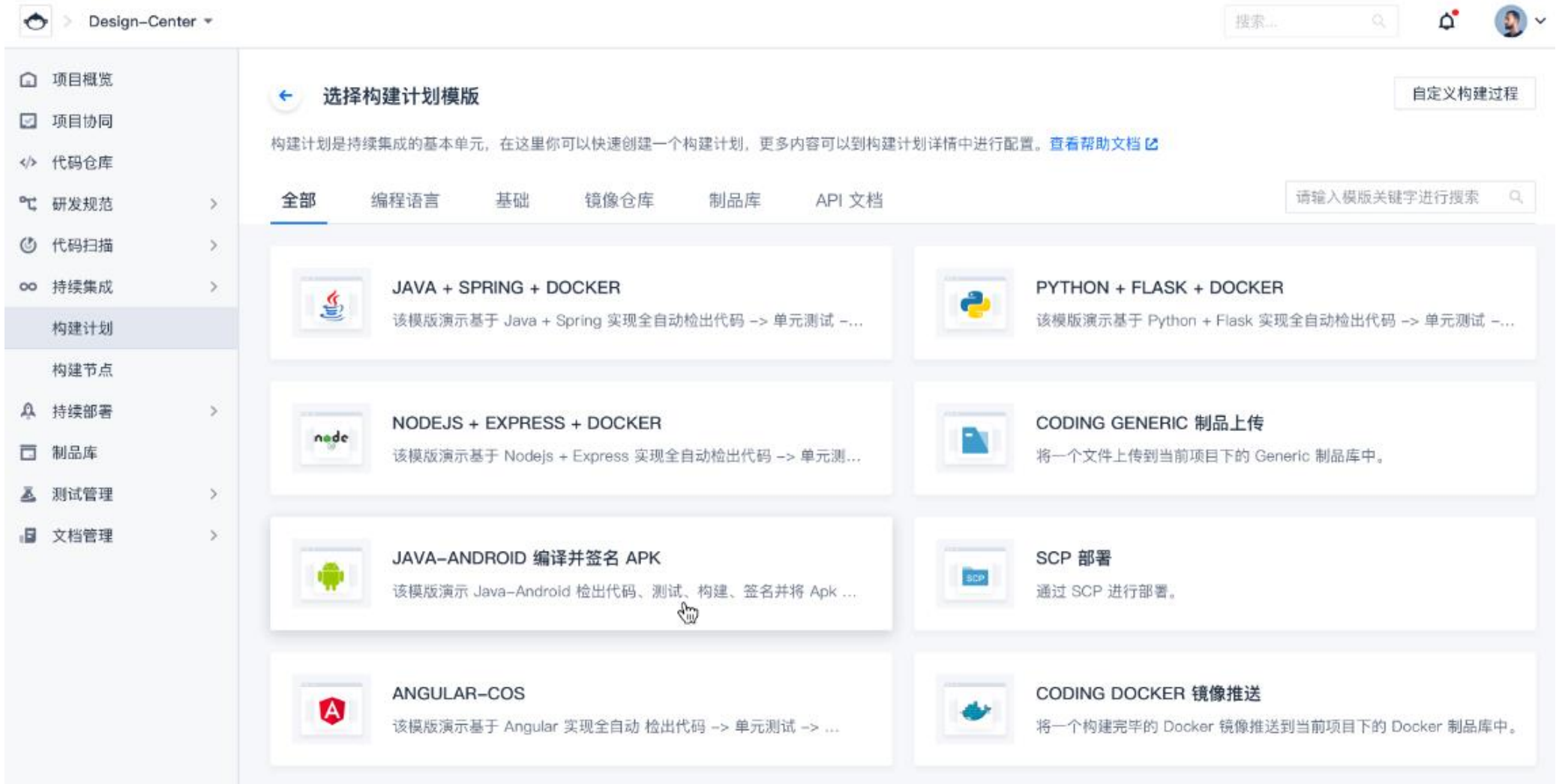
https://coding.net/



腾讯云DevOps平台



CODING 持续集成


Design-Center
项目概览
项目协同
代码仓库
研发规范
代码扫描
持续集成
构建计划
构建节点
持续部署
制品库
测试管理
文档管理
选择构建计划模版
自定义构建过程
构建计划是持续集成的基本单元，在这里你可以快速创建一个构建计划，更多内容可以到构建计划详情中进行配置。查看帮助文档
全部
编程语言
基础
镜像仓库
制品库
API 文档
请输入模版关键字进行搜索
JAVA + SPRING + DOCKER
该模版演示基于 Java + Spring 实现全自动检出代码 -> 单元测试 -...
PYTHON + FLASK + DOCKER
该模版演示基于 Python + Flask 实现全自动检出代码 -> 单元测试 -...
NODEJS + EXPRESS + DOCKER
该模版演示基于 Nodejs + Express 实现全自动检出代码 -> 单元测...
CODING GENERIC 制品上传
将一个文件上传到当前项目下的 Generic 制品库中。
JAVA-ANDROID 编译并签名 APK
该模版演示 Java-Android 检出代码、测试、构建、签名并将 Apk ...
SCP 部署
通过 SCP 进行部署。
ANGULAR-COS
该模版演示基于 Angular 实现全自动 检出代码 -> 单元测试 -> ...
CODING DOCKER 镜像推送
将一个构建完毕的 Docker 镜像推送到当前项目下的 Docker 制品库中。

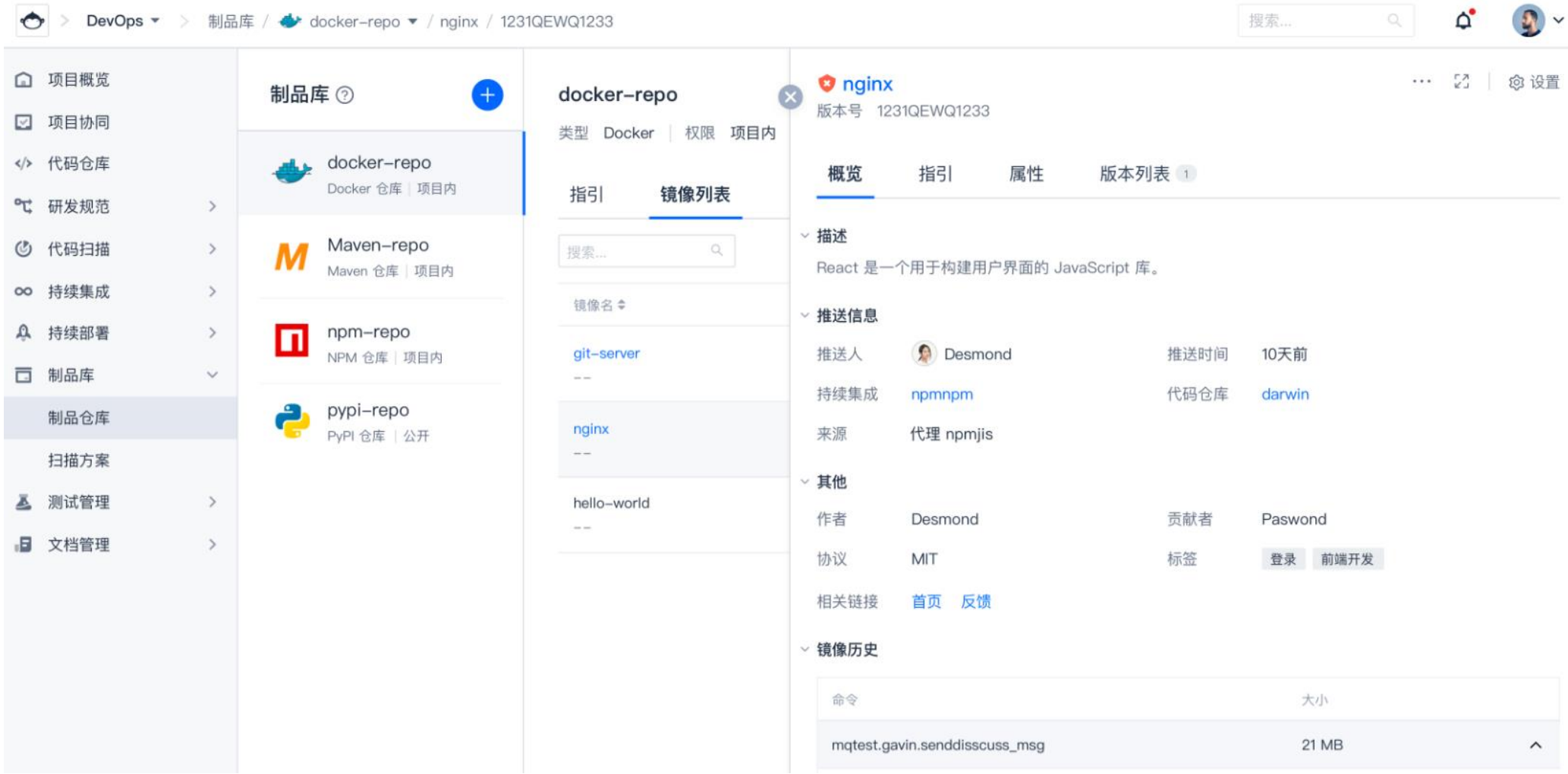
CODING 持续集成


Design-Center
持续集成 / 代码托管构建 / 修改配件
搜索...
项目概览
项目协同
代码仓库
研发规范
代码扫描
持续集成
构建计划
构建节点
持续部署
制品库
测试管理
文档管理
项目设置
代码托管构建
基础配置
流程配置
通知订阅
变量与缓存
触发策略
操作
分支
master
中的 build.groovy
图形化编辑器
文本编辑器
环境变量
放弃编辑
保存
2-1 检出
从代码仓库检出
+ 增加并行阶段
3-1 构建
打印消息
执行 Pipeline 脚本
收集构建物
打印消息
3-2 接口测试
打印消息
执行 Shell 脚本
收集构建物
收集通用报告
+ 增加并行阶段
收集通用报告
在持续集成过程中，用户可能会产生 HTML 格式的测试报告 / 代码覆盖率报告 / API 文档等，通过 CODING 通用报告收集插件，支持用户将各种类型 HTML 格式的报告收集起来，并提供对应独立的静态页面展示能力。
报告名称 *
my-report
路径 *
index.html
入口文件
请输入阶段名称
标签
请输入阶段名称
描述
请输入阶段名称

CODING 持续集成

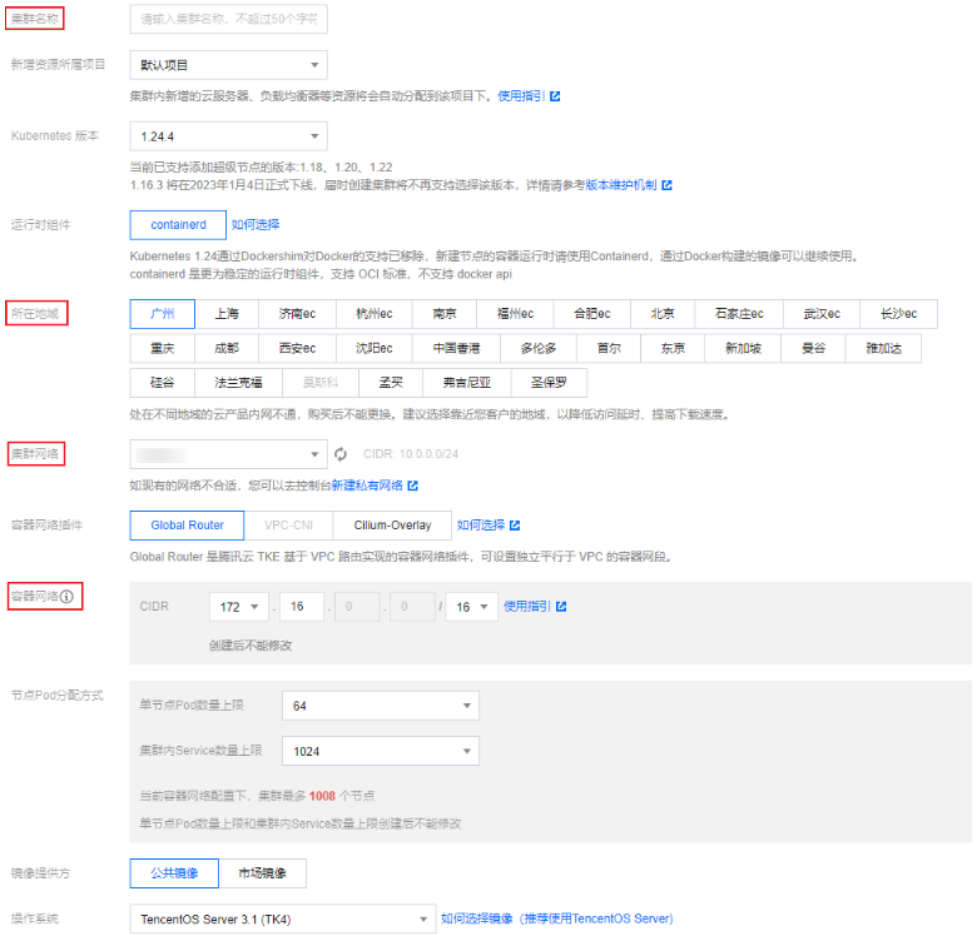
Design-Center
搜索...
项目概览
项目协同
代码仓库
研发规范
代码扫描
持续集成
构建计划
构建节点
持续部署
制品库
测试管理
文档管理
构建记录#8
构建成功
Accept Merge Request #12908 修复 CoreException 中的错误反馈代码
tank 提交于 1 小时前
构建过程
改动记录
测试报告
构建产物
配置快照
开始
检出代码 11 s
从代码仓库检出 11 s
前端构建 4 m 41 s
构建模块 ci-... 4 m 41 s
后端构建 4 m 22 s
构建模块 cci-... 2 m 52 s
构建模块 ci-m... 1 m 30 s
从代码仓库检出
2 分钟 3 秒
全屏
using credential e54cd5f6-1aff-45c9-a761-b70edde0b6bf
> git rev-parse --is-inside-work-tree # timeout=30
Fetching changes from the remote Git repository
> git config remote.origin.url
git@e.coding.net:codingcorp/coding-ci.git
timeout=30Fetching upstream changes from git@e.coding.
net:codingcorp/coding-ci.git --version # timeout=30
using GIT_SSH to set credentials
> git fetch --tags --progress git@e.coding.net:codingcorp/
coding-ci.git +refs/heads/*:refs/remotes/origin/* +refs/
merge/*:refs/remotes/origin/merge/*
> git rev-parse
688a53f88b05fb245d713d07a38fe9d8625140c7^{commit} #
timeout=30
Checking out Revision
688a53f88b05fb245d713d07a38fe9d8625140c7 (detached)
> git config core.sparsecheckout # timeout=30*:refs/
remotes/origin/* +refs/merge/*:refs/remotes/origin/merge/*
> git rev-parse
688a53f88b05fb245d713d07a38fe9d8625140c7^{commit} #
timeout=30
Checking out Revision
688a53f88b05fb245d713d07a38fe9d8625140c7 (detached)
> git config core.sparsecheckout # timeout=30*:refs/
remotes/origin/* +refs/merge/*:refs/remotes/origin/merge/*
> git rev-parse
688a53f88b05fb245d713d07a38fe9d8625140c7^{commit} #
timeout=30
Checking out Revision
688a53f88b05fb245d713d07a38fe9d8625140c7 (detached)
> git config core.sparsecheckout # timeout=30

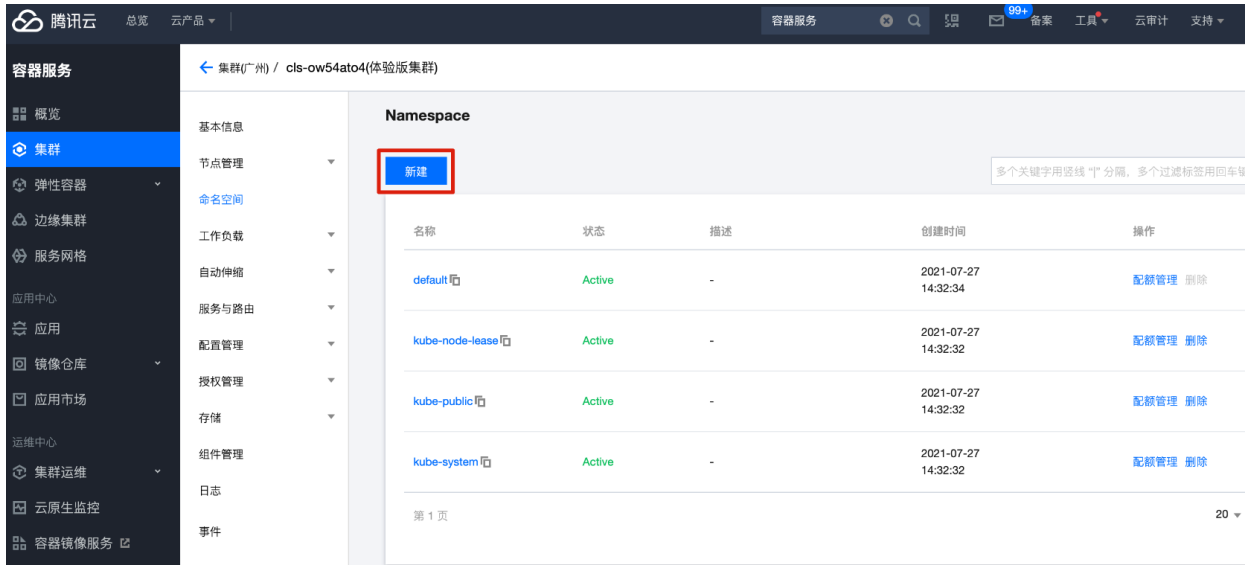
CODING 持续集成


DevOps
制品库 / docker-repo / nginx / 1231QEWQ1233
搜索...
项目概览
项目协同
代码仓库
研发规范
代码扫描
持续集成
持续部署
制品库
制品仓库
扫描方案
测试管理
文档管理
制品库
docker-repo
Docker 仓库 | 项目内
Maven-repo
Maven 仓库 | 项目内
npm-repo
NPM 仓库 | 项目内
pypi-repo
PyPI 仓库 | 公开
docker-repo
类型 Docker | 权限 项目内
指引
镜像列表
搜索...
镜像名
git-server
--
nginx
--
hello-world
--
nginx
版本号 1231QEWQ1233
设置
概览
指引
属性
版本列表 1
描述
React 是一个用于构建用户界面的 JavaScript 库。
推送信息
推送人 Desmond
推送时间 10天前
持续集成 npmnpm
代码仓库 darwin
来源 代理 npmjis
其他
作者 Desmond
贡献者 Paswond
协议 MIT
标签 登录 前端开发
相关链接 首页 反馈
镜像历史
命令
大小
mqtest.gavin.senddisscuss_msg
21 MB

CODING 持续部署

持续、可控、自动化地把软件制品发布到服务集群中，支持蓝绿发布、灰度发布（金丝雀发布）等多种发布策略。

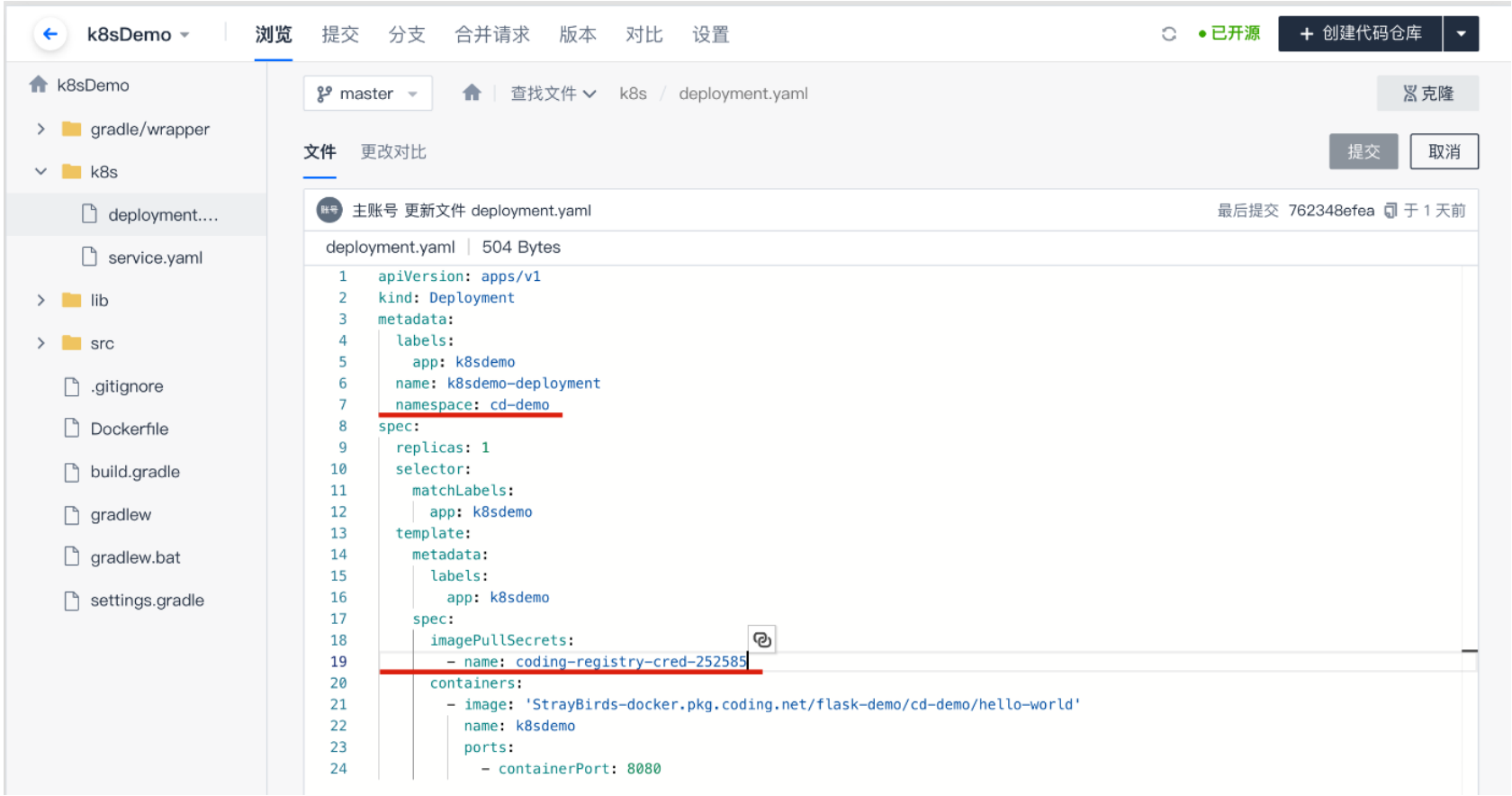




https://cloud.tencent.com/document/product/457/54231

CODING 持续部署

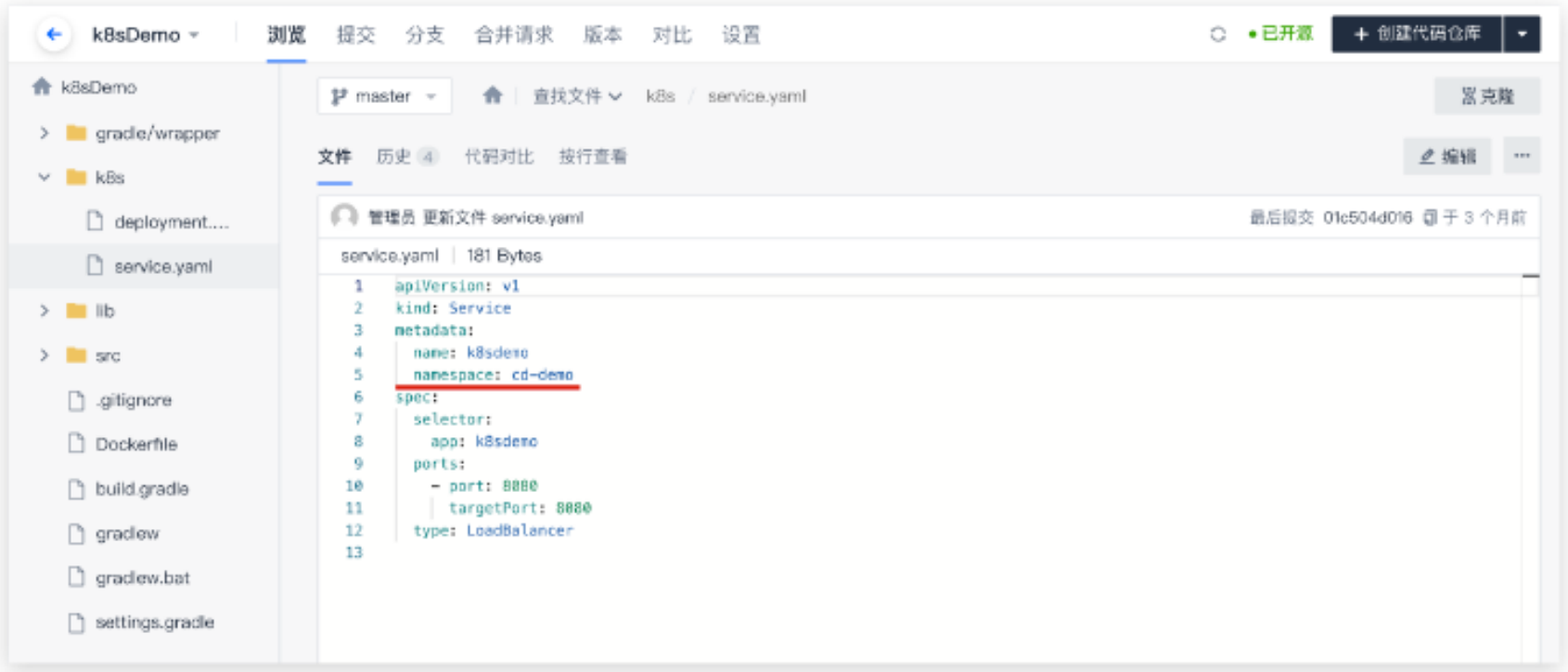
持续、可控、自动化地把软件制品发布到服务集群中，支持蓝绿发布、灰度发布（金丝雀发布）等多种发布策略。



https://coding.net/help/docs/cd/best-practice/circle.html

CODING 持续部署

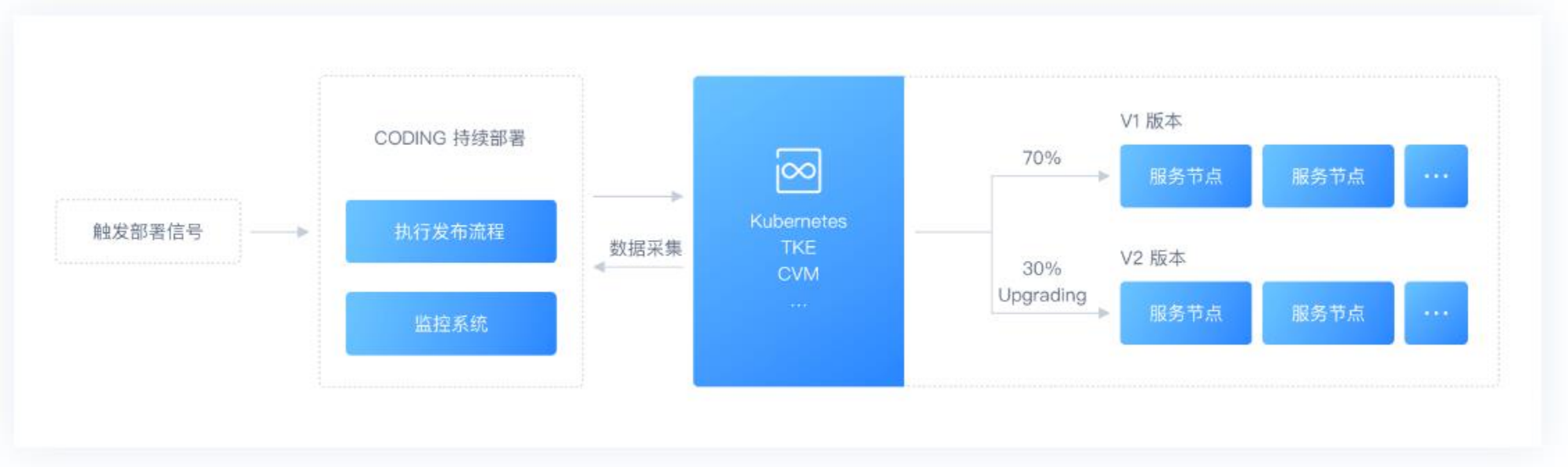
持续、可控、自动化地把软件制品发布到服务集群中，支持蓝绿发布、灰度发布（金丝雀发布）等多种发布策略。



https://coding.net/help/docs/cd/best-practice/circle.html

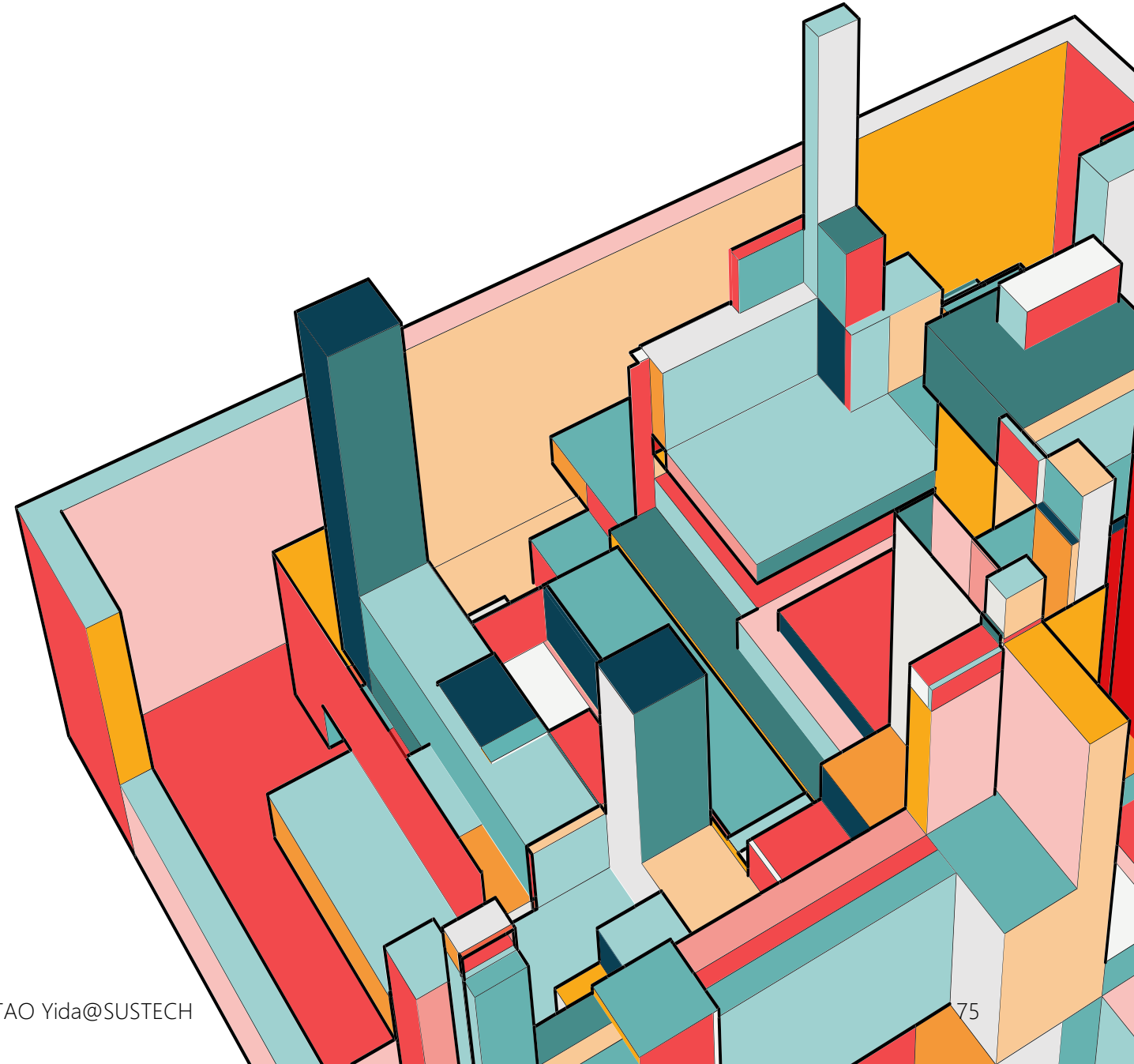
CODING 持续部署

持续、可控、自动化地把软件制品发布到服务集群中，支持蓝绿发布、灰度发布（金丝雀发布）等多种发布策略。



READINGS

- Chapter 23-24. Software Engineering at Google by Winters et al.
- 第7章 软件体系结构. 现代软件工程基础 by 彭鑫 et al.
- 第10章 软件集成与发布. 现代软件工程基础 by 彭鑫 et al.



NEXT

- AI + SE