探探云平台建设实践

×

蔡城杨

探探科技
高级技术专家

目录

第一部分

什么是“云平台”

什么是“云平台”


amazon
web services


Google Cloud


阿里云


Red Hat
OpenShift


KubeVela


Apache CloudStack™
Open Source Cloud Computing™


openstack.


kubernetes

什么是“云平台”?

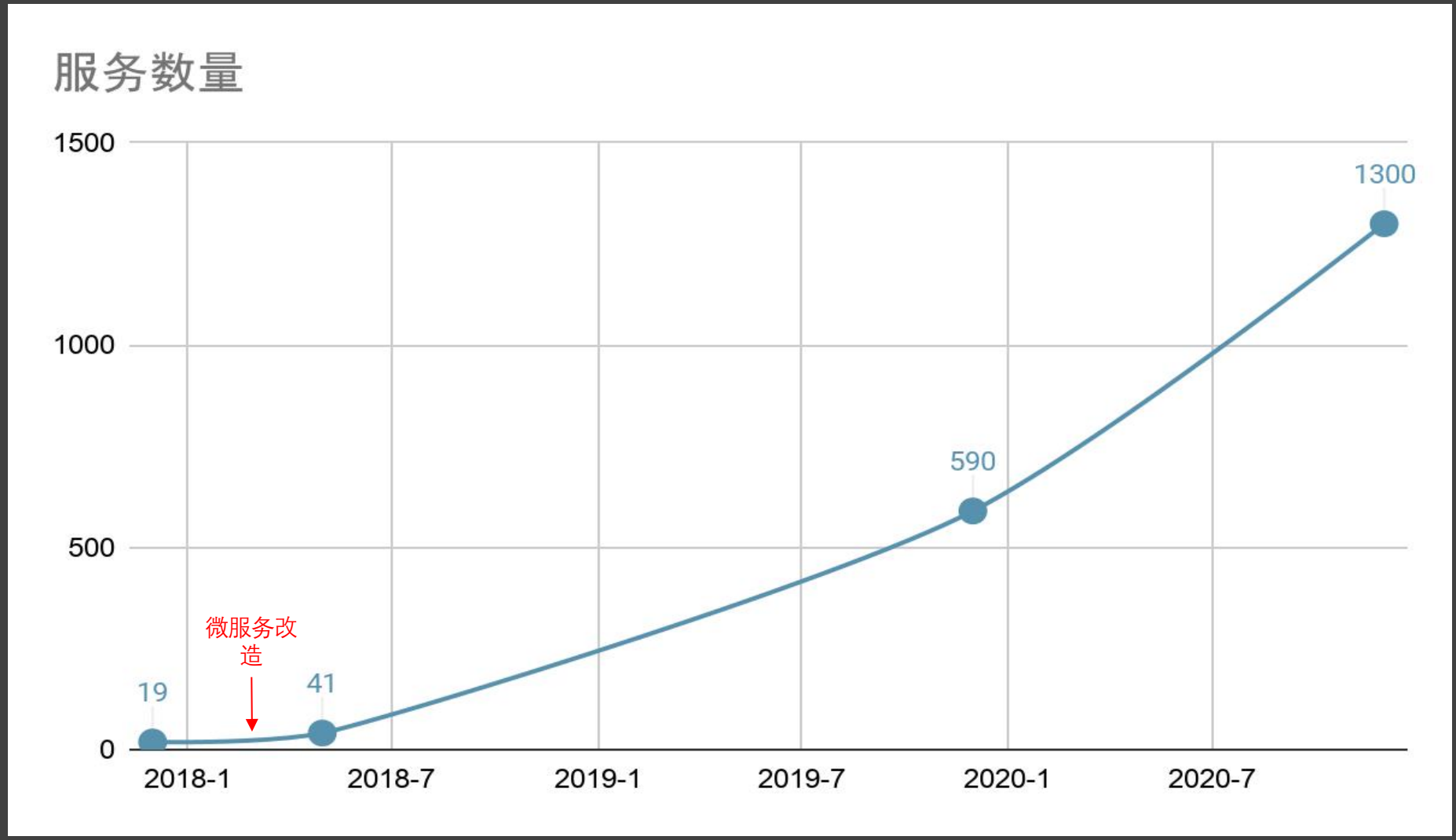
第二部分

为什么要建设“云平台”

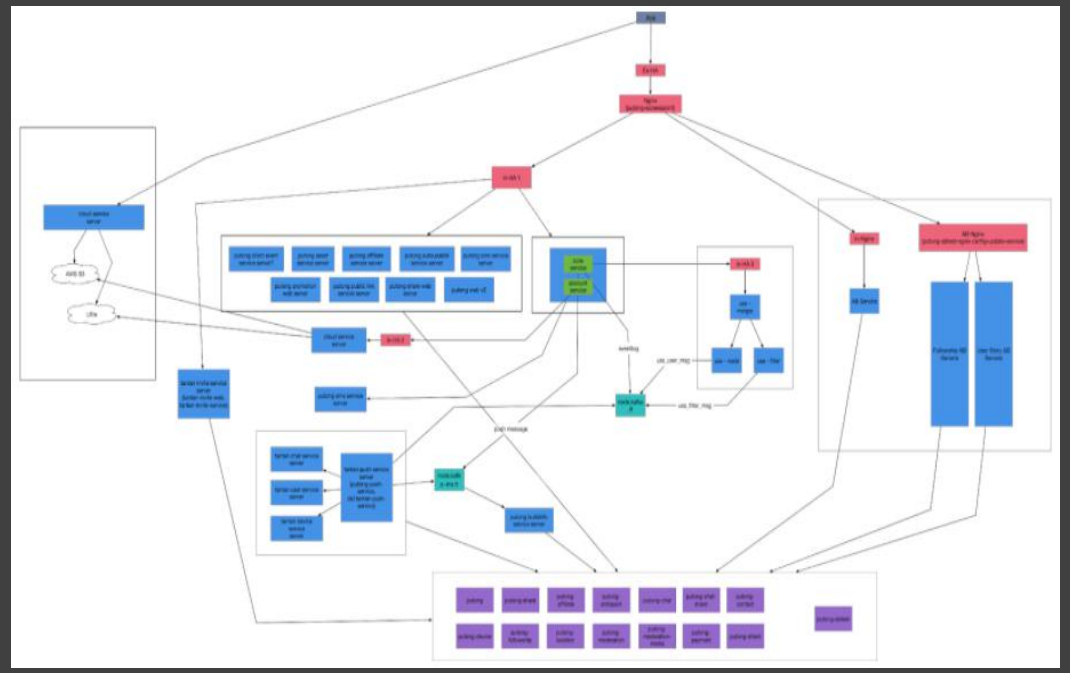
为什么要建设“云平台”

服务	服务	服务	服务
19	41	590+	1300+
2017-12	2018-05	2019-05	2020-12

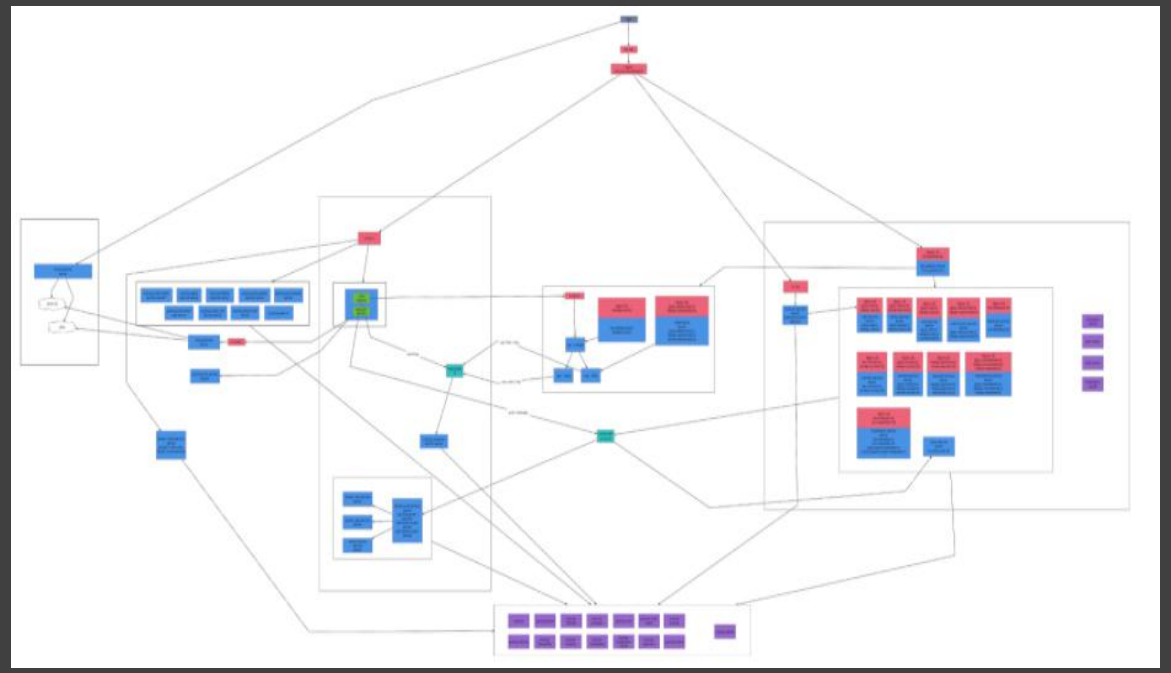
为什么要建设“云平台”


服务数量
1500
1000
500
0
1300
590
19
41
微服务改造
2018-1
2018-7
2019-1
2019-7
2020-1
2020-7

为什么要建设“云平台”

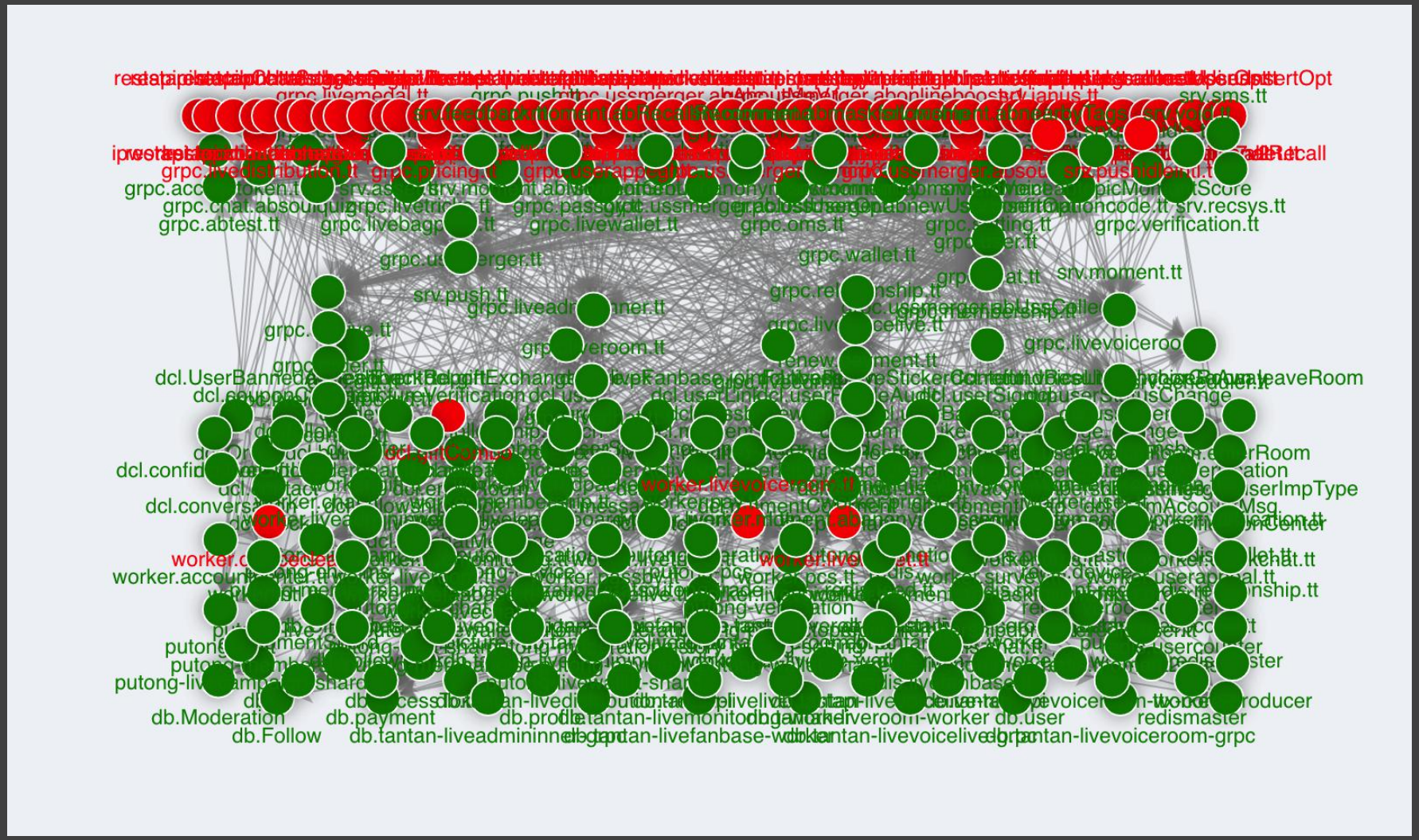


微服务改造 “开始前”
架构



微服务改造 “上线前”
架构

为什么要建设“云平台”


grpc.livemedal.tt
srv.sms.tt
grpc.livedistribution.tt
grpc.pricing.tt
grpc.chat.absoulu
grpc.livebagpr
grpc.livewallet.tt
grpc.oms.tt
grpc.verification.tt
srv.push.tt
grpc.wallet.tt
srv.moment.tt
grpc.livevoiceroo
dcl.UserBanned
dcl.getConfig
worker.livevoiceroom.tt
worker.live
worker.pcs.tt
db.Moderation
db.payment
db.Follow
db.user
redismaster

为什么要建设“云平台”

复杂度急剧上升

认知负荷

理解软件的设计、接口或者实现所需要的心智负担

- 整体逻辑难以理解，维护易出错，bug多，故障率高
- 原有程序在团队变化时候被抛弃，但是又无法下线

协同成本

团队在维护软件时需要在协同上额外付出的成本

- 系统演进速度变慢，效率变低
  - 交付一个功能，需要多个团队配合才能完成
  - 上线一个服务，需要多处操作

第三部分

什么时候建设“云平台”

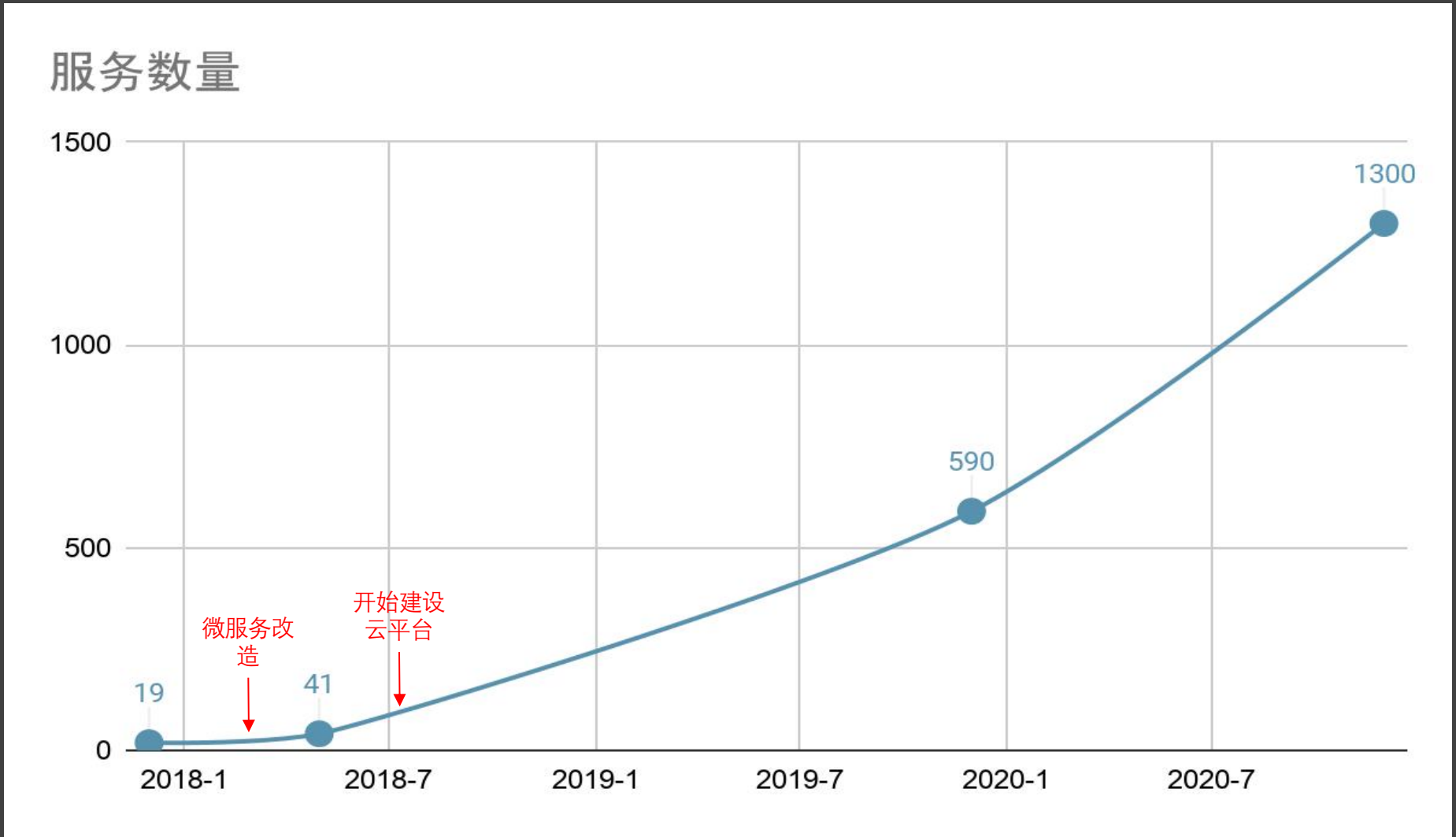
什么时候建设“云平台”

在复杂度带来的问题

显著影响 研发效率和质量

之前

什么时候建设“云平台”


服务数量
1500
1000
500
0
19
41
590
1300
微服务改造
开始建设云平台
2018-1
2018-7
2019-1
2019-7
2020-1
2020-7

第四部分

平台要支持哪些需求场景

平台需要支持哪些需求场景

- 1个 前提
  - 面向企业内部
- 2个（用户）视角
  - 研发视角：研发工程师、QA工程师、研发管理者
  - 运维视角
- 3个（考察）维度
  - 效率
  - 稳定性
  - 成本

平台需要支持哪些需求场景

	研发视角	运维视角
效率	● 服务交付、变更 ● 服务管理：扩容、缩容、降级、重启...	● 资源交付、管理 ○ CMDB：以“业务模块/服务”为核心 ○ 基础组件（Kafka、Redis）的管理、交付 ○ 网络（DNS、LB）的管理
稳定性	● 监控报警 ○ 层次：系统、网络、业务 ○ 数据来源：日志、Metrics、Tracing ○ 关注点：所负责服务的访问量、错误、延迟、水位	● 监控报警 ○ 层次：硬件，系统，网络，业务 ○ 数据来源：Metrics ○ 关注点：核心业务大盘指标，关键网络、基础组件 ● 运营 ○ 巡检 ○ 故障管理
成本	n/a	● Capex，Opex ● 预算、采购、分配、回收、报废

平台需要支持哪些需求场景

平台需要支持哪些需求场景

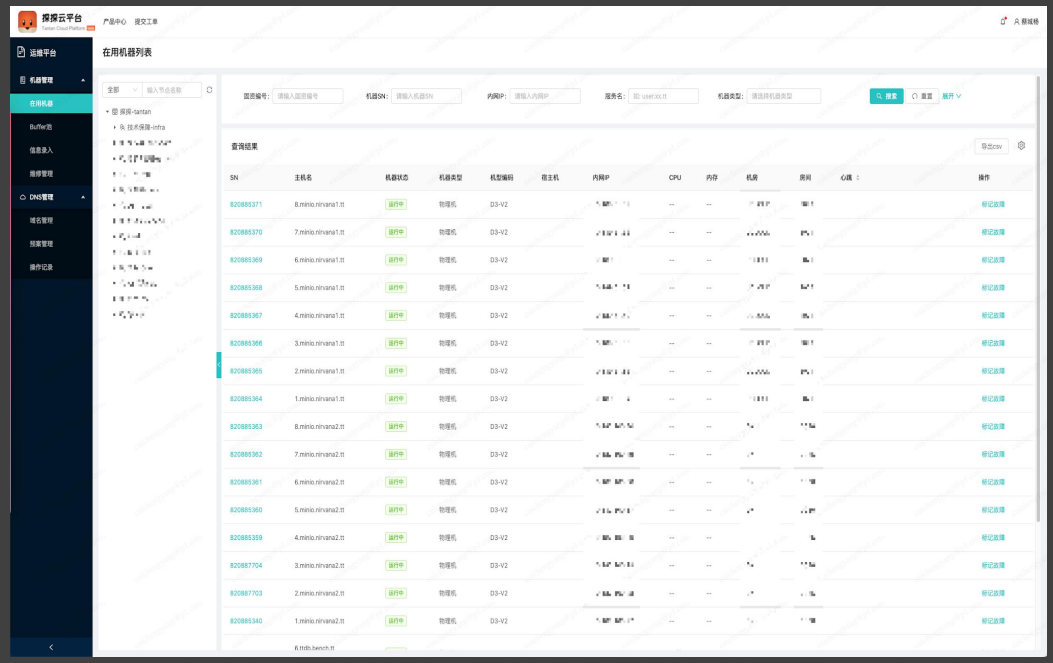
- 资源交付、管理
  - 资源包含：
    - 计算：主机、容器
    - 存储：关系DB（PG、MySQL）、NoSQL DB（Redis、ES）、对象存储（Minio）
    - 网络：域名、LB
    - 其它：消息队列（Kafka）
- 稳定性
  - 监控：
    - 同研发：业务量、服务Metrics、服务日志
    - 运维独有：网络（外部、内部）、硬件、基础组件
  - 运营
    - 巡检
    - 故障管理：定损、case study、改进
    - 非标准化的运维变更
  - …
- 成本

第五部分

如何建设“云平台”

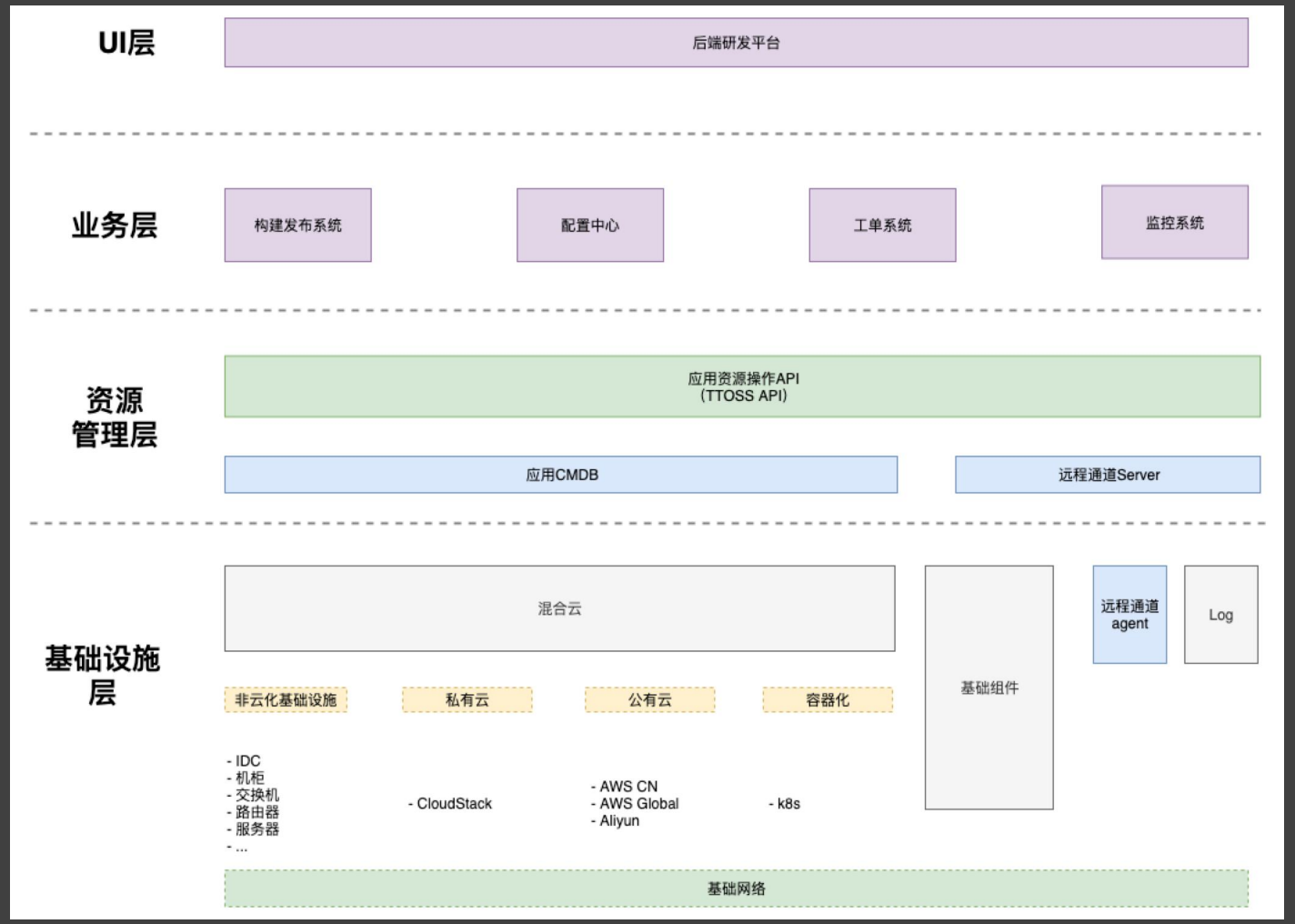
如何建设云平台

（后端）研发平台

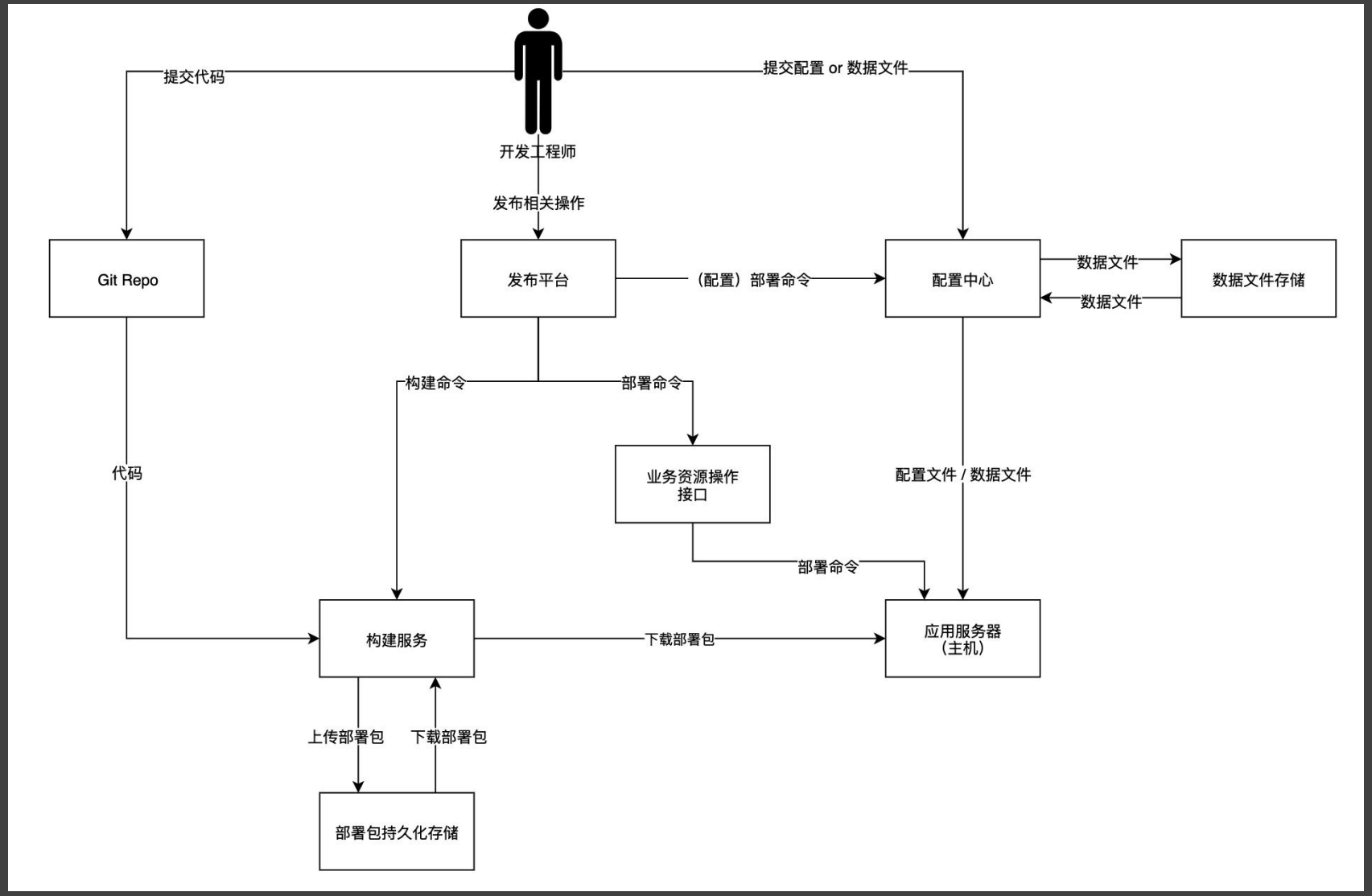


运维平台

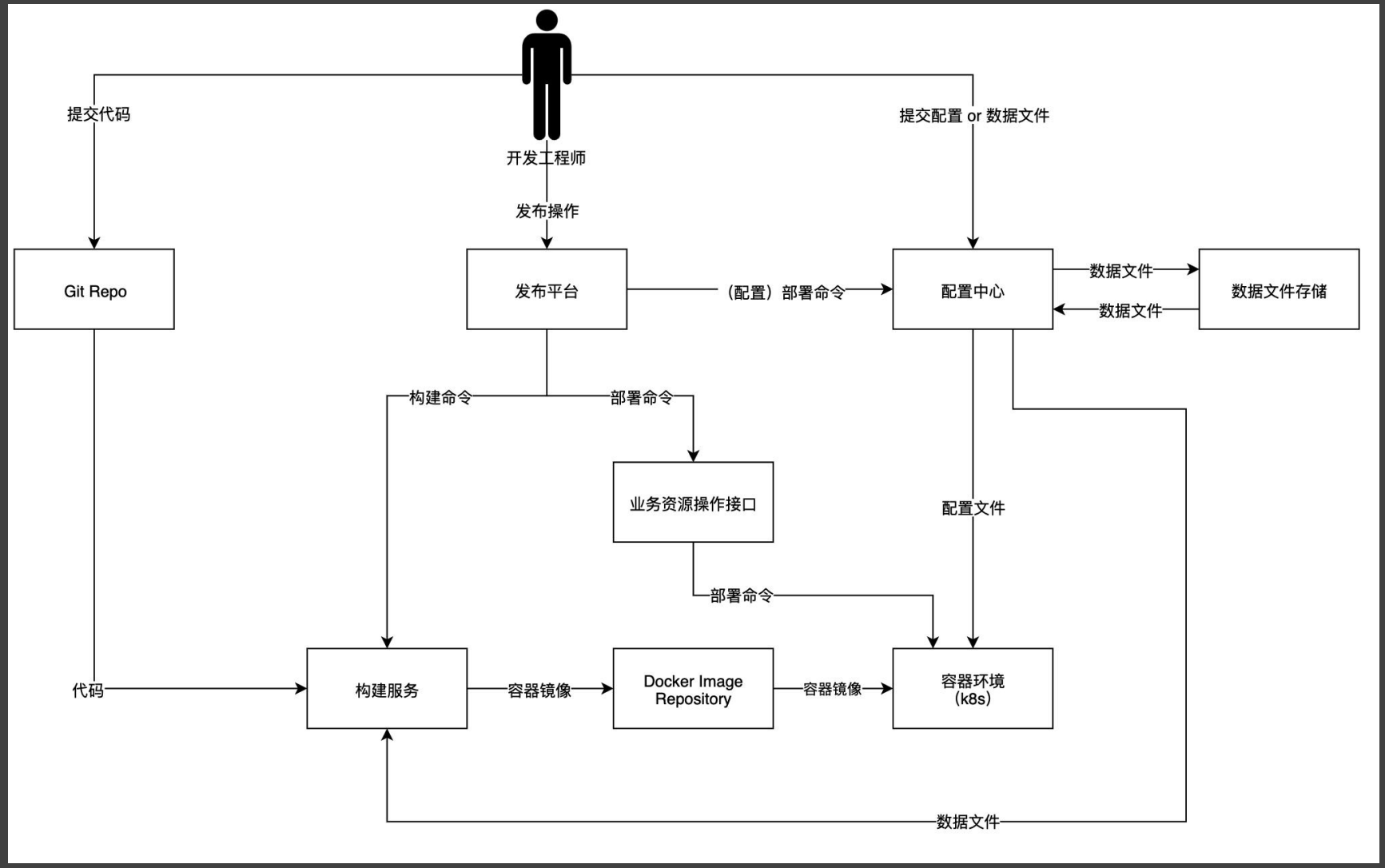
如何建设云平台


UI层
后端研发平台
业务层
构建发布系统
配置中心
工单系统
监控系统
资源管理层
应用资源操作API
(TTOSS API)
应用CMDB
远程通道Server
基础设施层
混合云
基础组件
远程通道agent
Log
非云化基础设施
私有云
公有云
容器化
- IDC
- 机柜
- 交换机
- 路由器
- 服务器
- ...
- CloudStack
- AWS CN
- AWS Global
- Aliyun
- k8s
基础网络

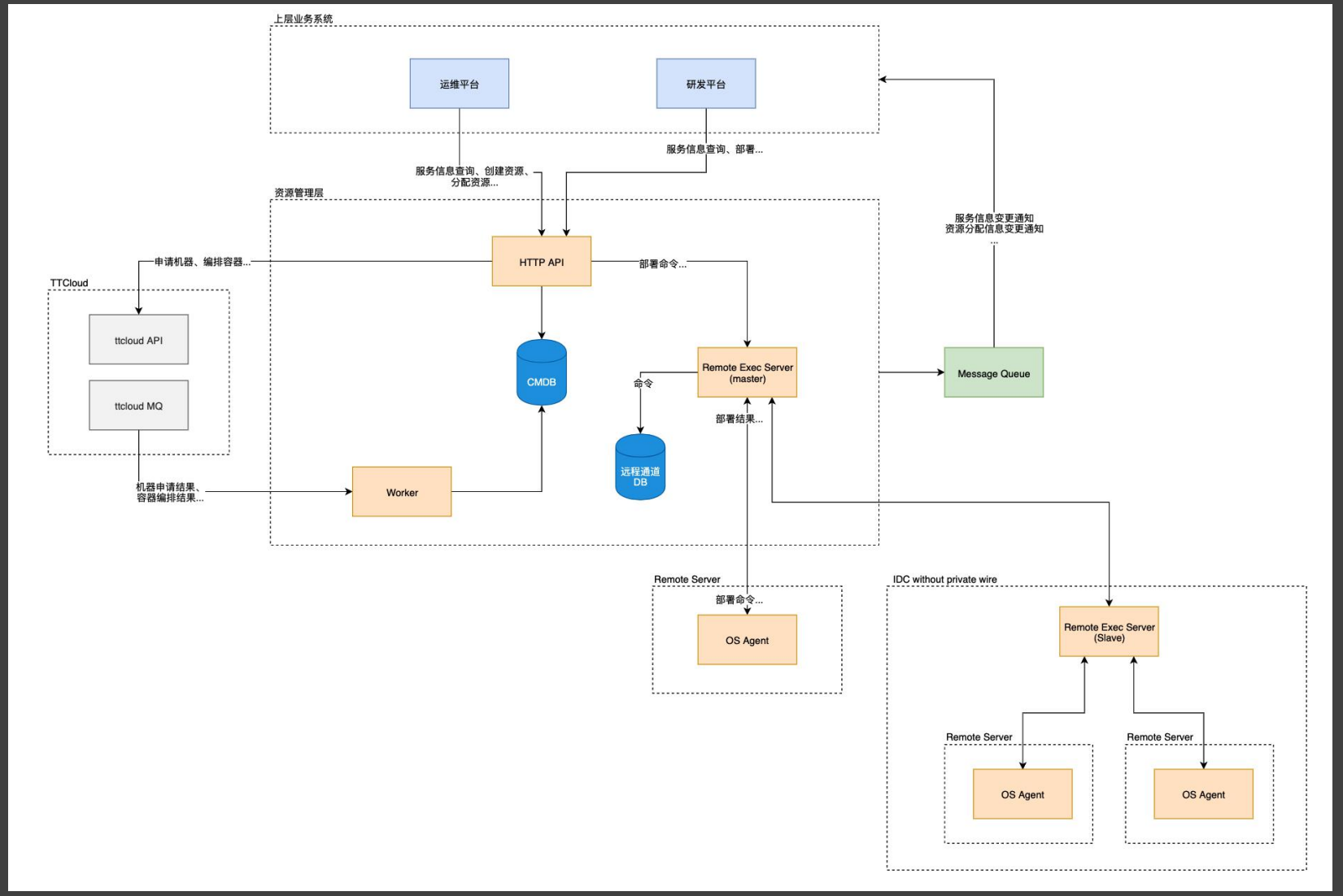
如何建设云平台


提交代码
提交配置 or 数据文件
开发工程师
发布相关操作
Git Repo
发布平台
（配置）部署命令
配置中心
数据文件
数据文件
数据文件存储
构建命令
部署命令
代码
业务资源操作
接口
配置文件 / 数据文件
部署命令
构建服务
下载部署包
应用服务器
（主机）
上传部署包
下载部署包
部署包持久化存储

如何建设云平台


提交代码
开发工程师
提交配置 or 数据文件
发布操作
Git Repo
发布平台
（配置）部署命令
配置中心
数据文件
数据文件
数据文件存储
构建命令
部署命令
业务资源操作接口
配置文件
部署命令
代码
构建服务
容器镜像
Docker Image Repository
容器镜像
容器环境 (k8s)
数据文件

如何建设云平台


上层业务系统
运维平台
研发平台
服务信息查询、部署...
服务信息查询、创建资源、
分配资源...
资源管理层
服务信息变更通知
资源分配信息变更通知
...
HTTP API
申请机器、编排容器...
部署命令...
TTCloud
ttcloud API
ttcloud MQ
CMDB
Remote Exec Server
(master)
Message Queue
命令
部署结果...
远程通道
DB
机器申请结果、
容器编排结果...
Worker
Remote Server
部署命令...
OS Agent
IDC without private wire
Remote Exec Server
(Slave)
Remote Server
OS Agent
Remote Server
OS Agent

如何建设云平台

- 产品设计
  - 避免产品功能过于分散
  - 集成上下游产品，减少用户的使用成本
  - 厘清平台、用户（比如后端工程师）之间的业务边界，减少协作、依赖，一定要考虑到功能上线后的推广、使用成本
- 架构
  - 规范先行
  - 控制复杂度：对于遗留系统、遗留逻辑，适当时刻一次性彻底解决
- 项目&团队管理
  - 基础建设需要同时推进
  - 注重工程师、团队全栈能力培养、建设

如何建设云平台

- 服务类型
  - HTTP API
  - Worker
  - Cron Job
  - Agent
- 开发语言：Go
- Package
  - github.com/gin-gonic/gin
  - github.com/go-resty/resty/v2
  - github.com/jinzhu/gorm
  - github.com/pkg/errors
  - github.com/sirupsen/logrus
  - etc

如何建设云平台

```
tantan-greeter/
  app/
    config/{parser.go}          # 配置解析
    providers/                  # 所有依赖相关的
      cache/cache.go
      db/pgv6/pgv6.go
      eventlog/eventlog.go
      dcl/dcl.go
    rest/rest.go                # http service 相关
    rpc/rpc.go                  # grpc service 相关
    greeter/                    # 代表一个业务模块
      api/service.go            # 代表 api 类型的 App
      worker/service.go         # 代表 worker 类型的 App
  cmd/
    srv.greeter.tt/main.go      # 目录名与 App 名称相同
    worker.greeter.tt/main.go
  conf/
    dev/
      srv.greeter.tt/
        {common.json,rpc.json,http.json,service.json,cache.json,dcl.json,db.json}
      worker.greeter.tt/
        {common.json,service.json,dcl.json}
  pm/
    srv.greeter.tt.yaml         # 服务描述文件，详细说明见下面的章节
    worker.greeter.tt.yaml      # 服务描述文件，详细说明见下面的章节
  Makefile
  .gitlab-ci.yml                # ci相关功能配置
```

如何建设云平台

- make build
- make 命令的 targets：all，build，pack，clean，test
- make build、make pack 的参数

参数名	意义	例子	备注
version	可执行程序的版本	v1.2.0.45095 make build version=v1.2.0.45095 v1.11.2-ab183-1.0.3.45247 make build version=v1.11.2-ab183-1.0.3.45247	version的规则：${git_tag}.{ci_pipeline_id} git repo tag规则： • master branch tag：v1.2.0 • ab branch tag：${master_branch_version}-{ab_name}-{ab_version}
env	目标环境	prod *make build env=prod version=v1.2.0.45095*	取值范围： • dev：开发环境 • test：测试环境 • prod：生产环境

如何建设云平台

```
LDFLAGSPREFIX := gitlab.tantan.com/$(REPO_PATH)
...
LDFLAGS += -X $(LDFLAGSPREFIX)/version.version=$(VERSION)
LDFLAGS += -X $(LDFLAGSPREFIX)/version.gitTag=$(GIT_TAG)
LDFLAGS += -X $(LDFLAGSPREFIX)/version.branch=$(BRANCH)
LDFLAGS += -X $(LDFLAGSPREFIX)/version.commitId=$(COMMIT_ID)
...
LDFLAGS += -X $(LDFLAGSPREFIX)/version.buildTime=$(BUILD_TIME)
LDFLAGS += -X '$(LDFLAGSPREFIX)/version.goVersion=\"$(GO_VERSION)\"'
...


build:
   …

   export GO111MODULE=on && go build -o build/bin/$(SERVICE_NAME) \
      -mod vendor -ldflags "$(LDFLAGS)" \
      gitlab.tantan.com/$(REPO_PATH)/cmd/$(SERVICE_NAME)/...
```

如何建设云平台

```
version: v1

service_name: ${service_name}          # 服务名
service_type: daemon                   # 服务部署类型 daemon cron static,默认daemon
user: tantan                           # 可选配置，默认是tantan

prestop:                               # 可选配置，停止服务前置操作
  rest:                                # REST API方式
    method: DELETE
    path: /debug/health                # 停止服务接入API
    wait: 3                            # 停止服务接入等待时间(秒)
stop:                                  # 可选配置，停止服务时的操作
    timeout: 10                        # 停止服务超时时间(秒)，默认10秒

start:                                 # 启动配置
  timeout: 60                          # 启动服务超时时间(秒)，默认60秒
  machine:                             # 主机：物理机or虚拟机
    systemd:
      env:                             # Systemd环境变量，用于配置Environment
        - OMP_NUM_THREADS: 4           # 指定线程数
      execStart: "${service_root}/bin/${service_name} --config=${service_root}/conf/main.yaml"
  container:                           # 容器
    …
```

如何建设云平台

```
type Time time.Time

//实现 Unmarshaler interface
func (t *Time) UnmarshalJSON(data []byte) error {
        millis, err := strconv.ParseInt(string(data),
10, 64)
        if err != nil {
                return err
        }
        *t = Time(ConvertInt642Time(millis))
        return nil
}

//实现 Marshaler interface
func (t Time) MarshalJSON() ([]byte, error) {
        millis := ConvertTime2Int64(time.Time(t))
        return json.Marshal(millis)
}
```

```
//https://pkg.go.dev/database/sql/sql#Scanner
func (t Time) Value() (driver.Value, error) {
        return time.Time(t), nil
}

//https://pkg.go.dev/database/sql/driver#Valuer
func (t *Time) Scan(value interface{}) error {
        if v, ok := value.(time.Time); ok {
                *t = Time(v)
                return nil
        } else {
                return errors.Errorf("type convert
error : %+v", value)
        }
}
```

如何建设云平台

```
[Unit]
Description={__service_name}
After=network-online.target

StartLimitInterval=60       # 尝试重启的时间间隔，StartLimitInterval 需要大于 StartLimitBurst *
RestartSec
StartLimitBurst=3           # 尝试重启次数

[Service]
Type=simple
User={__user_run_app}

LimitNOFILE=10240000

WorkingDirectory={__app_dir}

ExecStart={__start_command}

Restart=on-failure
RestartSec=1                # 连续两次重启的时间间隔


[Install]
WantedBy=multi-user.target
```

Q&A