



架构设计分论坛

12-Factors 原则与云端应用

张孝峰

AWS 资深解决方案架构师



12要素应用宣言

十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

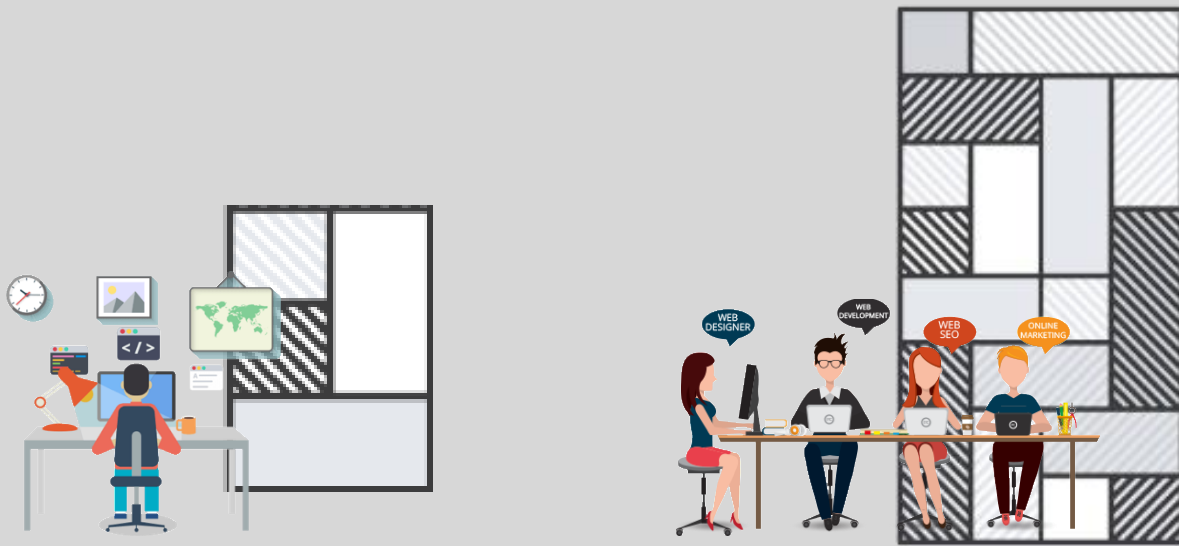
XI. 日志

XII. 管理进程

12要素应用宣言的目标







目标1 —— 标准化

使用**标准化**流程进行自动配置，从而使新的开发者花费最少的学习成本加入这个项目



目标2 —— 最大的可移植性

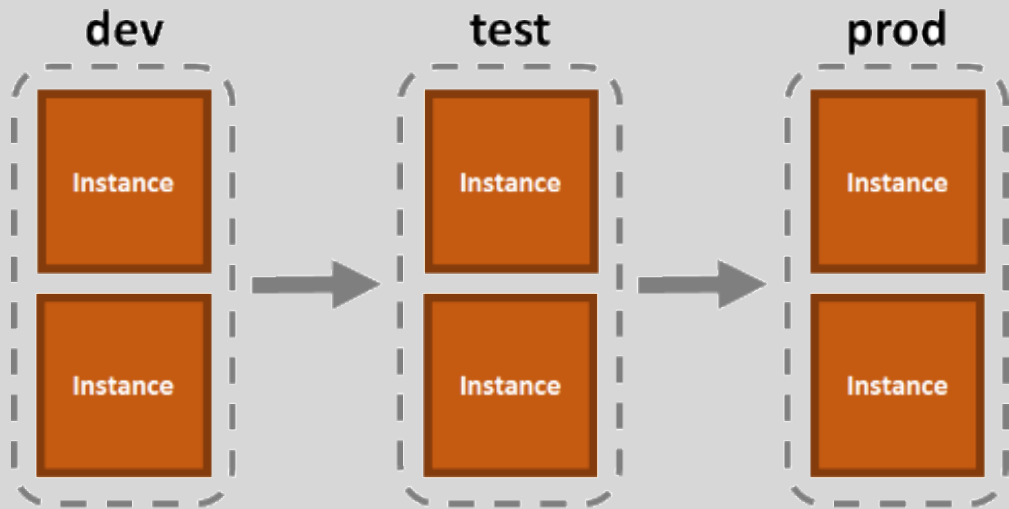
和操作系统之间尽可能的划清界限，在各个系统中提供最大的可移植性

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



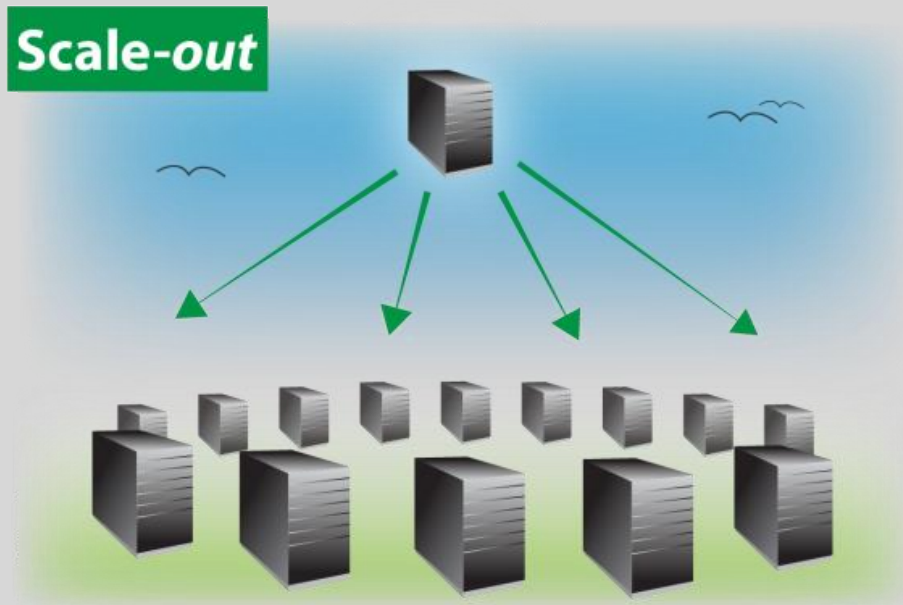
目标3 —— 统一生产、开发环境

将开发环境和生产环境的差异降至最低，并使用**持续交付**实施敏捷开发



目标4 —— 弹性的扩展

可以在工具、架构和开发流程不发生明显变化的前提下实现扩展



目标 —— 尽可能的利用现代化的云平台

适合部署在现代的云计算平台，从而在服务器和系统管理方面节省资源

1. 使用时付费，无需事先固定投资
2. 不再需要猜测容量
3. 增加创新：更快尝试、低成本、低风险
4. 摆脱无差异化的工作
5. IT整体成本降低
6. 数分钟就可全球化部署



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

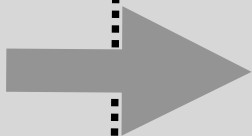
XI. 日志

XII. 管理进程

代码



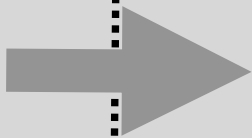
代码



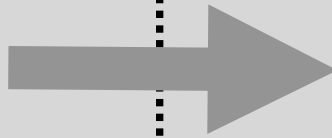
版本控制



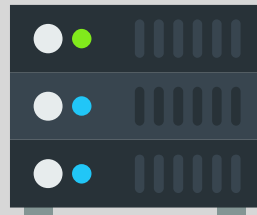
代码



版本控制



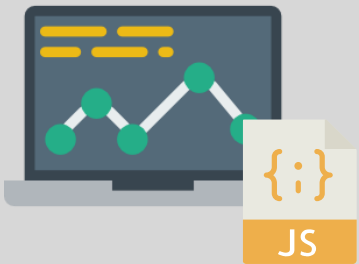
部署的版本



开发者1



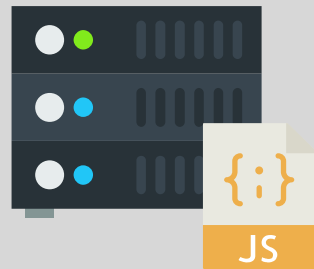
开发者2



集成测试环境

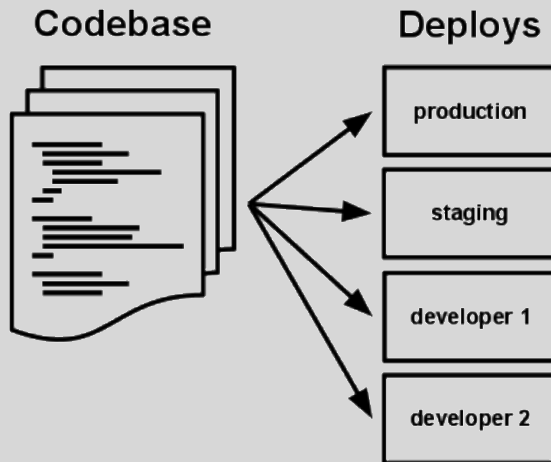


生产环境



基准代码的好处

- 容易审计代码
- 很容易rollback，只要部署版本 - 1
- 非常清晰上线步骤
- 并行开发
- 最小化重构的风险



在AWS上实现基准代码

AWS CodePipeline



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

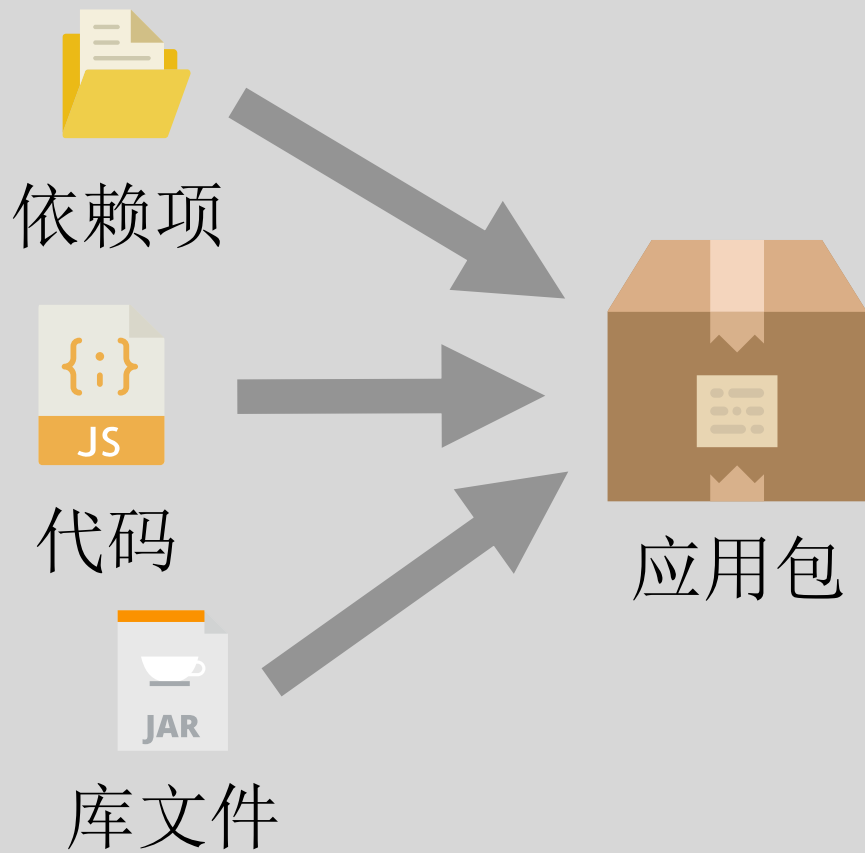
VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程



依赖声明: Node.js

package.json

```
{
  "dependencies": {
    "async": "2.1.4",
    "express": "4.16.2",
    "express-bearer-token": "2.1.0",
    "body-parser": "1.18.2",
    "jwt-simple": "0.5.1",
    "lodash": "4.17.4",
    "morgan": "1.7.0",
    "request": "2.81.0"
  }
}
```

npm install

依赖声明: Python

requirements.txt

```
django==1.6
bpython==0.12
django-braces==0.2.1
django-model-utils==1.1.0
logutils==0.3.3
South==0.7.6
requests==1.2.0
stripe==1.9.1
dj-database-url==0.2.1
django-oauth2-provider==0.2.4
django-rest-framework==2.3.1
```

pip install

依赖声明: Ruby

Gemfile

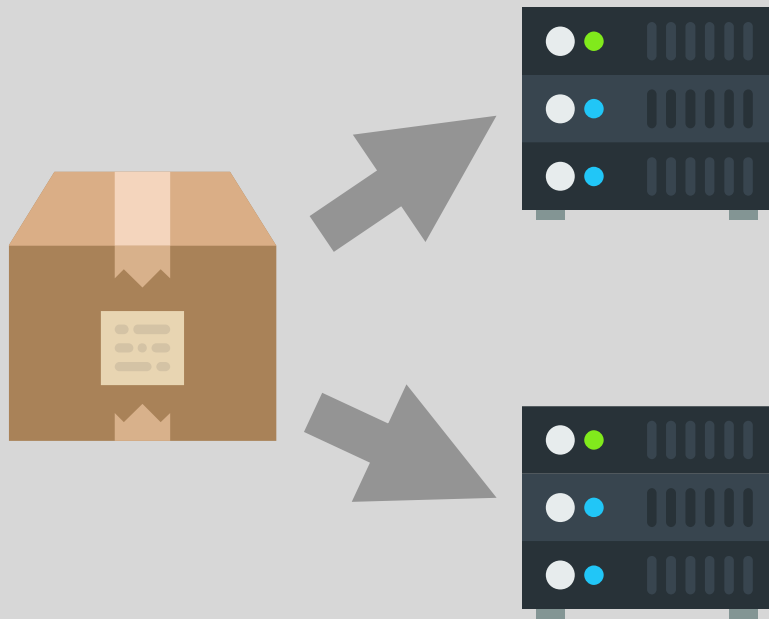
```
source 'https://rubygems.org'  
  
gem 'nokogiri'  
gem 'rails', '3.0.0.beta3'  
gem 'rack', '>=1.0'  
gem 'thin', '~>1.1'
```

bundle install

依赖隔离

不要隐式依赖系统级的类库.

应用程序部署应随附其所有
依赖项.





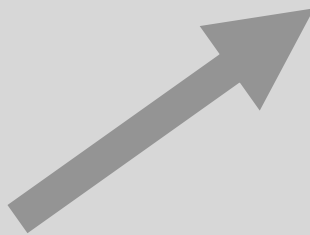
依赖项



代码



库文件



依赖的声明与隔离: Docker

Dockerfile

```
FROM mhart/alpine-node:8

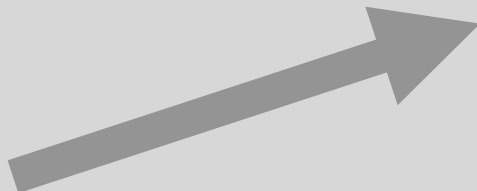
RUN apk add --no-cache make gcc g++ python

WORKDIR /srv
ADD . .
RUN npm install

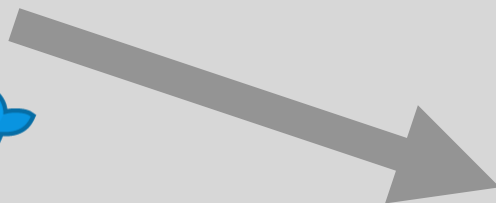
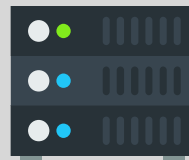
EXPOSE 3000
CMD ["node", "index.js"]
```

docker build

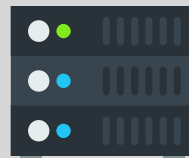
docker run



开发环境



生产环境



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

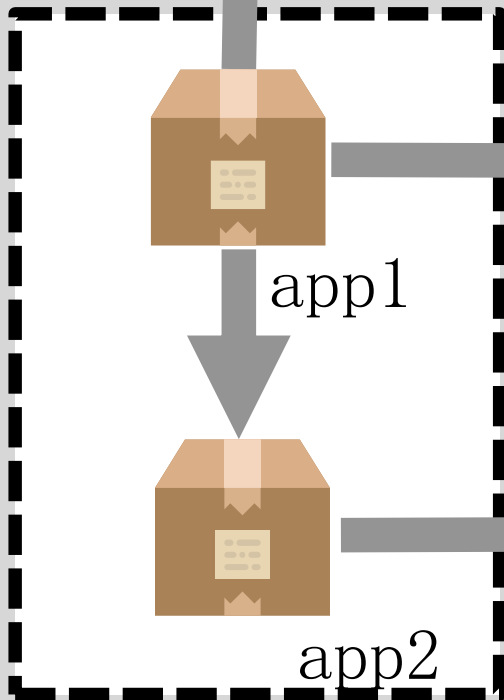
XII. 管理进程

Amazon S3



像远程第三方一样对待本地服务

主机



app1

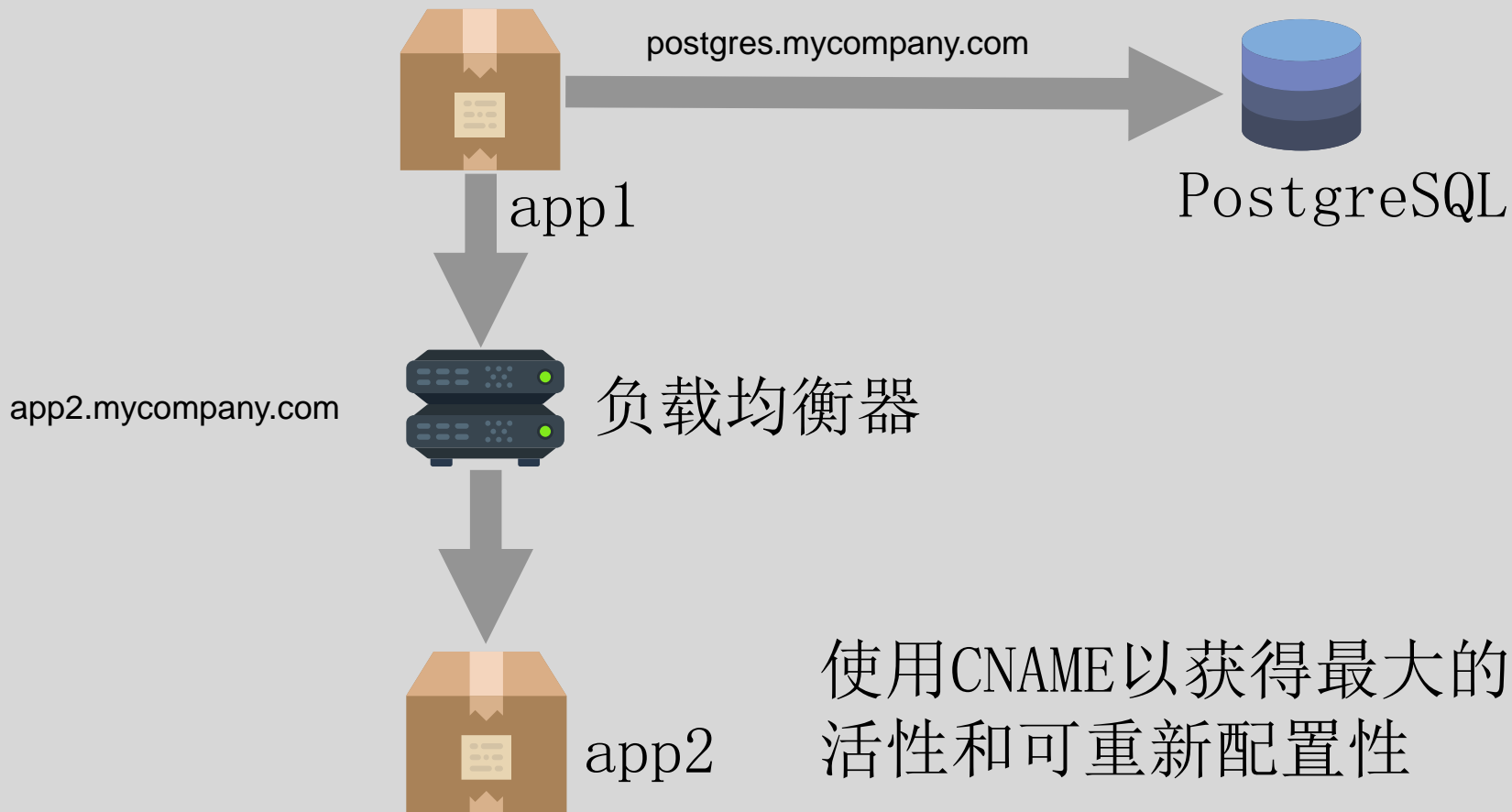
app2



PostgreSQL



第三方服务



使用CNAME以获得最大的灵活性和可重新配置性

云即服务



计算



存储



数据库



迁移



网络和内容分发



开发人员工具



管理工具



媒体服务



安全性、身份与合规性



分析



机器学习



移动服务



AR 和 VR



应用程序集成



客户参与



企业生产力



桌面和应用程序流式处理



物联网



游戏开发



查看所有产品

十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

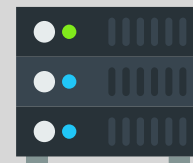
X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程



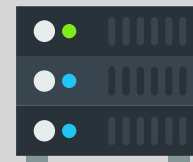
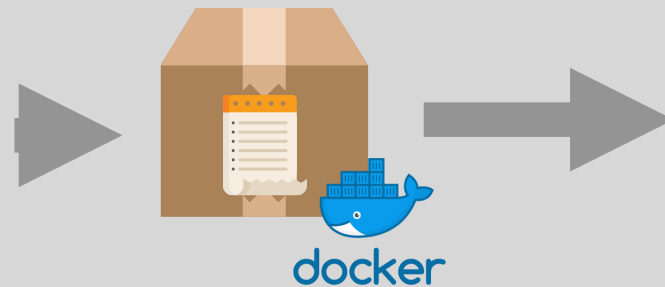
开发环境
配置文件



开发环境



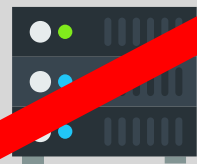
生产环境
配置文件



生产环境



开发环境
配置文件

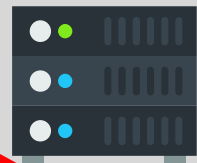


开发环境

不推荐

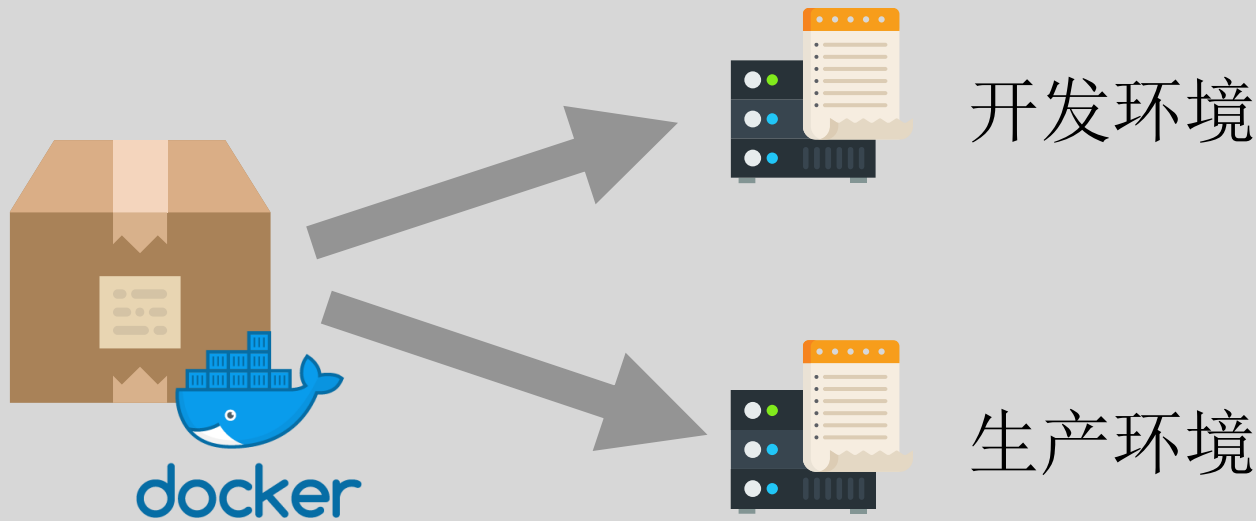


生产环境
配置文件

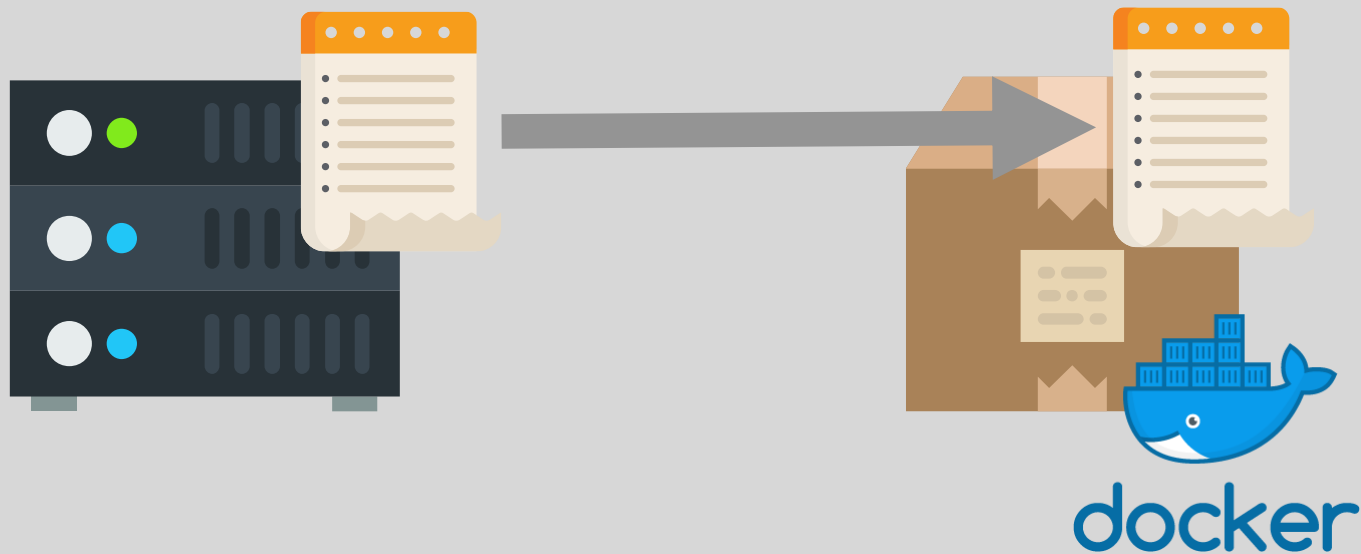


生产环境

部署在两个环境中的同一容器。配置是主机上环境的一部分。



在运行时，容器从环境中获取配置。



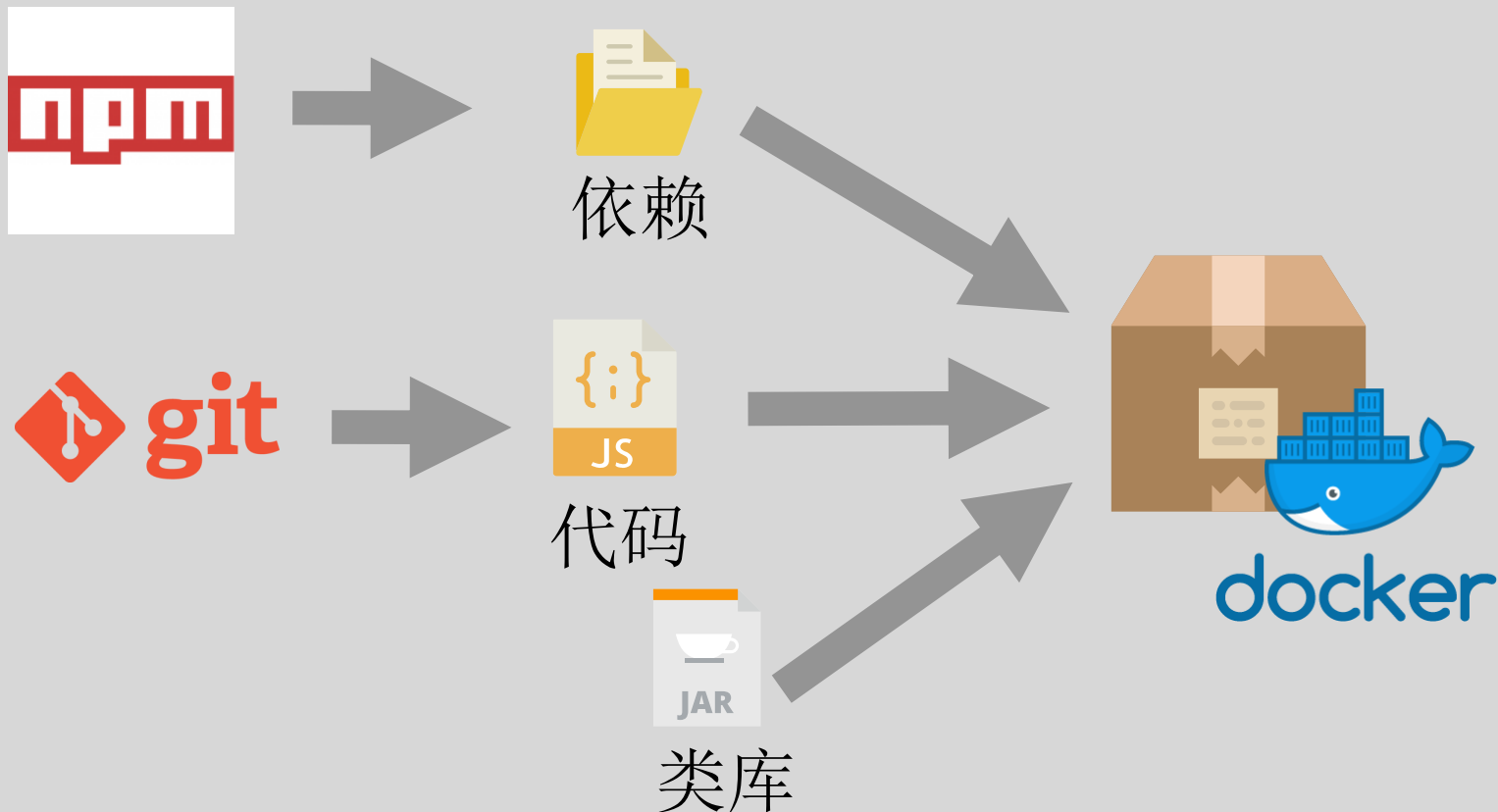
应用代码从环境中读取

```
module.exports = {  
  DATABASE: process.env.DATABASE,  
  SECRET: process.env.SECRET  
};
```

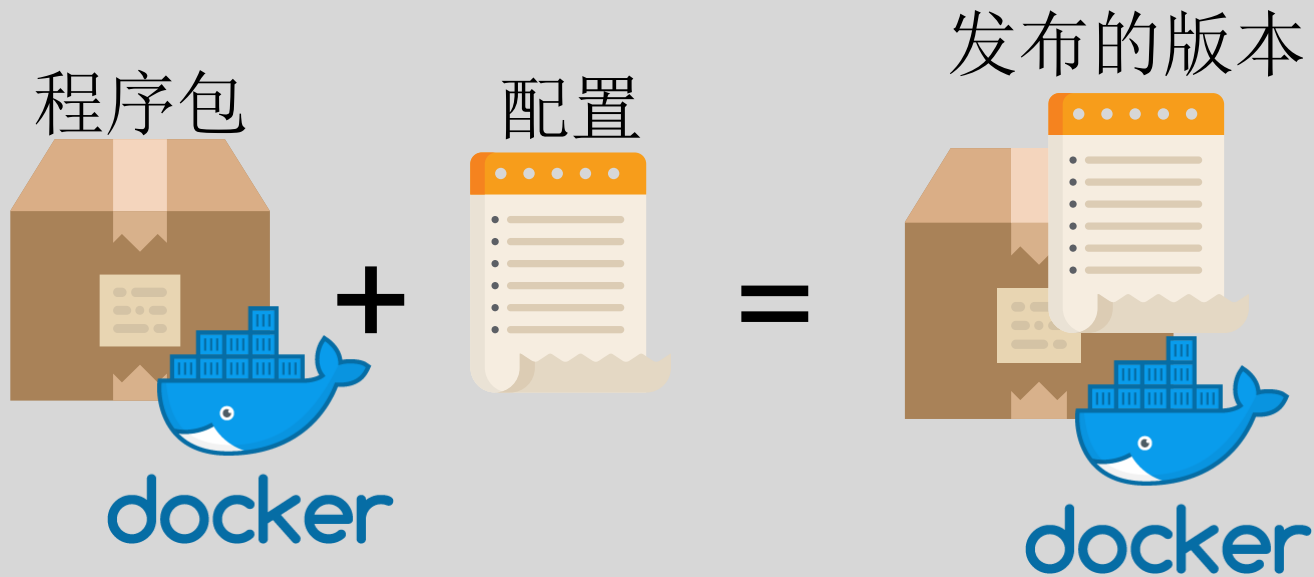
在运行容器时定制环境

```
docker run -e "DATABASE=mongodb://localhost:27017" -e "SECRET=default" myapp  
  
docker run -e "DATABASE=mongodb://db1.mycompany.com,db2.mycompany.com/  
production?replicaSet=production" -e "SECRET=hunter2" myapp
```

构建



发布



Amazon Elastic Container Service

```
# The task definition that describes how to run the docker container
```

TaskDefinition:

```
Type: AWS::ECS::TaskDefinition
```

Properties:

```
Family: !Ref 'ServiceName'
```

```
TaskRoleArn: !Ref TaskRole
```

ContainerDefinitions:

```
- Name: !Ref 'ServiceName'
```

```
Cpu: 512
```

```
Memory: 256
```

```
Image: !Ref 'ImageUrl'
```

Ulimits:

```
- Name: nofile
```

```
HardLimit: 65535
```

```
SoftLimit: 65535
```

PortMappings:

```
- ContainerPort: 3000
```

```
HostPort: 0
```

Environment:

```
- Name: 'UV_THREAD_POOL'
```

```
Value: '15'
```

```
- Name: 'NODE_ENV'
```

```
Value: 'production'
```



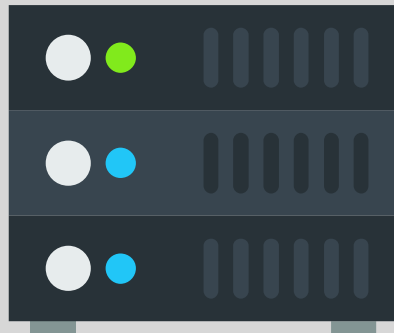
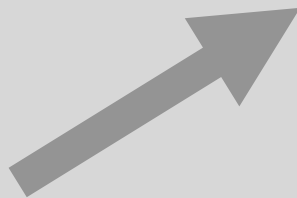
配置

Run

Task Definition
Release v1.0.0



Task Definition
Release v1.0.1



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

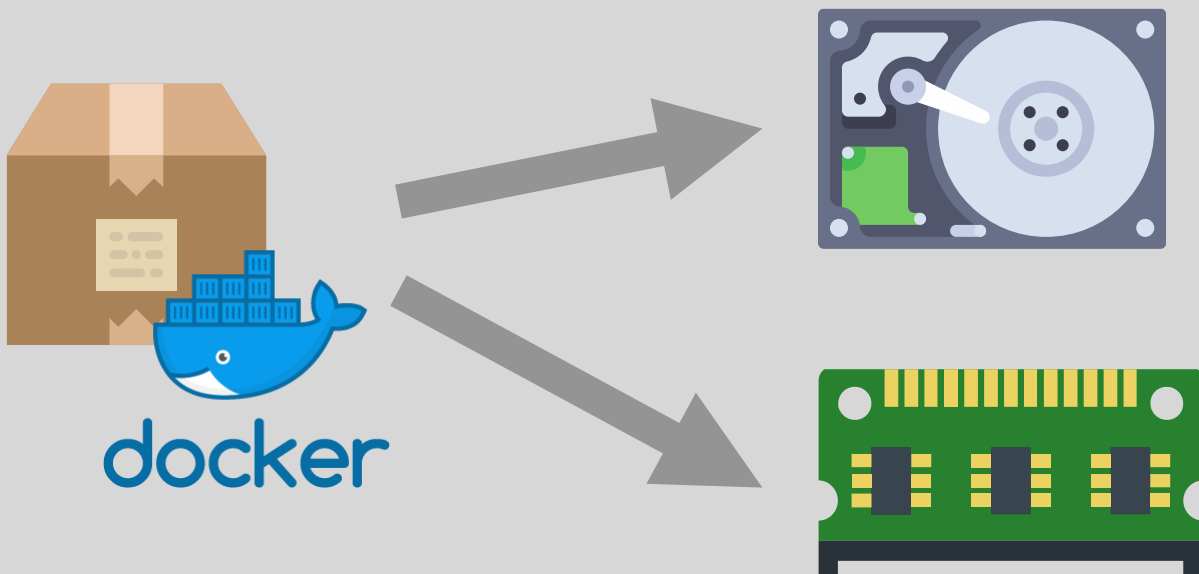
IX. 易处理

X. 开发环境与线上环境等价

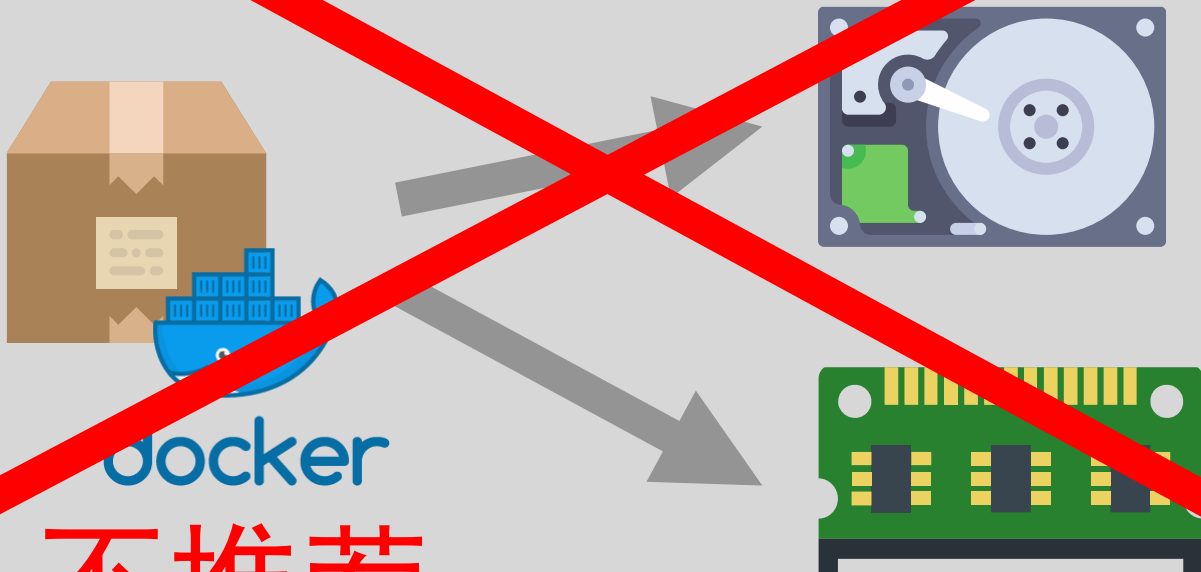
XI. 日志

XII. 管理进程

有状态容器将状态存储在本地磁盘或本地内存中。
工作负载最终绑定到具有状态数据的特定主机。



有状态容器将状态存储在本地磁盘或本地内存中。
工作负载最终绑定到具有状态数据的特定主机。



不推荐

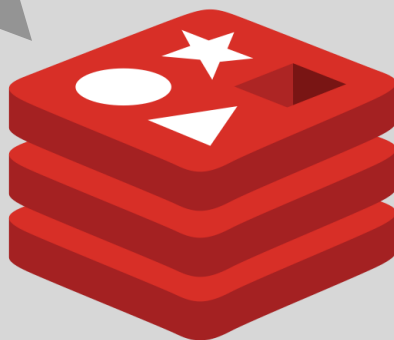


数据库

用于持久储存

MySQL, PostgreSQL,
Auroa

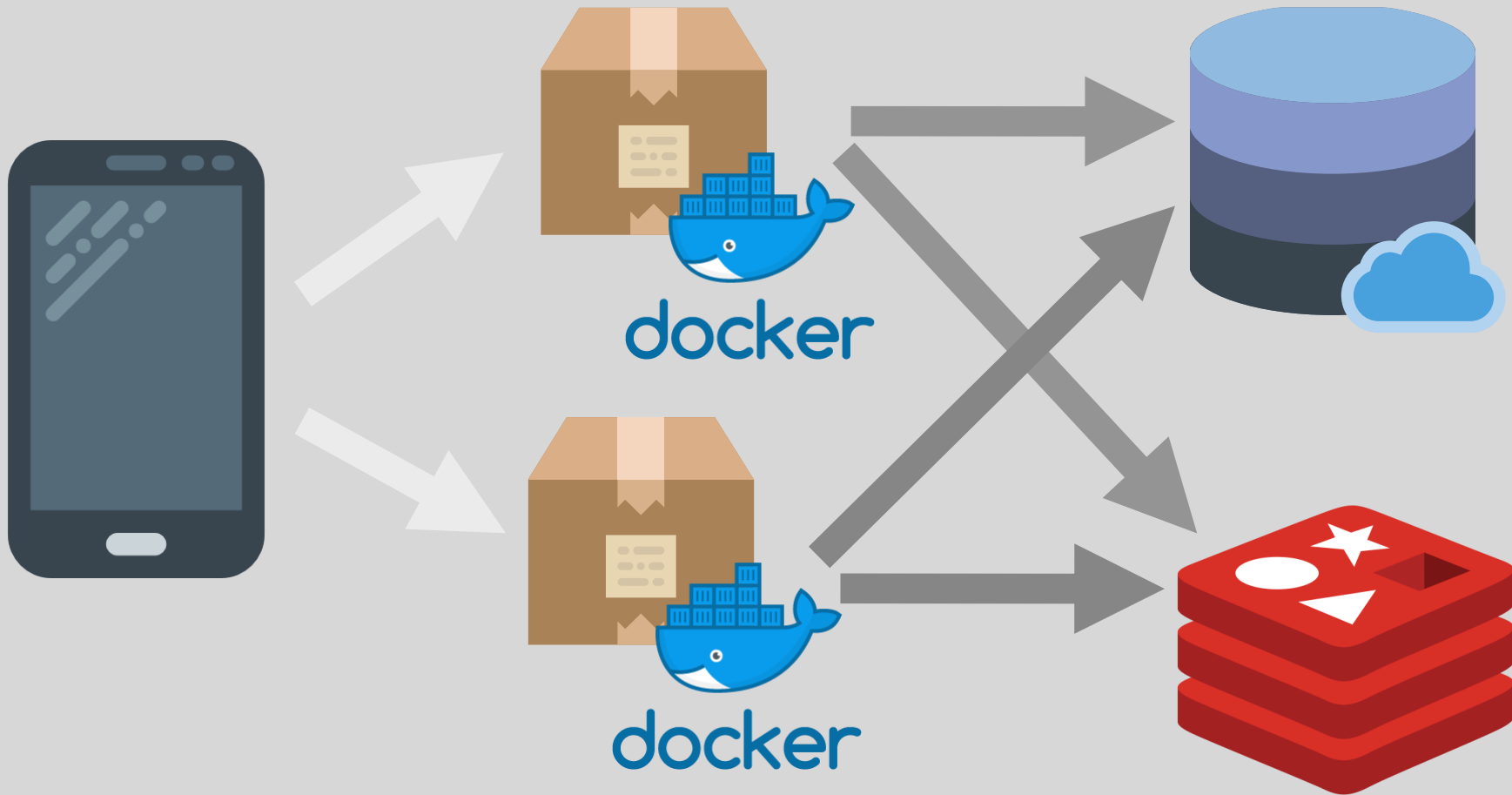
MongoDB, DynamoDB



缓存

高速的临时状态
储存

redis,
memcached



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程

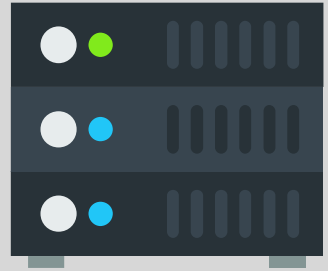
Port 32456



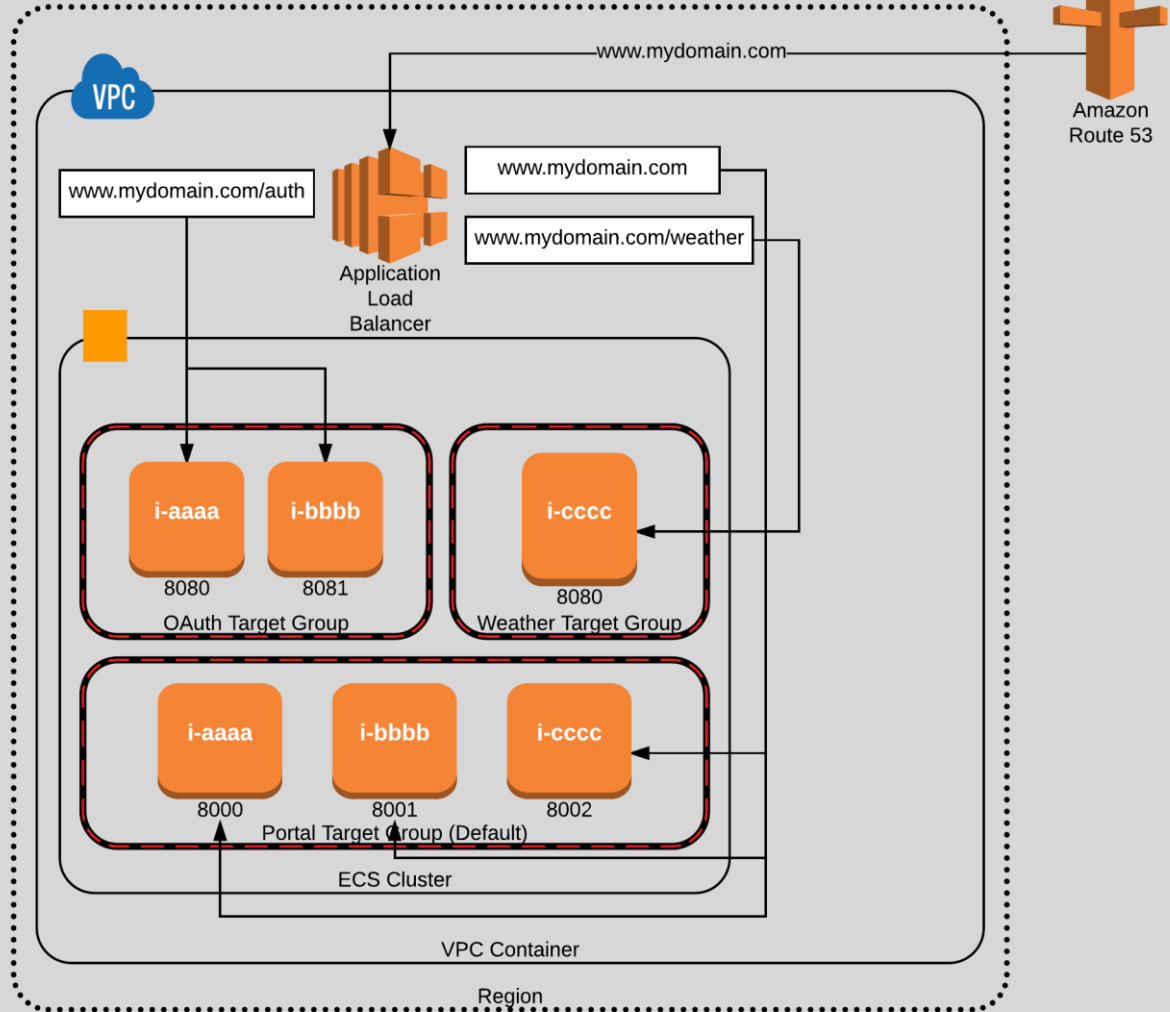
Port 32457



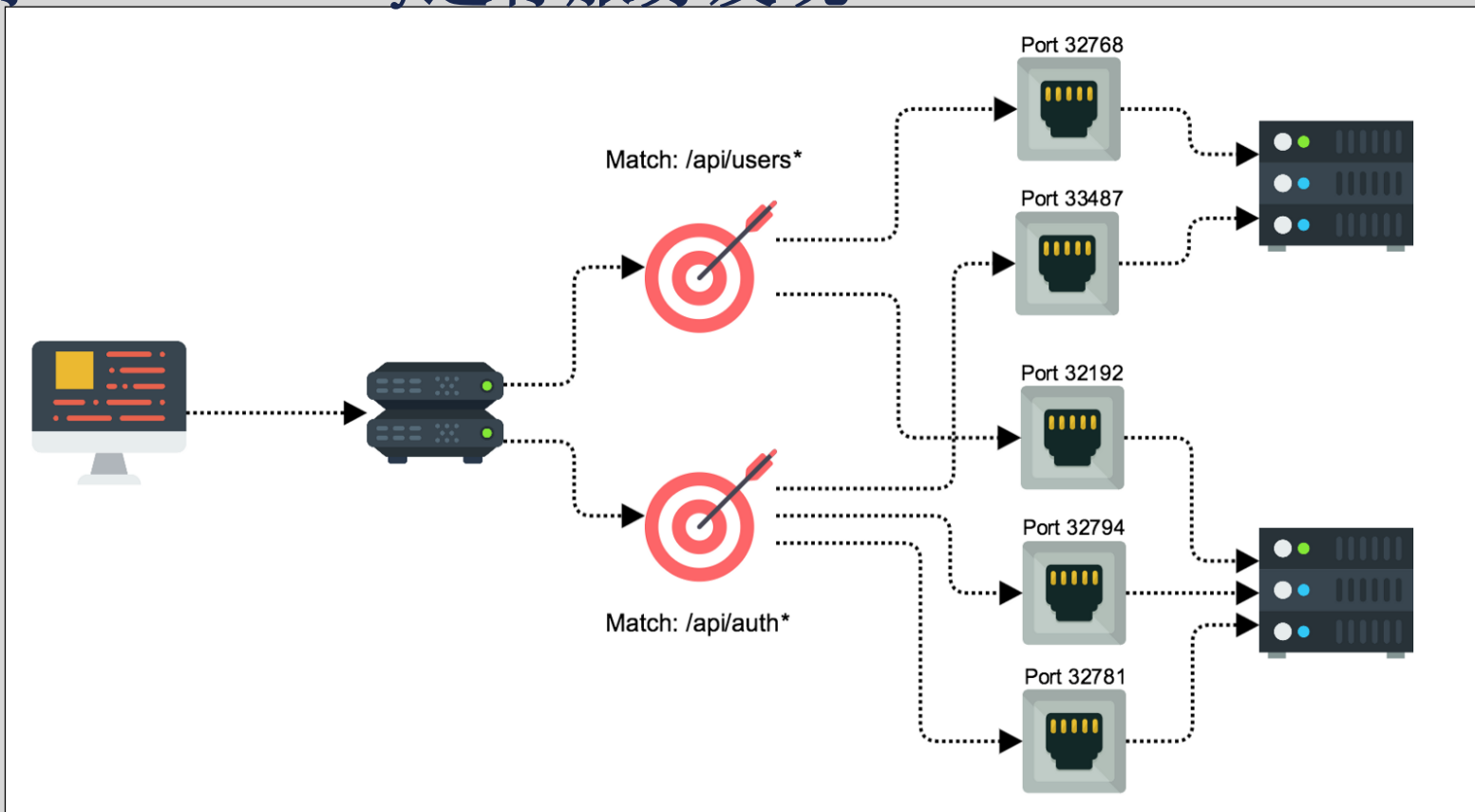
Port 32458



使用 Route53 和 Application Load Balancer 进行服务发现



使用API Gateway进行服务发现



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

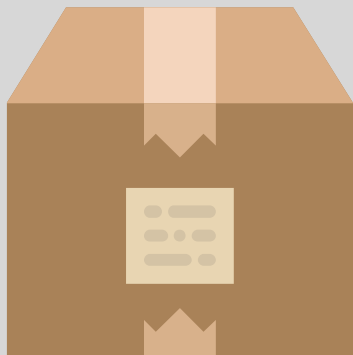
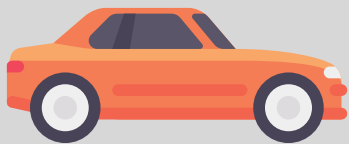
VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

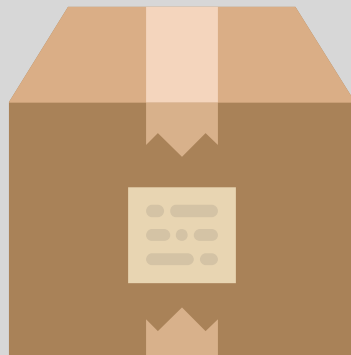
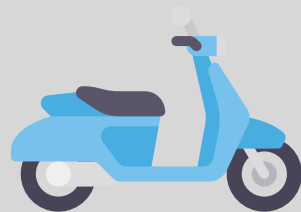
XII. 管理进程



API

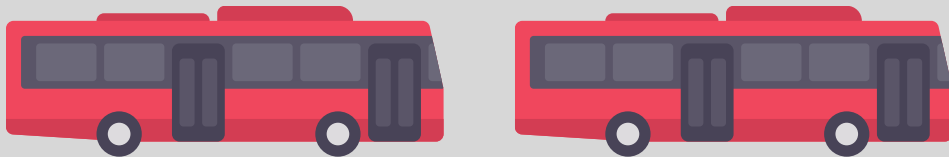


网站



后台

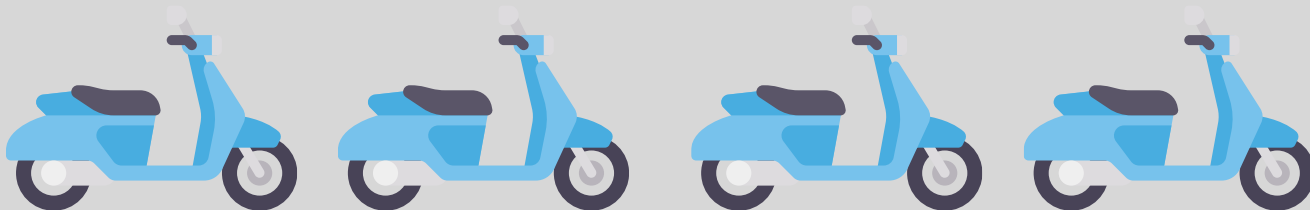
网站



API



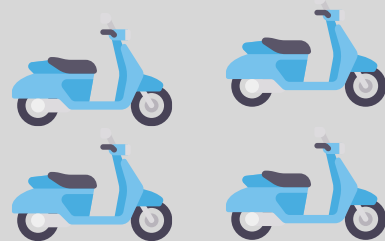
后台



主机



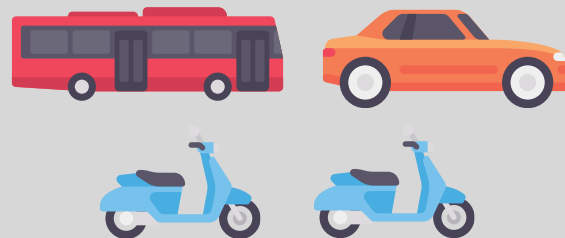
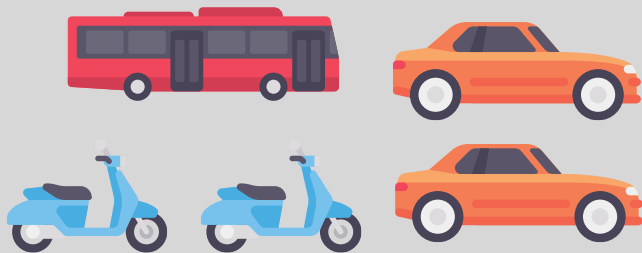
进程

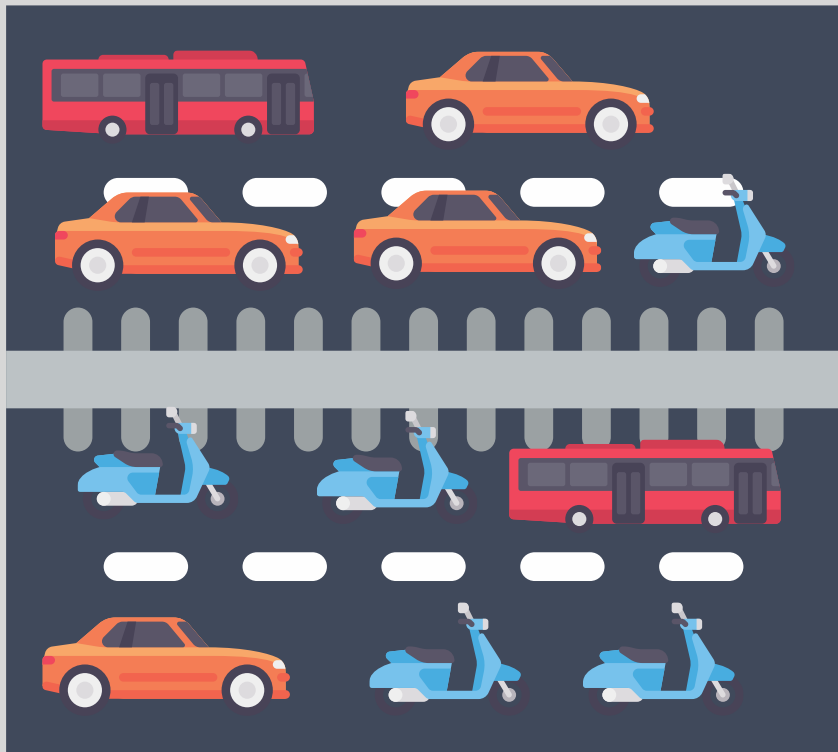


主机

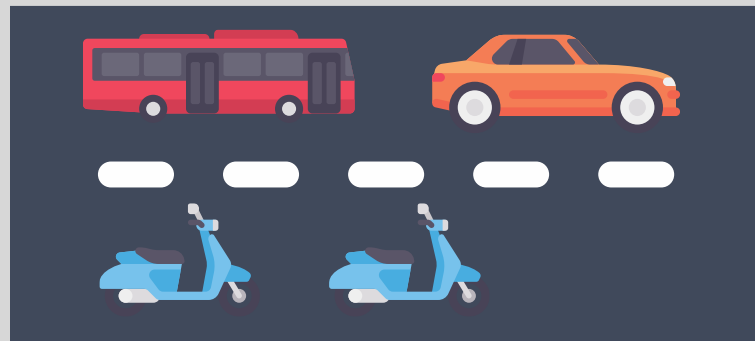


进程





大主机 = 多一点的并发



小主机 = 少一点的并发

十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

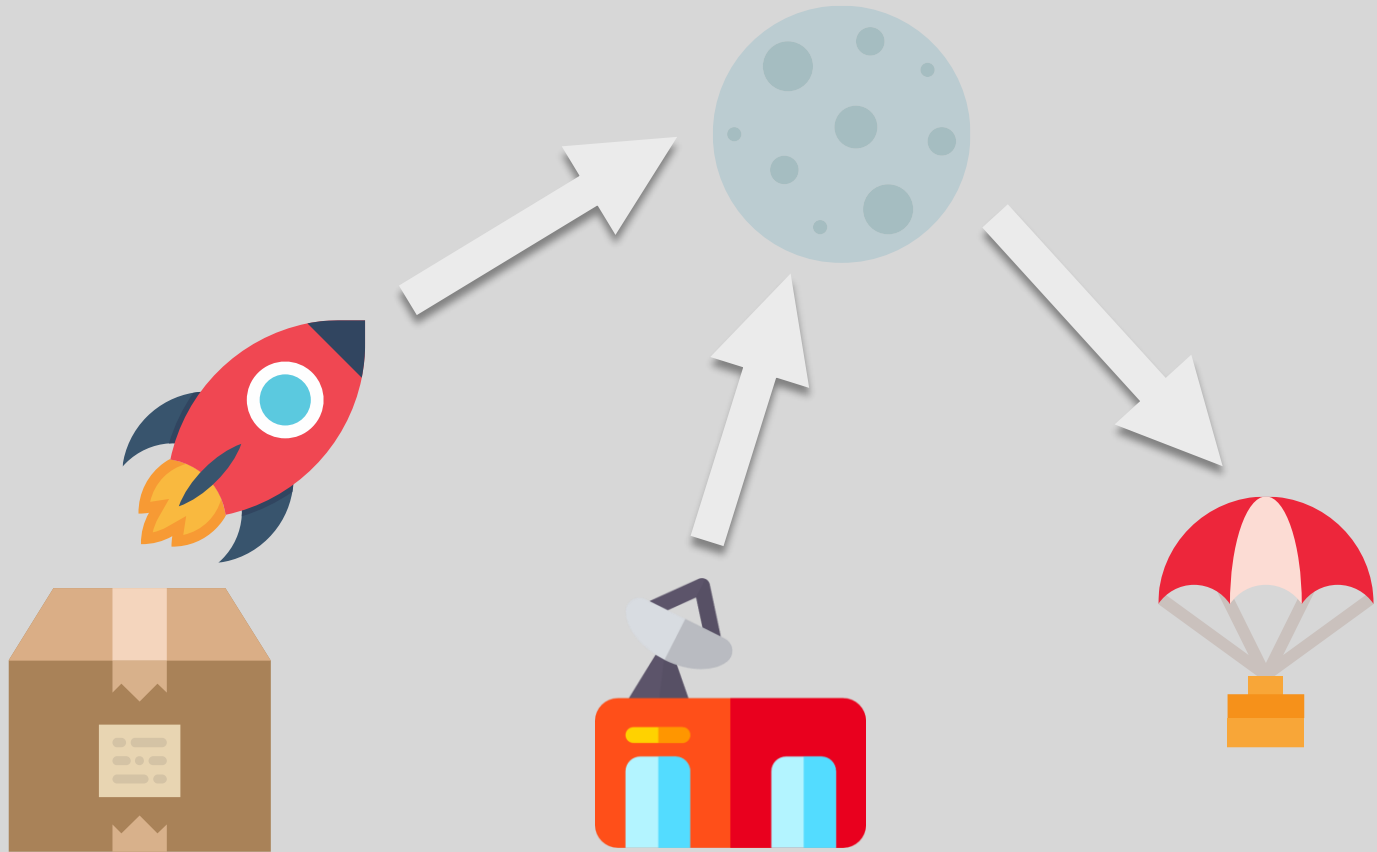
VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程



快速启动

敏捷响应

优雅关闭

快速启动



最大限度地减少进程的启动时间：

- 针对峰值快速扩展
- 能够根据需要将进程移动到另一个主机
- 更快地替换崩溃的进程

敏捷响应，优雅关闭



对SIGTERM做出响应，优雅地关闭

```
var server = app.listen(3000);

console.log('Message service started');

process.on('SIGTERM', function() {
  console.log('Shutting down message service');
  server.close();
});
```

十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程

开发环境1



开发环境2

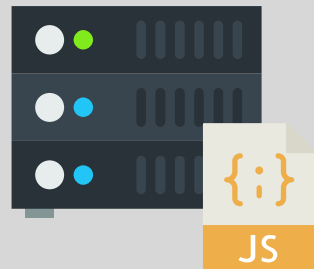


git

集成测试环境



生产环境



本地

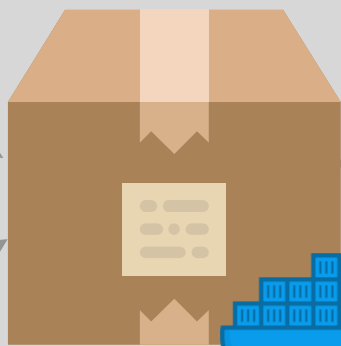
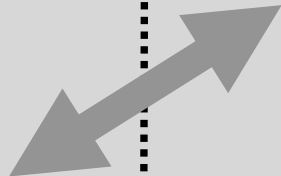
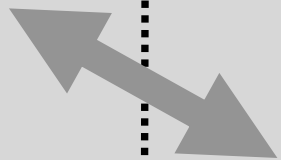
应用

远程

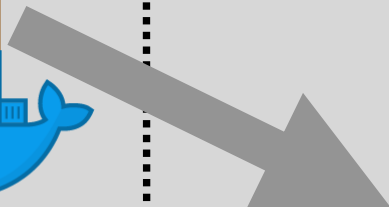
开发环境1



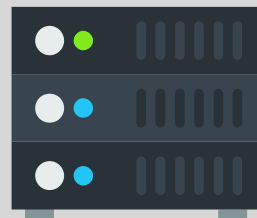
开发环境2



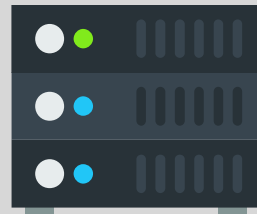
docker



集成测试环境



生产环境



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

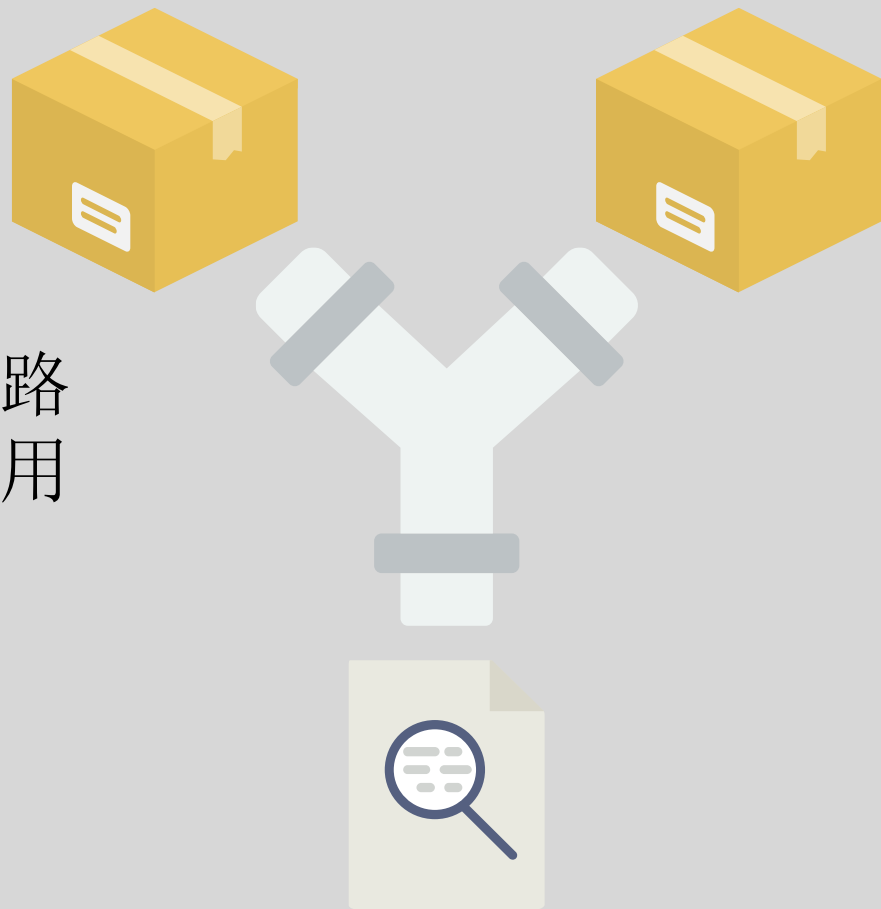
IX. 易处理

X. 开发环境与线上环境等价

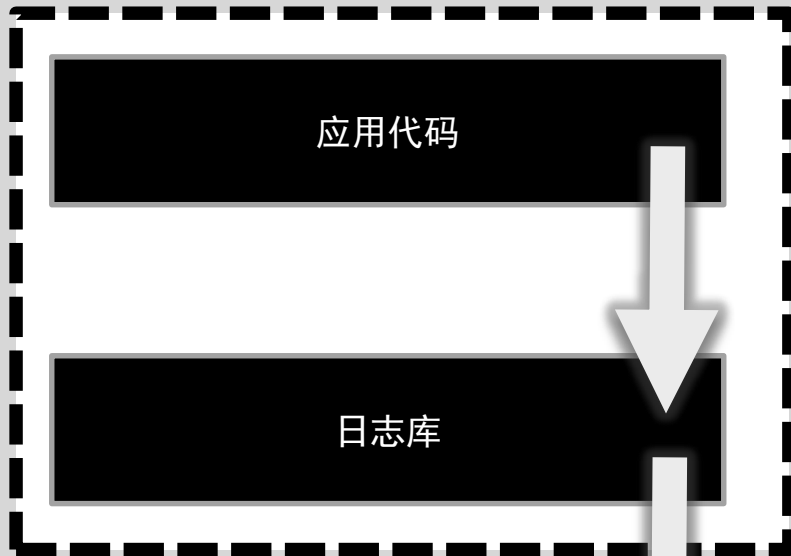
XI. 日志

XII. 管理进程

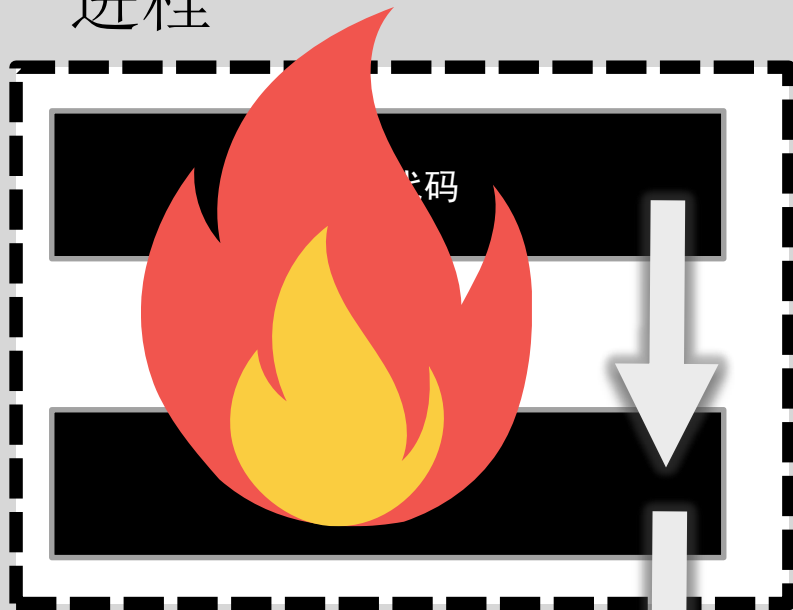
将日志视为事件流，并将路由和处理日志的逻辑与应用程序本身分开。



进程



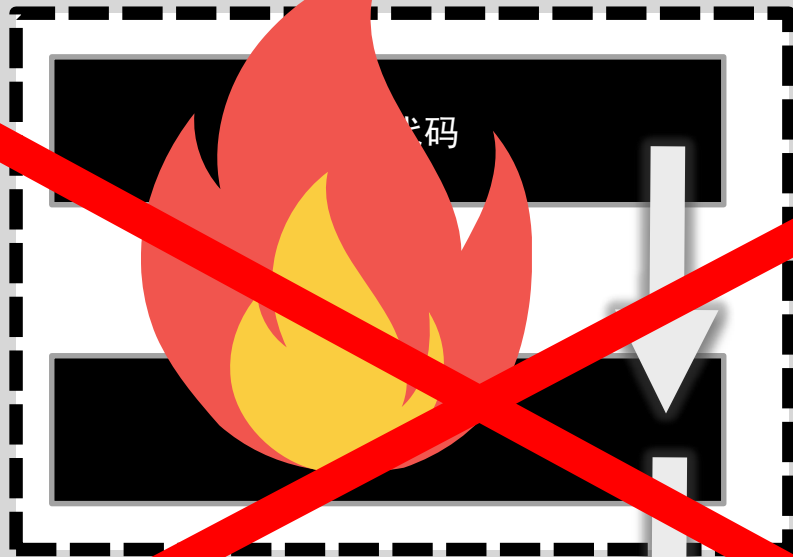
进程



如果有些日志尚未完全写入，则会丢失



进程



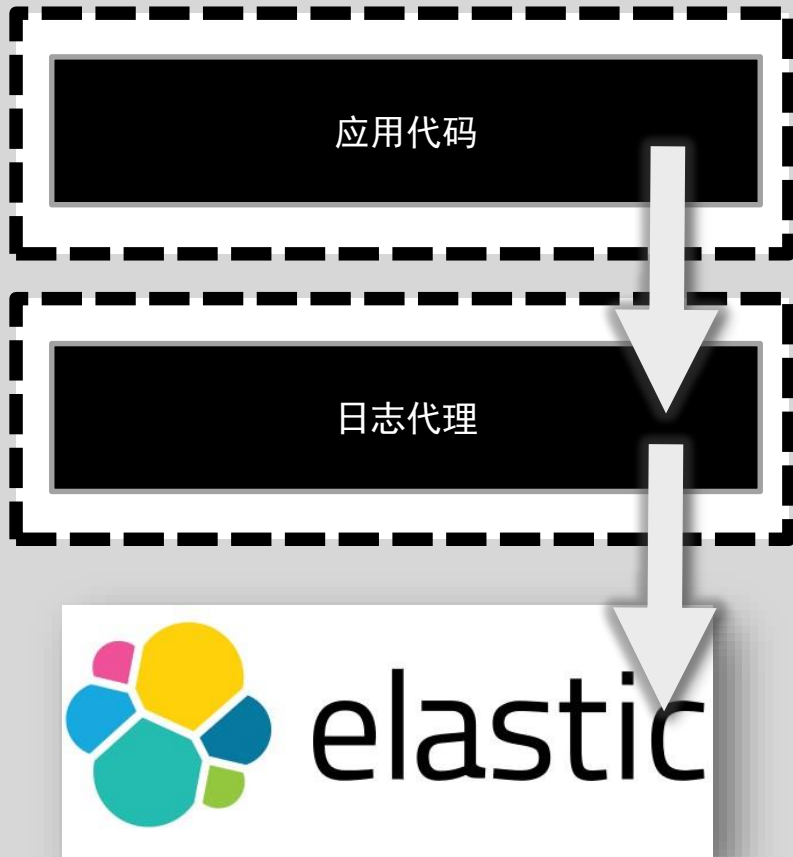
如果有些日志尚未完全写入，则会丢失

不推荐



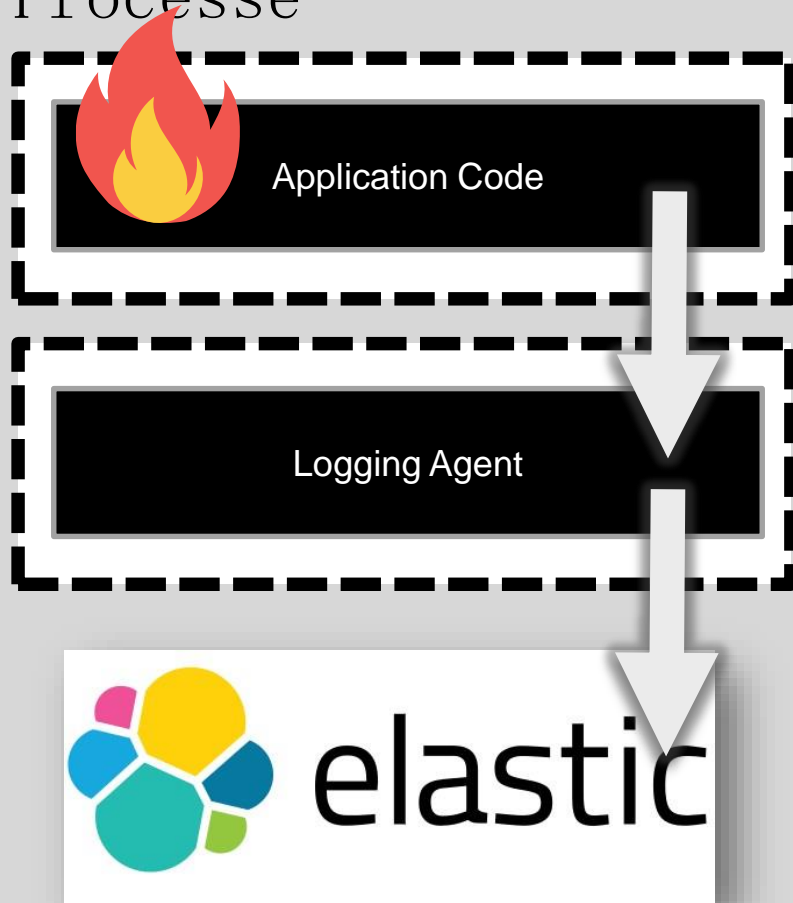
elastic

进程



将日志交由主机上的日志代理程序处理

Processe



日志依然可以到达代理程序，最后传递到日志收集系统

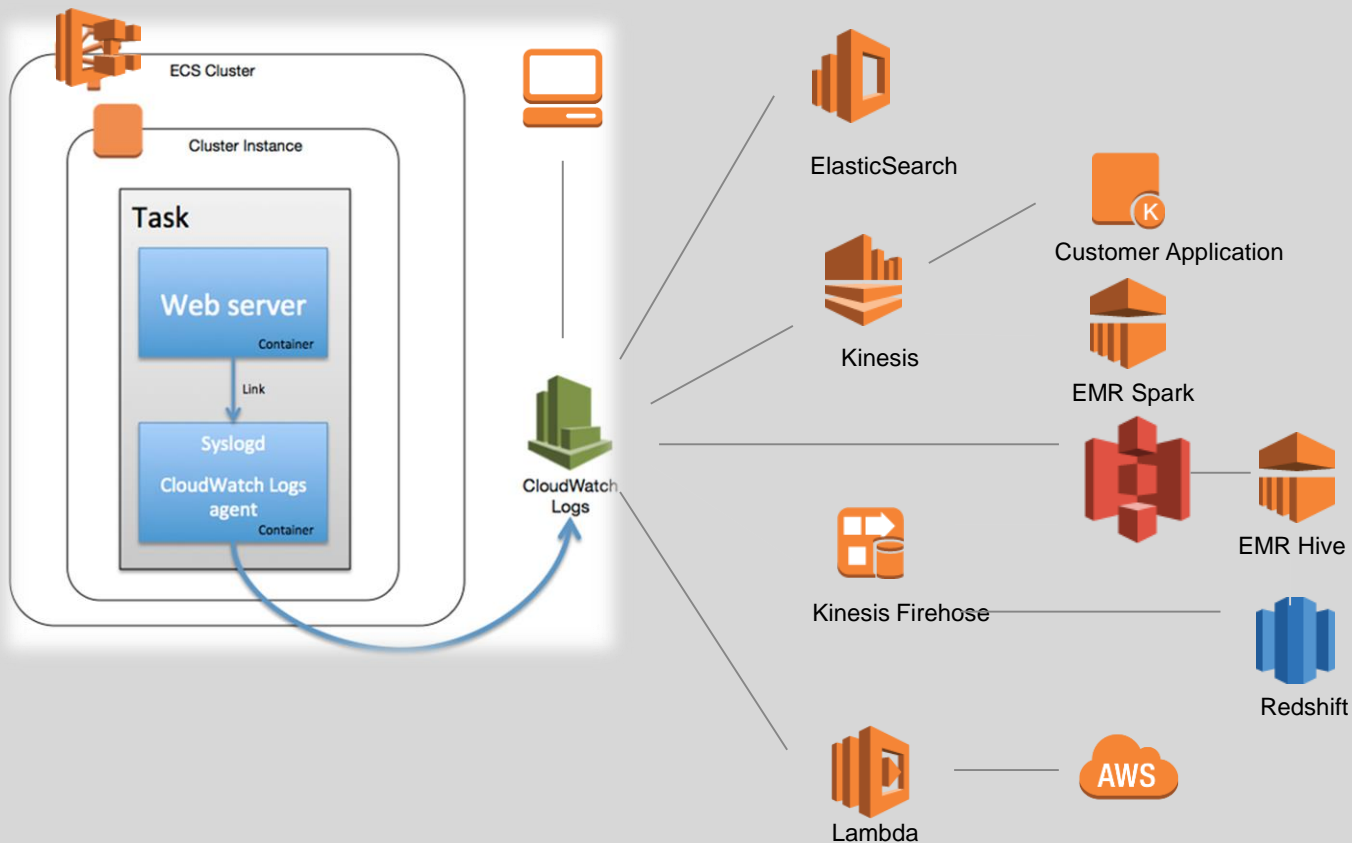
容器代码写入stdout

```
var express = require('express');  
var app = express();  
var logger = require('morgan');  
  
app.use(logger('tiny'));
```

Docker将容器的stdout储存到日志磁盘

```
docker run --log-driver awslogs myapp  
  
docker run --log-driver fluentd myapp
```

容器日志的储存和分析



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

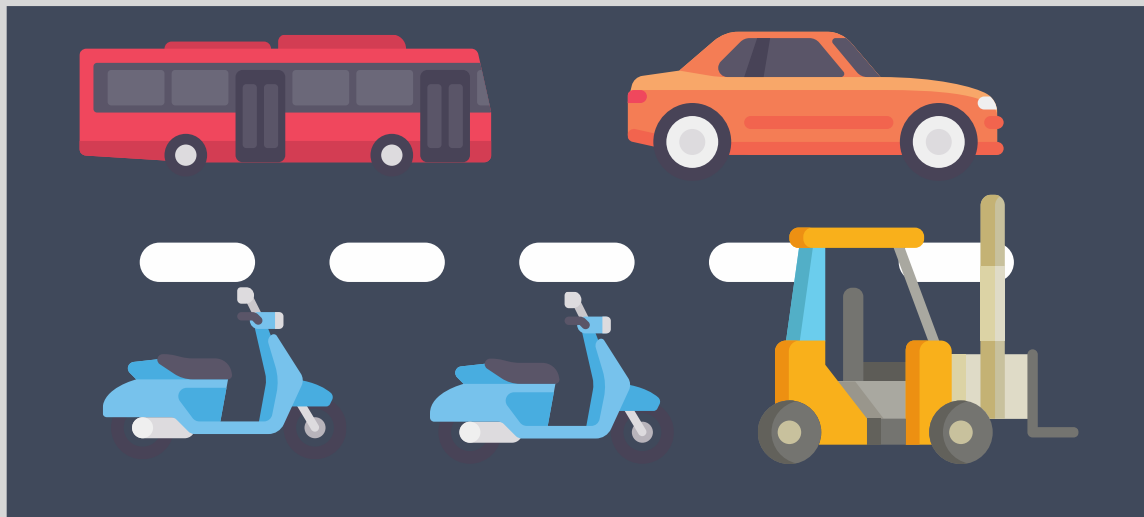
XI. 日志

XII. 管理进程



管理进程是不可或缺的：

- 迁移数据库
- 修复一些损坏的数据
- 每周一次将大于X的数据库记录移动到冷存储器
- 每天通过电子邮件向某人发送报告



像运行其他普通进程
一样运行管理进程

AWS与12要素应用



虚拟机



容器



无服务器化



虚拟机



容器



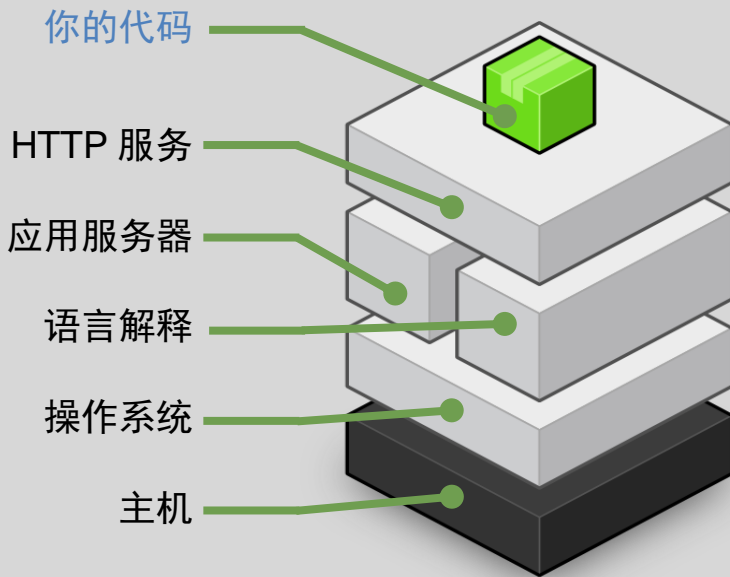
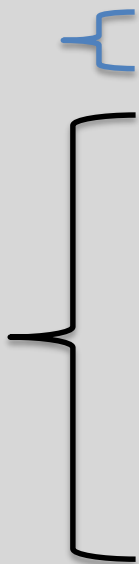
Lambda

	打包	更新	运行	运行时长
虚拟机	镜像	补丁	多线程,多进程	数小时到数月
容器	容器文件	版本控制	多线程,单进程	数分钟到数天
Lambda	代码	版本控制	单线程,单进程	数毫秒到数百秒

AWS Elastic Beanstalk

专注于构建你的应用程序

您只需上传代码，Elastic Beanstalk 即可自动处理包括容量预置、负载均衡、自动扩展和应用程序运行状况监控在内的部署工作。同时，您能够完全控制为应用程序提供支持的 AWS 资源，并可随时访问基础资源。



由您提供



由Elastic Beanstalk提供并进行管理

AWS容器服务一览

管理

部署, 编排, 扩展和管理
各种容器化的应用



Amazon Elastic
Container
Service



Amazon Elastic
Container
Service for
Kubernetes

宿主机

容器运行所在



Amazon EC2



AWS Fargate

镜像注册

容器镜像注册



Amazon Elastic
Container
Registry

不仅仅有容器



无服务器计算

AWS Lambda

无服务器的微服务

AWS Lambda + Amazon API Gateway

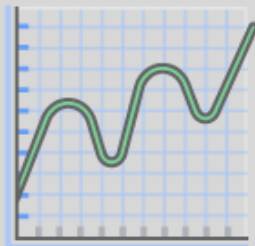
可以快速构建微服务

- 事件响应 一个函数对应一类事件
- 无服务器化的API后端 一个函数对应一个API
- 数据处理 一个函数对应一种数据类型

AWS Lambda的优势



无需管理服务器



持续扩展



无需为闲置资源付费

无服务器计算不仅仅是Lambda



AWS Lambda



Amazon
DynamoDB



Amazon SQS



Amazon
Kinesis Streams



Amazon API
Gateway



Amazon
ElastiCache



Amazon SNS



Amazon
Elasticsearch



Amazon S3



Amazon Athena



Amazon SES



Amazon
CloudSearch



AWS X-Ray

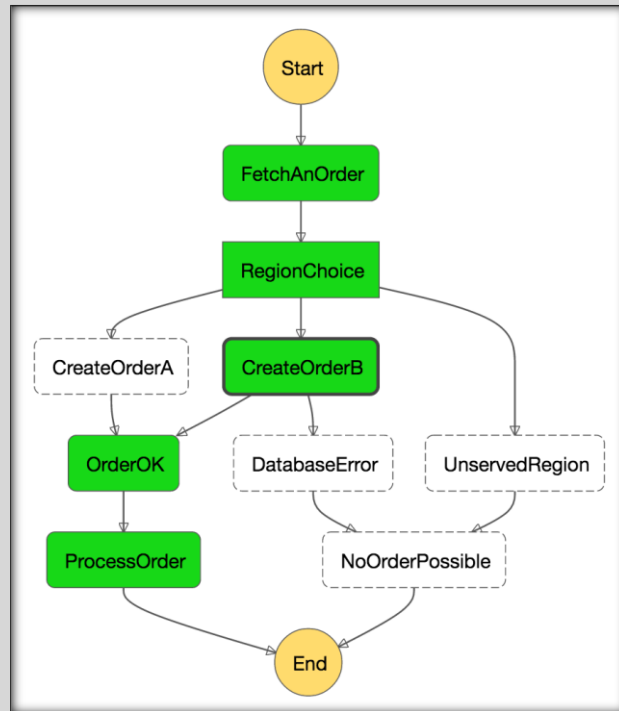


AWS CloudFront

AWS Step Functions



使用可视化的工作流协调分布式应用程序的组件



十二要素应用宣言

I. 基准代码

II. 依赖

III. 配置

IV. 后端服务

V. 构建、发布、运行

VI. 进程

VII. 端口绑定

VIII. 并发

IX. 易处理

X. 开发环境与线上环境等价

XI. 日志

XII. 管理进程

谢谢大家!

扫码下载演讲资料

