电商行业开发运维之路

田明晶 AWS解决方案架构师

2017.4.8

主要议题

- 电商/O2O行业 业务特征及技术特征
- 微服务与容器技术
- DevOps与AWS
- AWS的容器服务 - ECS
- 客户案例分享

电商/O2O行业 业务特征及技术特征

电商行业的业务特征

外贸商户/工厂

持续增量订单

交易服务一体化平台

DHgate & DHport

降低交易成本

批发商/零售商

海外市场拓展服务

跨境支付 DHpay

综合物流服务 DHlink

互联网金融服务 DHfinet

社交商务 Socialshops

其他增值服务

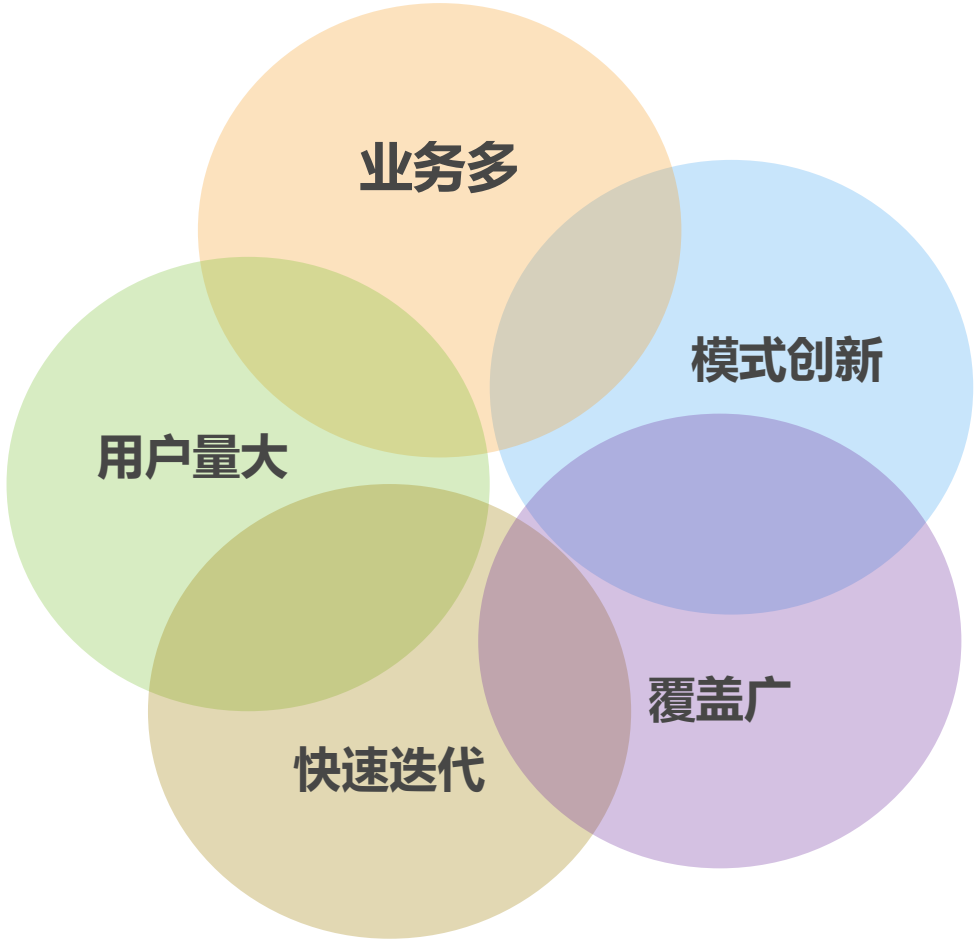
约120万家国内供应商；
1000万买家
遍布全球230个国家和地区

国际合作伙伴
物流&支付

8个多语言
平台

4000万
在线产品

电商行业的技术特点



高度可扩展

高并发

全球部署

高并发

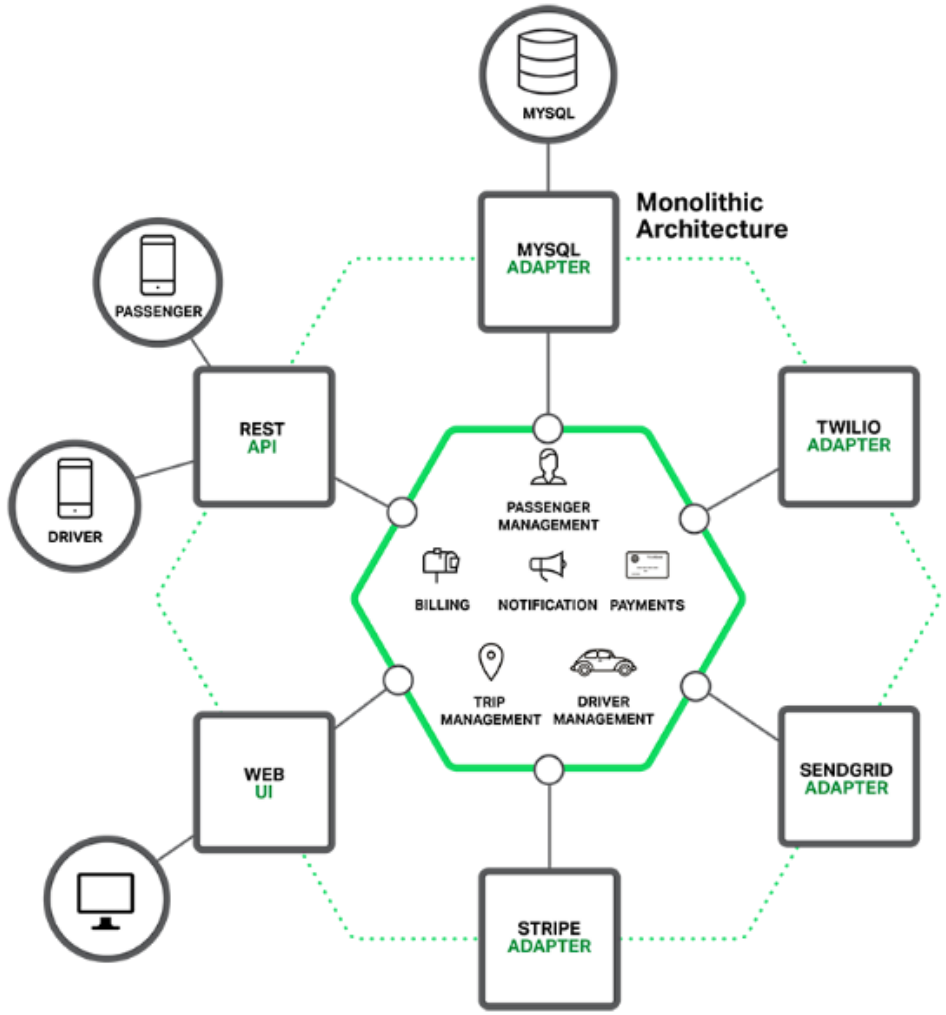
可管理

可监控

微服务与容器技术

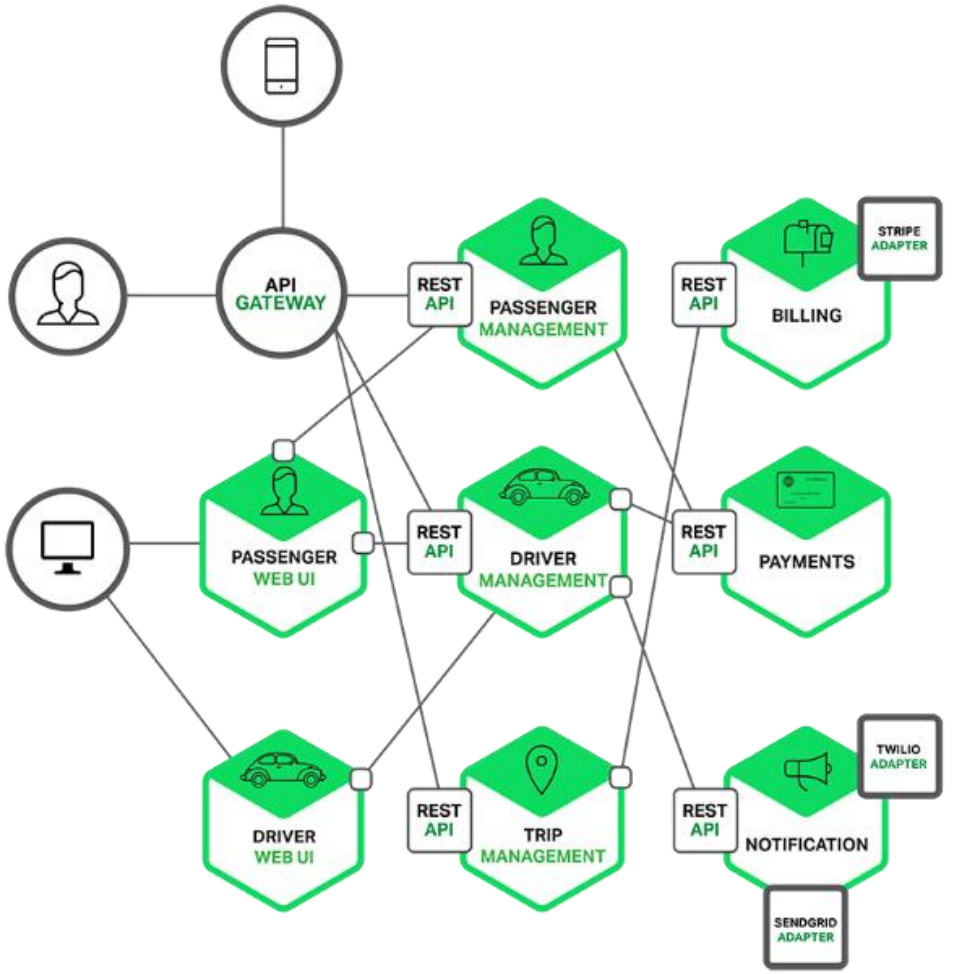
单体式应用

- 持续开发问题
- 模块扩展问题
- 可靠性问题
- 采用新架构和语言非常困难



微服务架构

- 易于开发/理解/维护
- 便与使用新开发技术
- 独立的部署/测试
- 易于容错



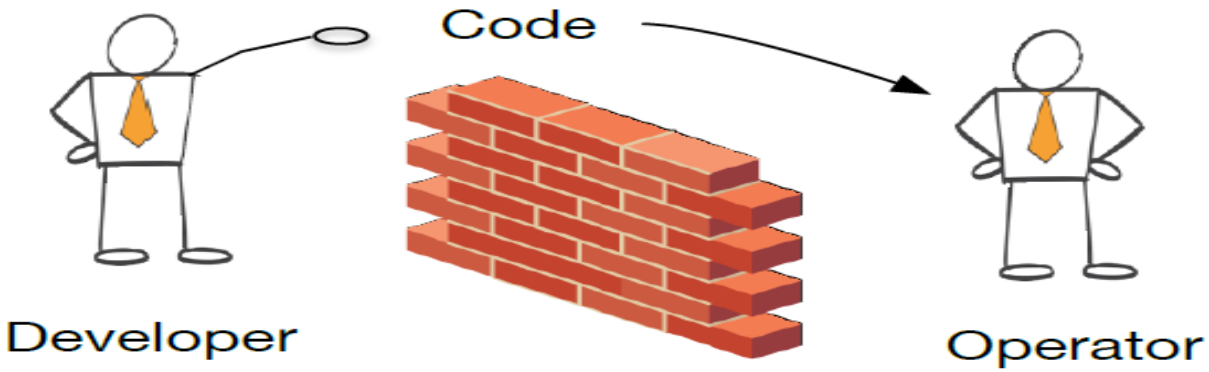
为什么要用容器

- **更高效的利用系统资源**
  - 不需要进行硬件虚拟，不需要运行完整操作系统
  - 可以运行更多数量的应用
- **更快速的启动时间（毫秒级）**
  - 创建新的目录
  - 装填文件系统
  - 设置网络，文件系统挂载点
  - 启动进程
- **一致的运行环境**
  - 除了内核，运行时环境保持一致
- **持续交付和部署（DevOps）**
  - Dockerfile
  - CI/CD
- **更轻松的迁移**
  - 物理机、虚拟机、公有云、私有云，甚至是笔记本
- **更轻松的维护和扩展**
  - 分层存储以及镜像的技术

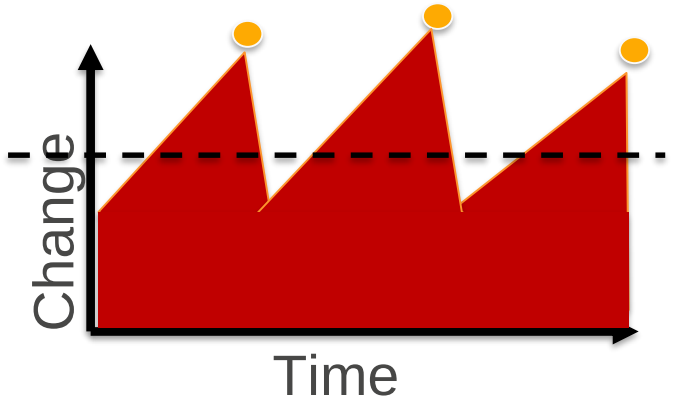
DevOps与AWS

为什么需要DevOps?

因为我们不希望
事情是这样的...

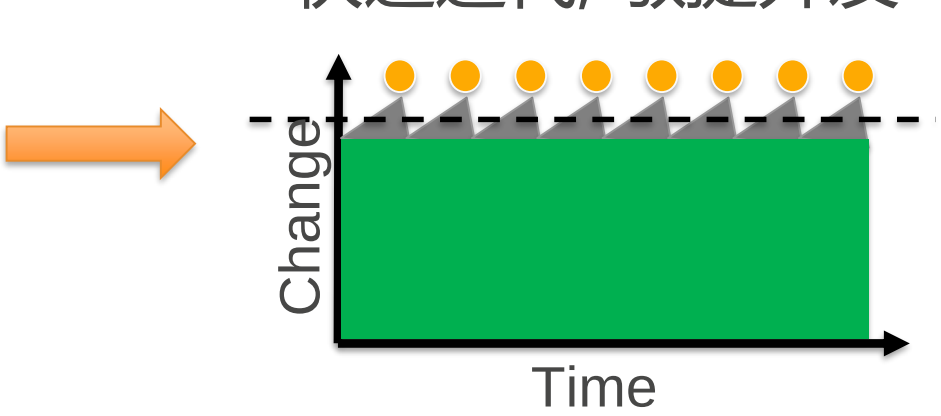


瀑布式开发，版本发布少

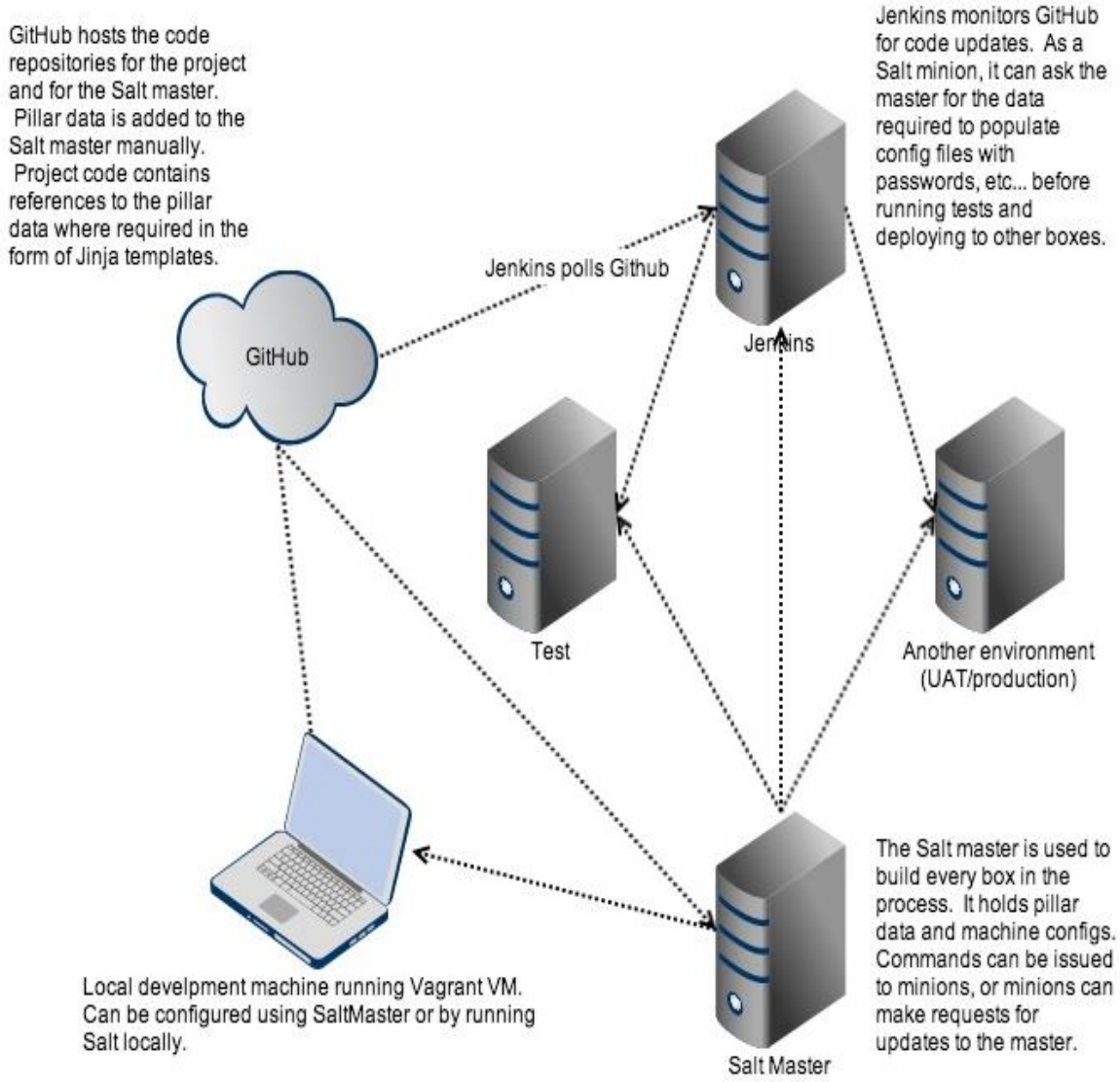


我们希望...

快速迭代, 敏捷开发

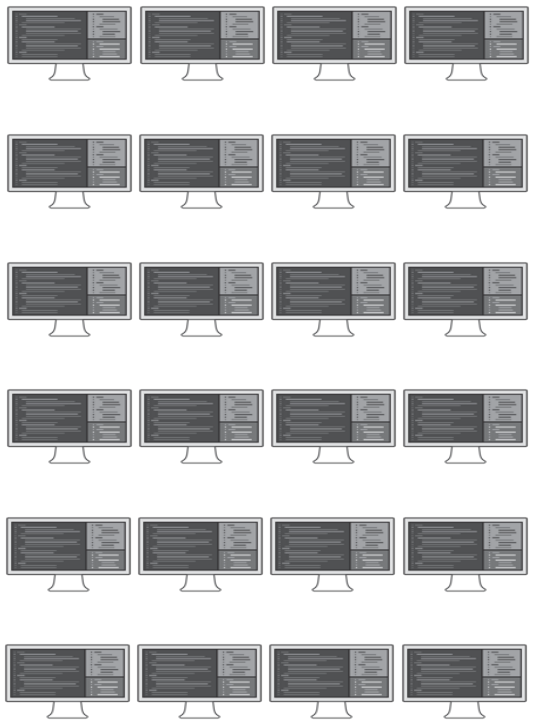


一个完整的 DevOps 开发流程



1. **提交（Commit）**：工程师将程式在本地测试过后，提交到版本控制系统，如Git等。
2. **建置（Build）**：持续整合系统，如 Jenkins CI，侦测到版本更新，便自动从 Git 里提取出最新的程式，进行建置（Build）。Jenkins 以 SaltStack 的 Minion 形式架构起来，意味着可以透过 SaltStack 的 Master 或其他的 Minions 取得更多的所需讯息来完成建置。
3. **单元测试（Unit Test）**：Jenkins 完成配置后，会自动执行指定的单元测试。
4. **部署到测试环境（Deploy to Test Environment）**：完成单元测试后，Jenkins 可将程式部署到跟应用环境（Production Environment）相近的测试环境进行最后测试。
5. **最终测试**：在测试环境里，可以进行一些最后的自动化测试，例如一些 Selenium 测试。以及跟实际情况类似的测试，可由开发人员或客户手动进行。
6. **部署到应用环境（Deploy to Production Environment）**：通过所有测试后，便可将最新版本部署到实际应用环境里。

单体Monolithic开发生命周期



开发者

app

交付渠道

微服务Microservice开发生命周期

开发者

services

交付渠道

亚马逊开发团队最佳实践

两个比萨饼团队

自主负责

承担责任

定向激励

“DevOps”

亚马逊开发团队最佳实践

数千个团队
× 微服务架构
× 持续交付
× 多环境部署

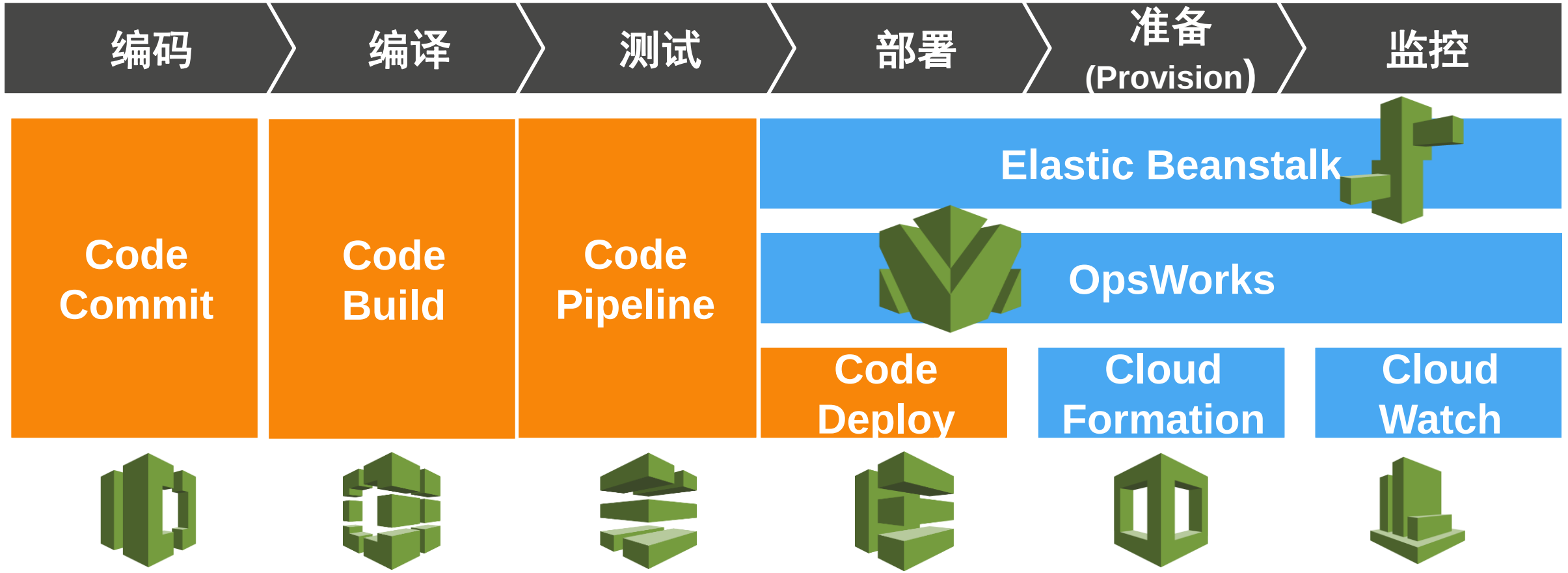
= 每年5千万次交付

- 基于2014年
- 平均1.5次部署/秒

AWS中国（北京）区域由光环新网运营

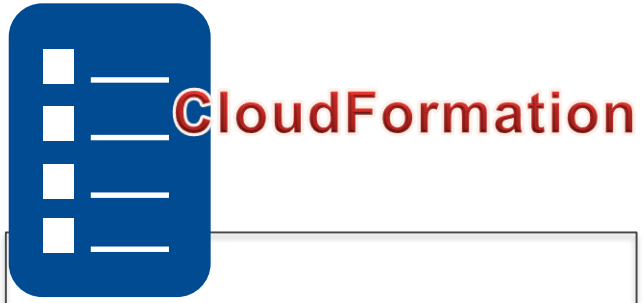
怎么实现DevOps？

基于AWS的DevOps实践要素



基于模板的快速部署 - CloudFormation

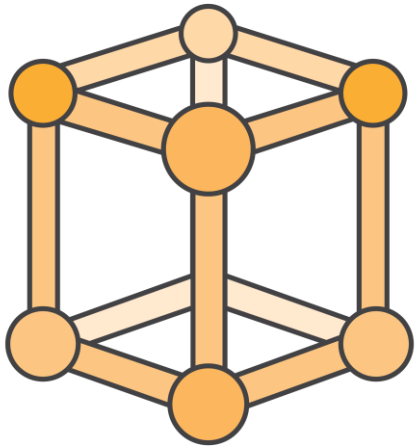
-- 在数分钟内将您的应用或应用堆栈复制到全球AWS站点



AWS中国（北京）区域由光环新网运营

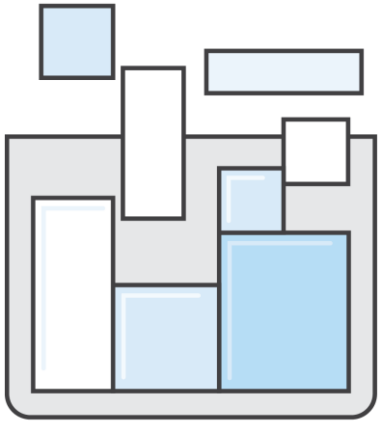
AWS的容器服务 - ECS

什么是Amazon ECS



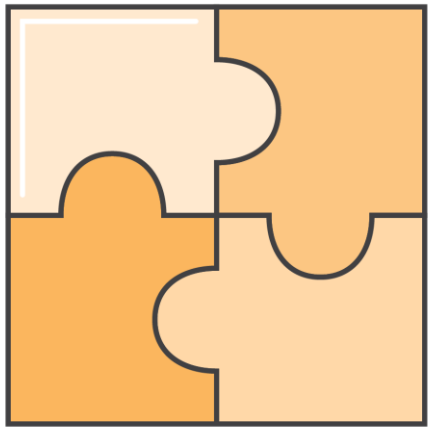
高效的容器管理

- 简单易用
- 高性能
- 大规模
- 安全性



灵活的容器部署

- 紧凑型
- 均衡型
- 亲和型
- 自定义



AWS平台集成

Application Load Balancing

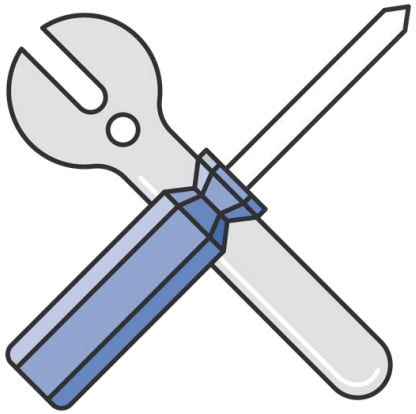
Amazon Elastic Block Store

Amazon Virtual Private Cloud

Amazon CloudWatch

AWS IAM

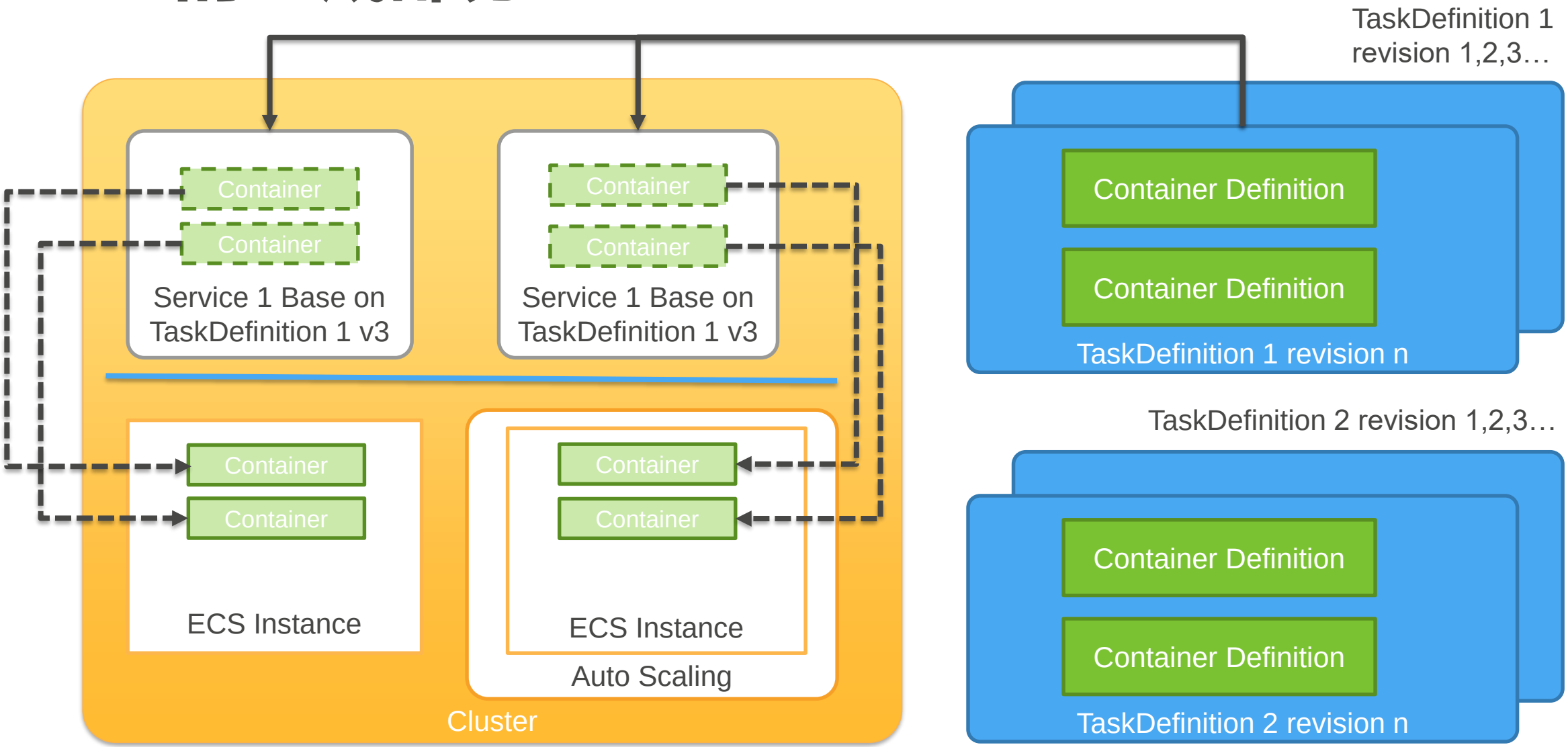
AWS CloudTrail



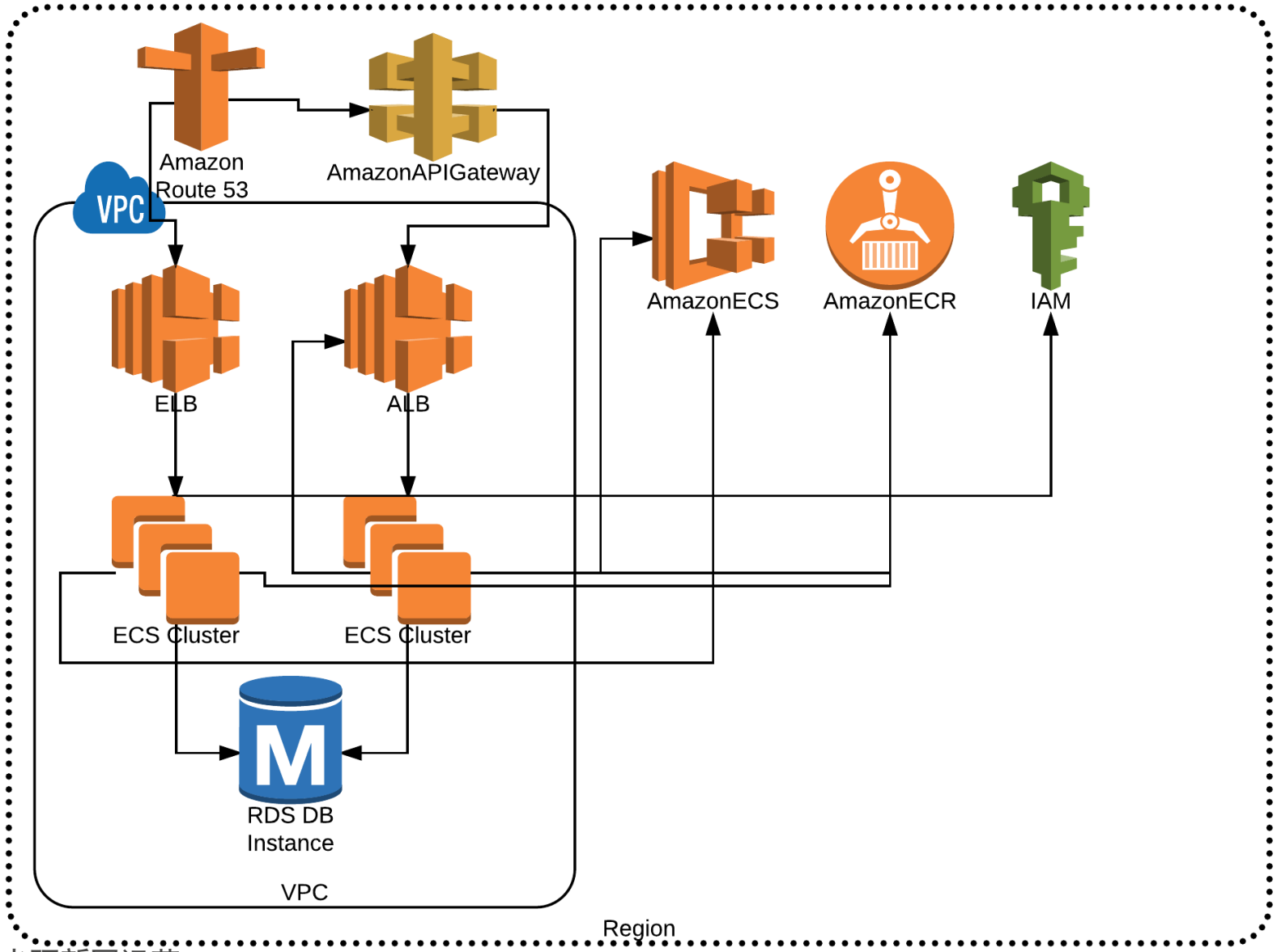
为扩展性设计

- 灵活/可定制的调度算法
- 广泛的API
- 开源代理和CLI

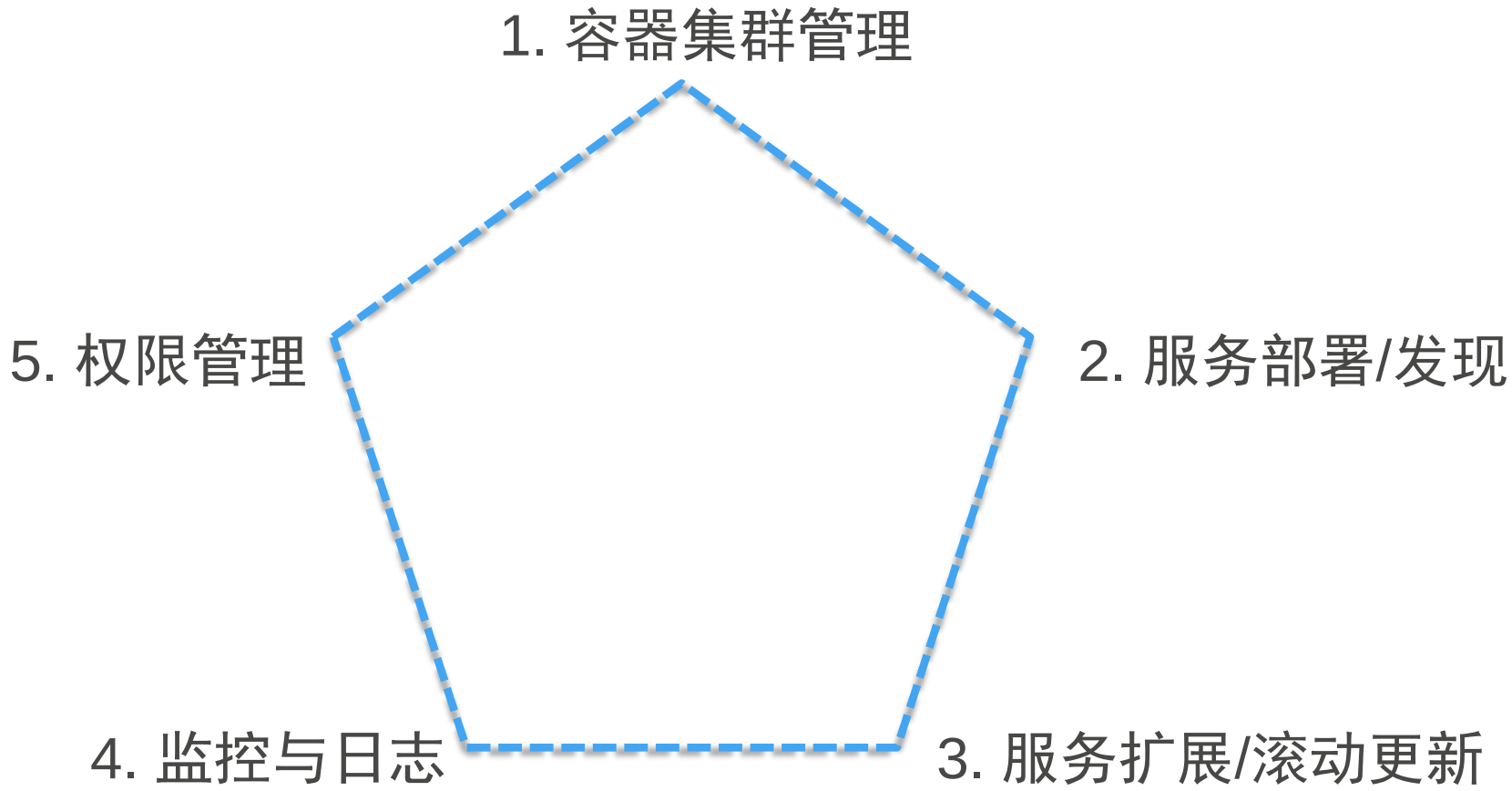
ECS的组成部分


TaskDefinition 1
revision 1,2,3…
Container
Container
Service 1 Base on
TaskDefinition 1 v3
Container
Container
Service 1 Base on
TaskDefinition 1 v3
Container Definition
Container Definition
TaskDefinition 1 revision n
TaskDefinition 2 revision 1,2,3…
Container
Container
ECS Instance
Container
Container
ECS Instance
Auto Scaling
Cluster
Container Definition
Container Definition
TaskDefinition 2 revision n

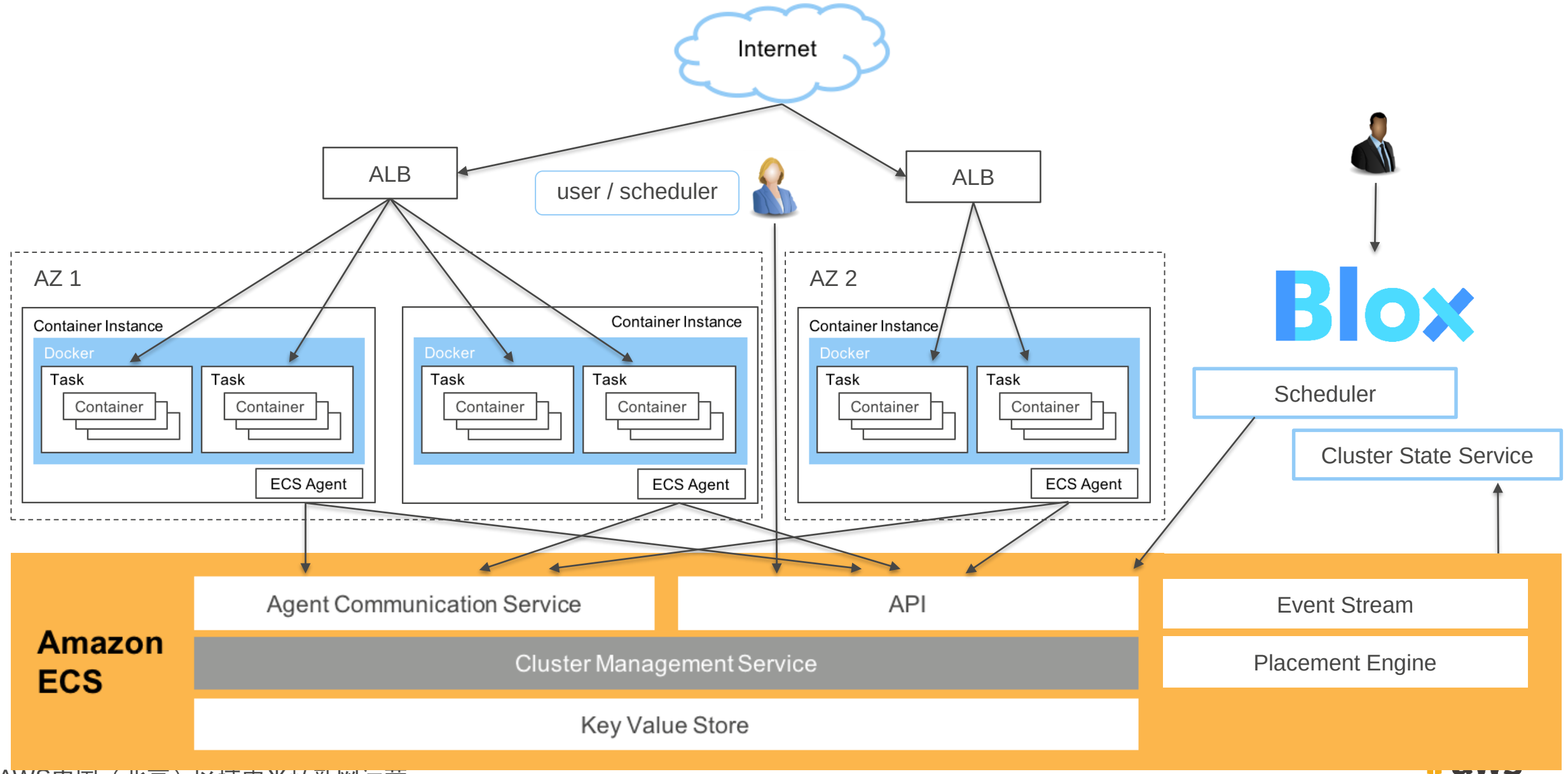
ECS上的微服务架构


Amazon
Route 53
AmazonAPIGateway
VPC
ELB
ALB
AmazonECS
AmazonECR
IAM
ECS Cluster
ECS Cluster
RDS DB
Instance
VPC
Region

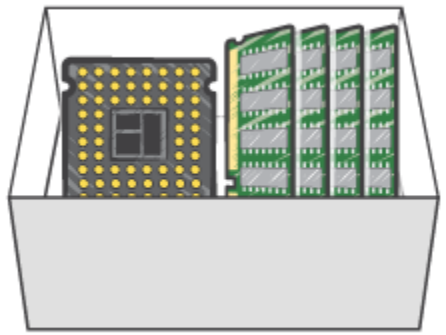
构建容器应用的要素



1. 容器集群管理

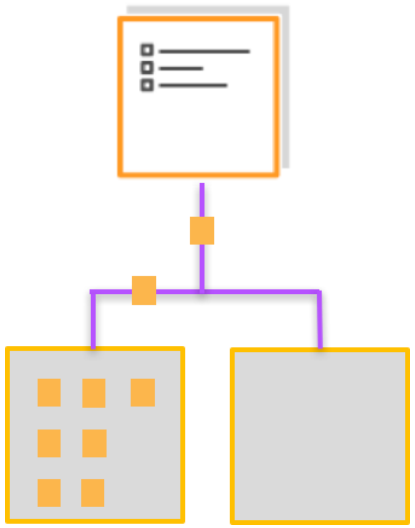

Internet
ALB
ALB
user / scheduler
AZ 1
AZ 2
Container Instance
Docker
Task
Container
ECS Agent
Blox
Scheduler
Cluster State Service
Amazon ECS
Agent Communication Service
API
Event Stream
Cluster Management Service
Placement Engine
Key Value Store

置放引擎 - 置放约束和属性

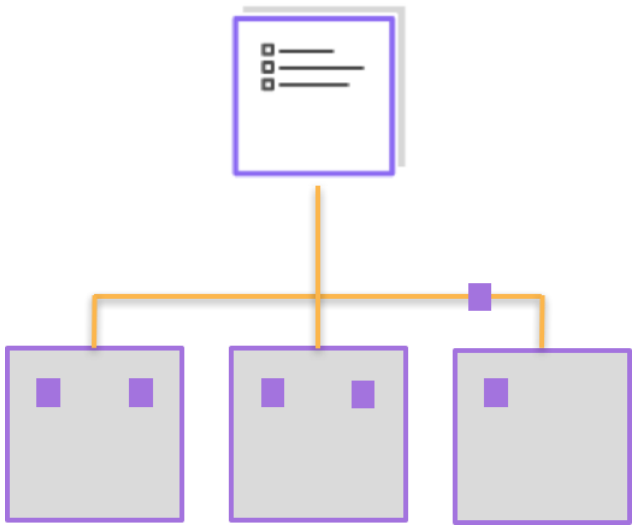


	名称	示例
✓	AMI ID	attribute:ecs.ami-id == ami-eca289fb
✓	Availability Zone	attribute:ecs.availability-zone == us-east-1a
✓	Instance Type	attribute:ecs.instance-type == t2.small
✓	Distinct Instances	type="distinctInstance"
✓	Custom	attribute:stack == prod

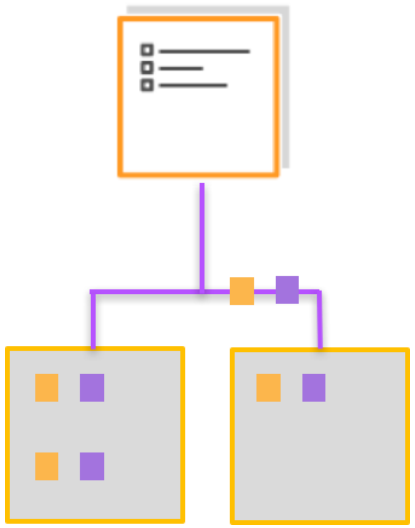
默认支持的置放策略



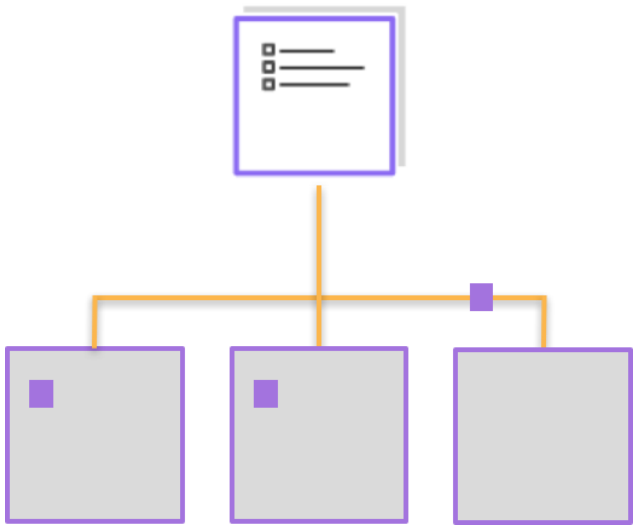
Binpacking



Spread

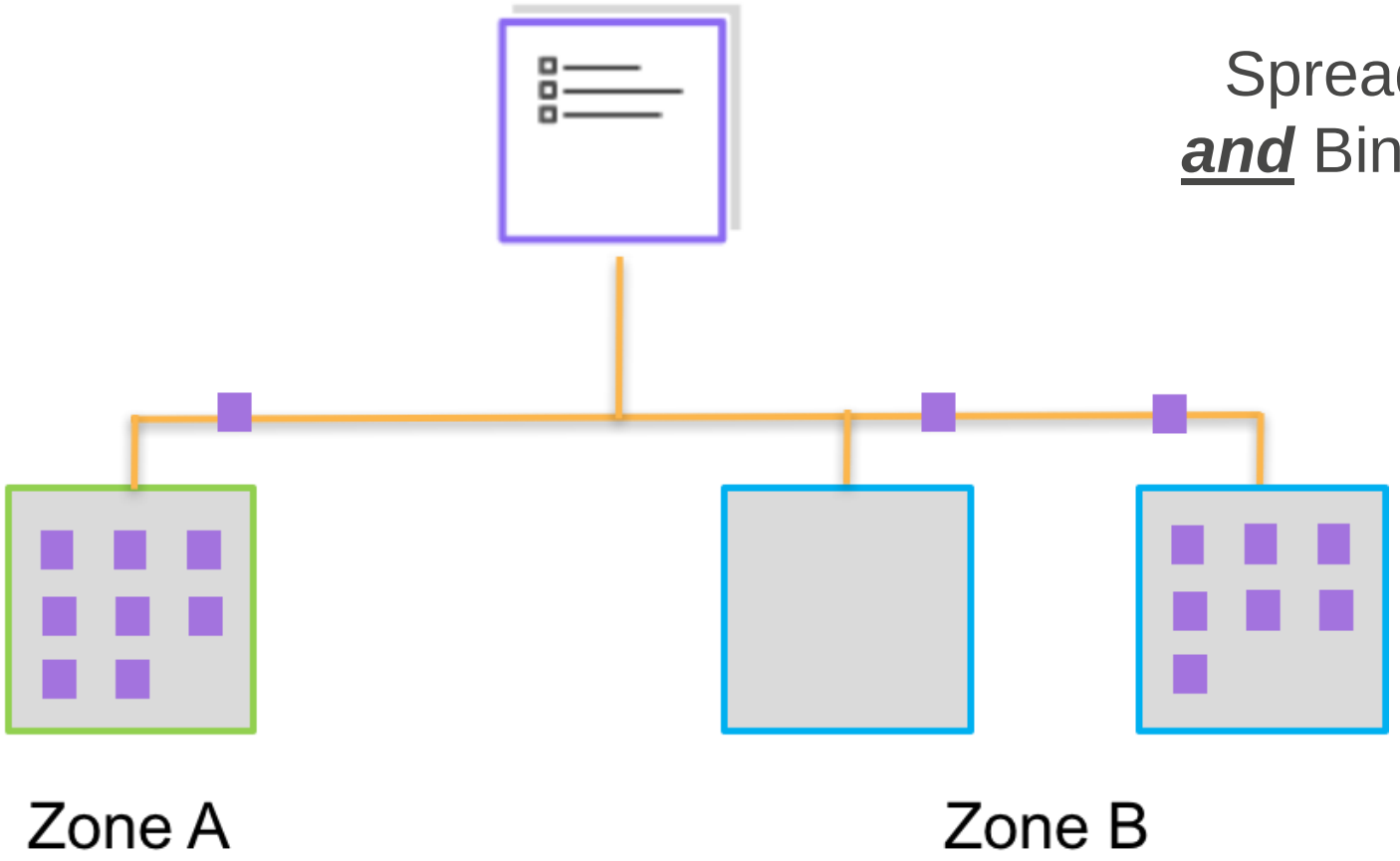


Affinity



Distinct Instance

置放策略叠加



Spread tasks across Zones
and Binpack within each Zone

2. 服务部署

Batch Jobs

ECS task scheduler

Run tasks once

Batch jobs

RunTask (random)

StartTask (placed)

Long-Running Apps

ECS service scheduler

Health management

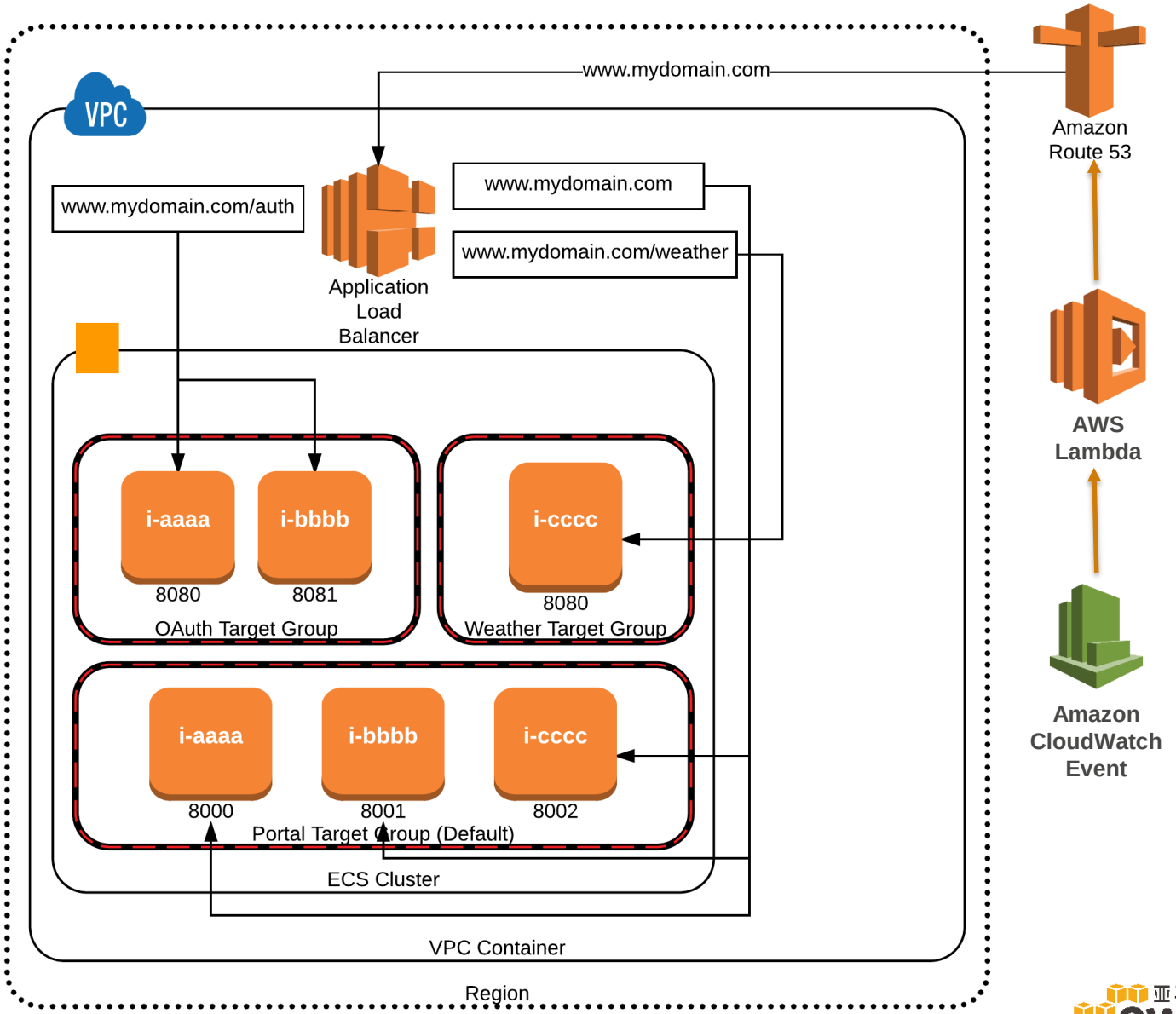
Scale-up and scale-down

AZ aware

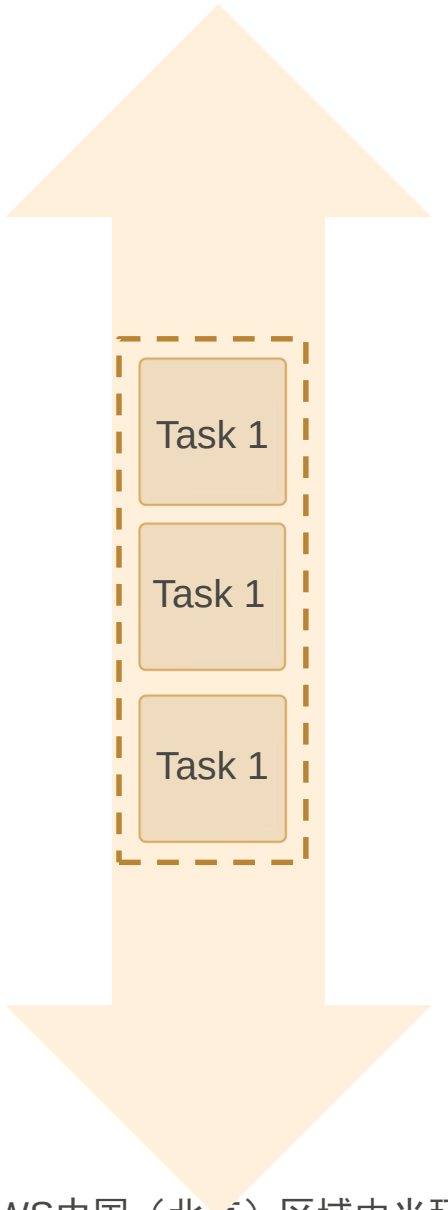
Grouped containers

2. 服务注册/发现

1. ECS创建服务
2. 事件流被CloudWatch捕获
3. CloudWatch过滤，触发Lambda
4. Lambda在Route 53中创建DNS记录
5. 服务通过Route 53注册/发现



3. 服务扩展



基于服务（容器）自动扩展

基于宿主机自动扩展

3. 服务滚动更新 – 最小容量更新

Deploy using the least space: *minimumHealthyPercent* = 50%, *maximumPercent* = 100%

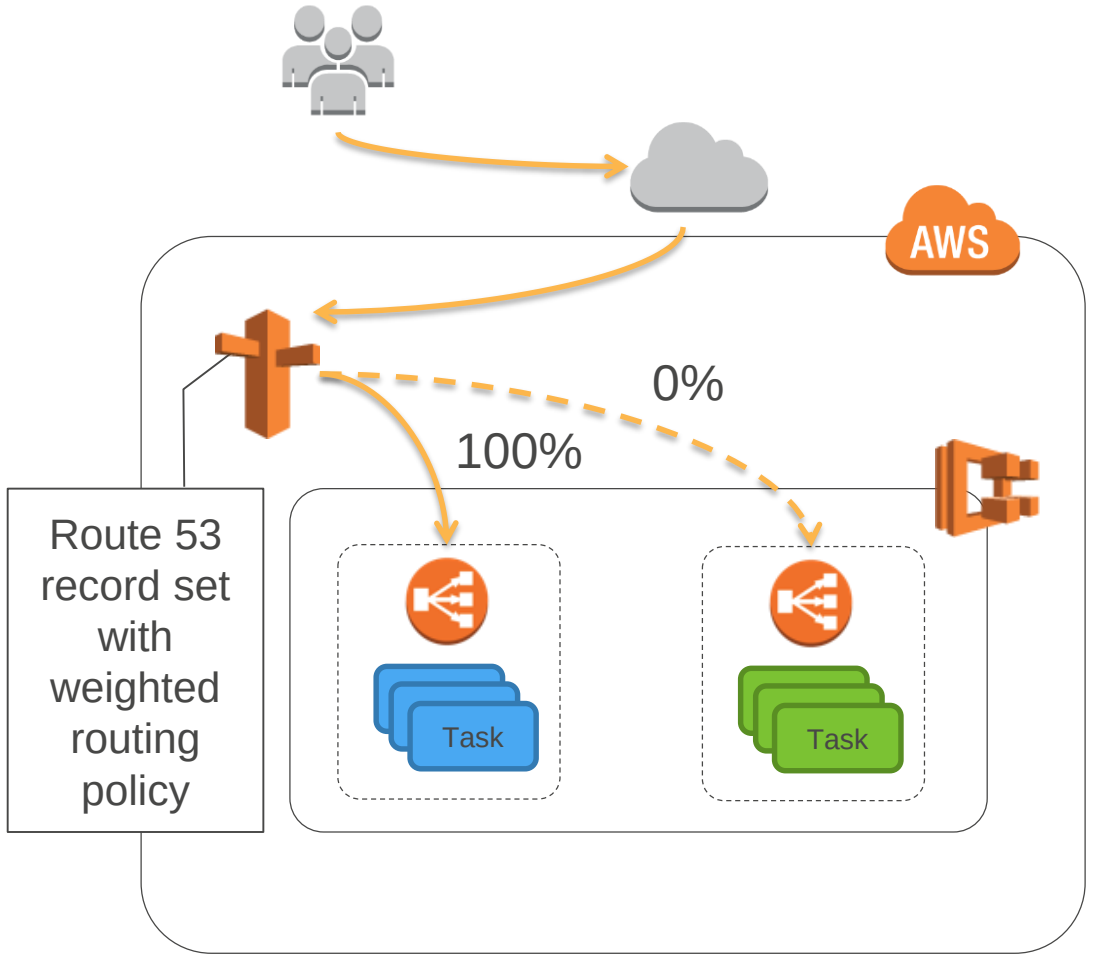
3. 服务滚动更新 – 同等规模更新

Deploy quickly without reducing service capacity: *minimumHealthyPercent* = 100%, *maximumPercent* = 200%

3. 服务更新 – 蓝绿部署

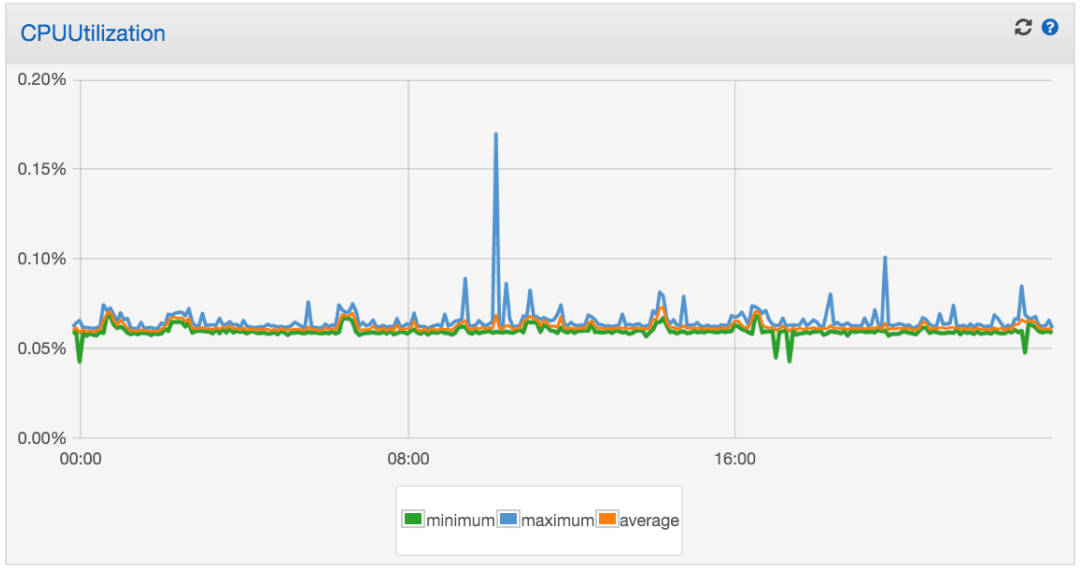
Blue-Green Deployments

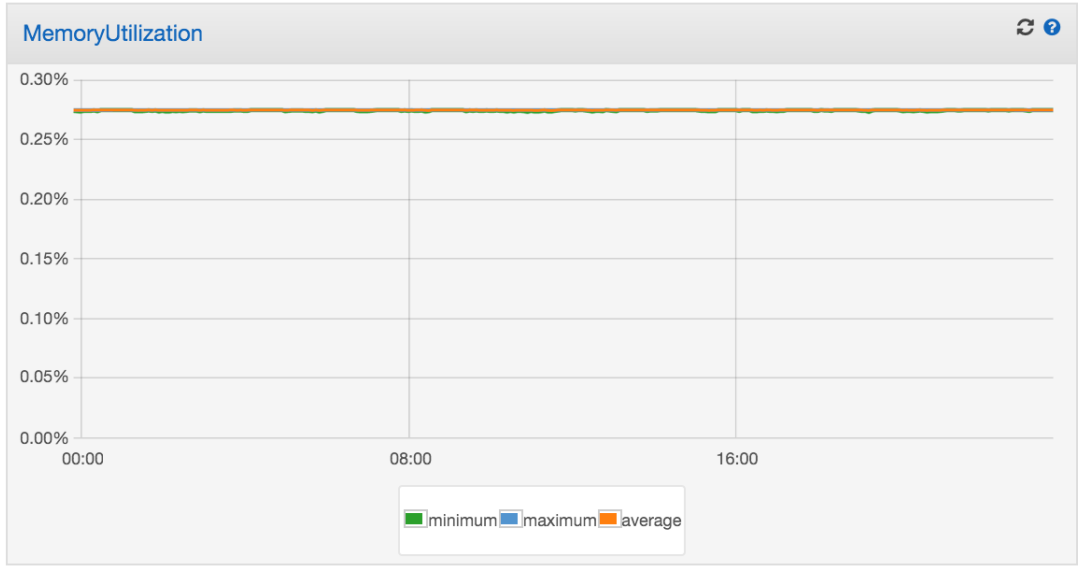
- Define two ECS services
- Each service is associated w/ load balancer
- Both load balancers in Route 53 record set with weighted routing policy, 100% primary, 0% secondary
- Deploy to blue or green service and switch weights

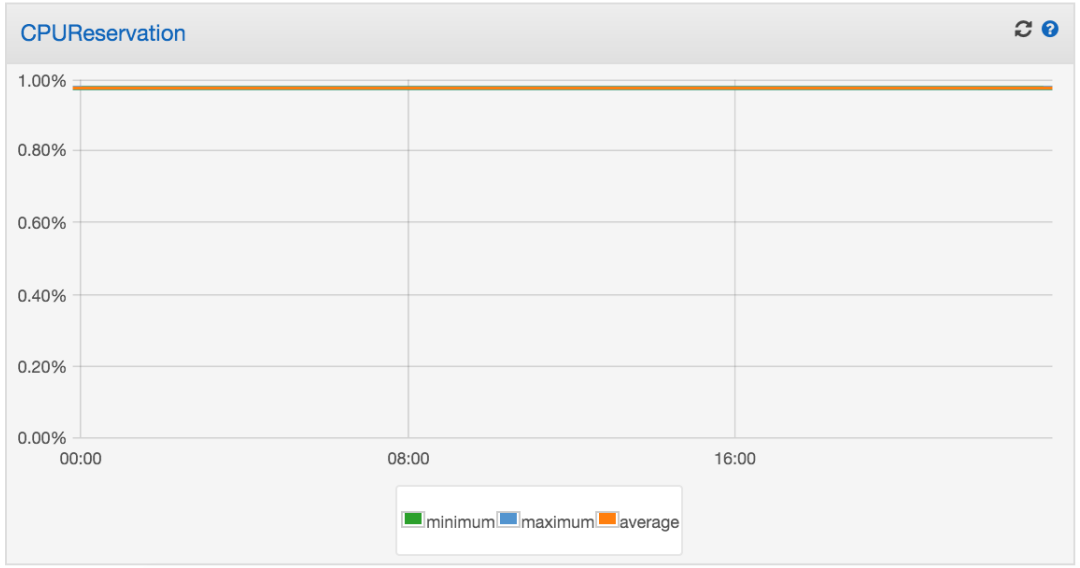


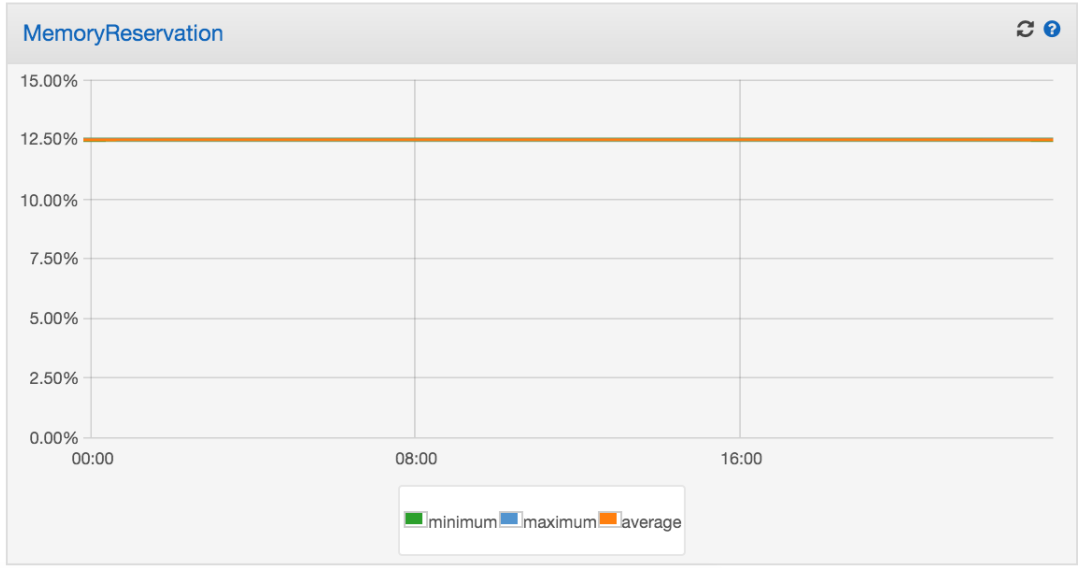
4. 日志与监控 - CloudWatch

Services | Tasks | ECS Instances | Metrics

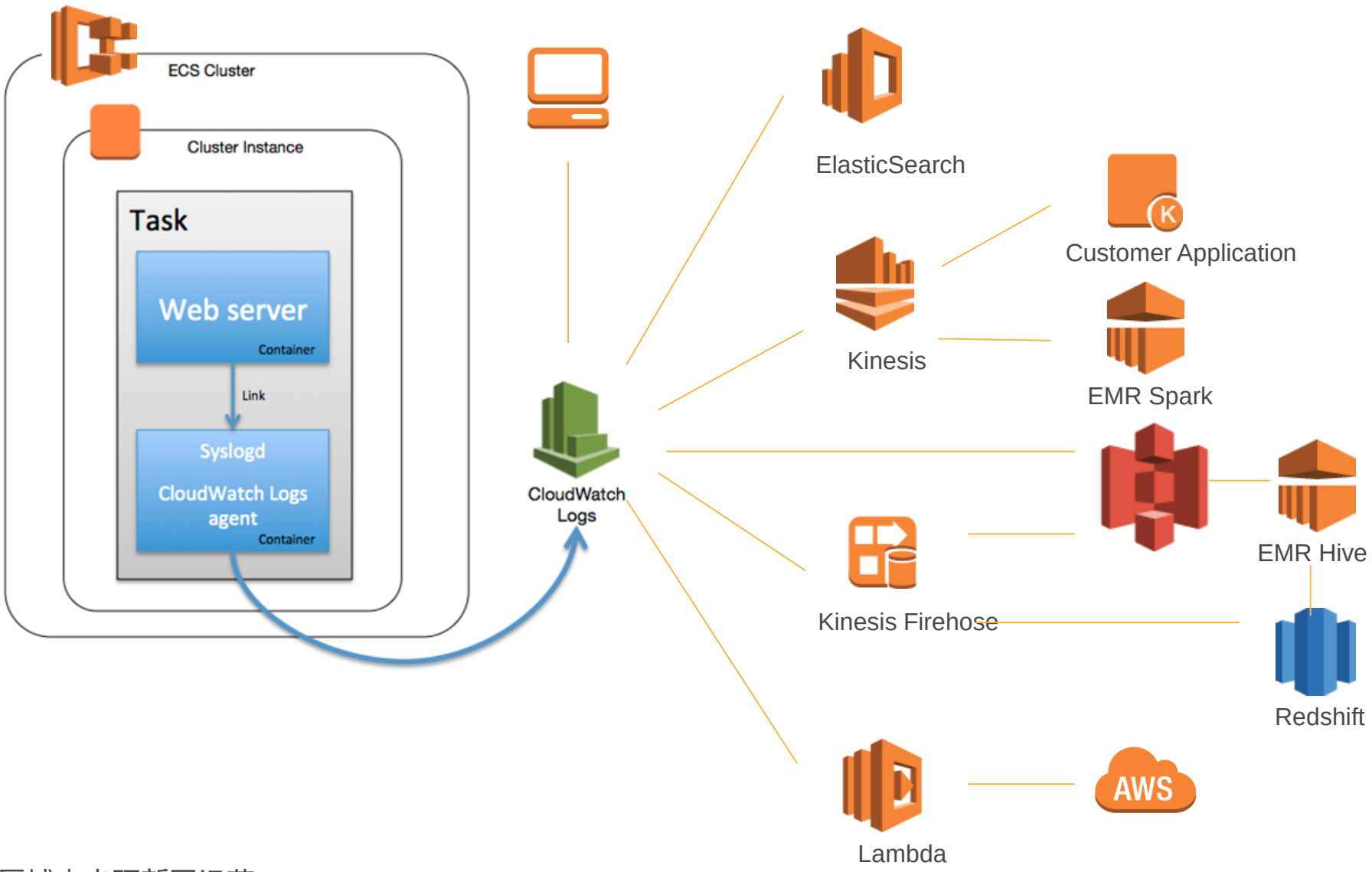








4. 日志与监控 - 容器日志记录与分析

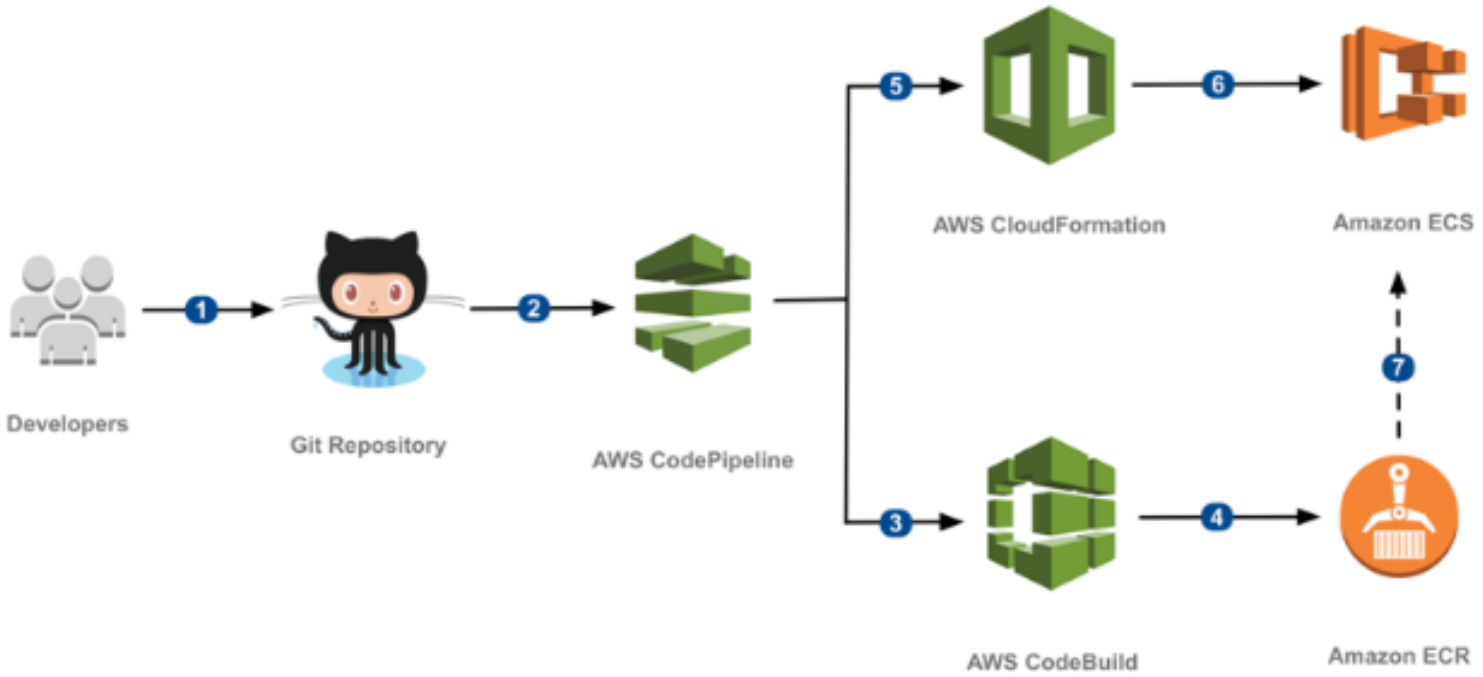


5. 权限与管理

用于Task的IAM 角色

用于宿主机的IAM 角色

参考架构 - 持续部署



1. Developers continually integrate their changes together into a main branch hosted within a source code repository system such as GitHub.
2. AWS CodePipeline polls the source code repository and triggers an execution of the continuously delivery pipeline when a new revision is found.
3. AWS CodePipeline sends the new revision to AWS CodeBuild which builds a Docker container image from the source code.
4. AWS CodeBuild pushes the newly built Docker container image tagged with the build ID to an Amazon ECR repository.
5. AWS CodePipeline initiates an update of the AWS CloudFormation stack which defines the Amazon ECS task definition and service.
6. AWS CloudFormation creates a new task definition revision referencing the newly built image and updates the Amazon ECS service.
7. Amazon ECS fetches the new container from Amazon ECR and replaces the old task with the new one which completes the deployment.

amazon web services

https://github.com/awslabs/ecs-refarch-continuous-deployment

实战问题1 – 镜像的制作和维护

VS

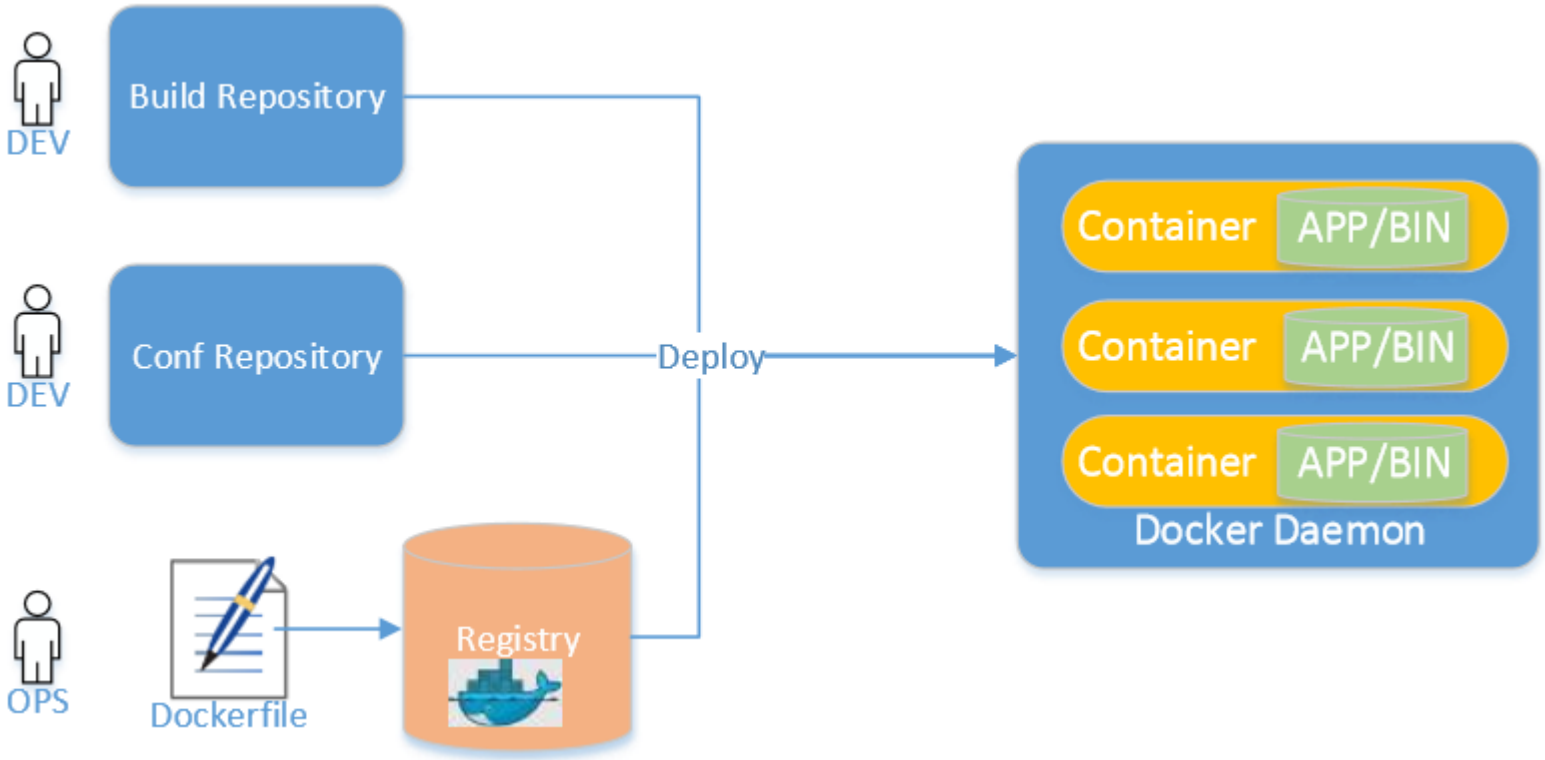
❑ **优点：**
- 应用和镜像合体，便于部署

❑ **缺点：**
- 频繁构建
- 镜像数量猛增
- 改造工作量大
- 不同环境配置文件不同

❑ **优点：**
- 镜像不用重构，配置即时生效

❑ **缺点：**
- 所有宿主机维护应用副本
- 违背了Docker集装箱原则

解决方案 – 镜像的制作和维护



- 开发时：镜像和应用分离，OPS维护Dockerfile，DEV维护代码
- 部署时：镜像和应用合体
- 多环境不同配置，分布式配置集中管理+配置文件中心

实战问题2 – 日志及排障

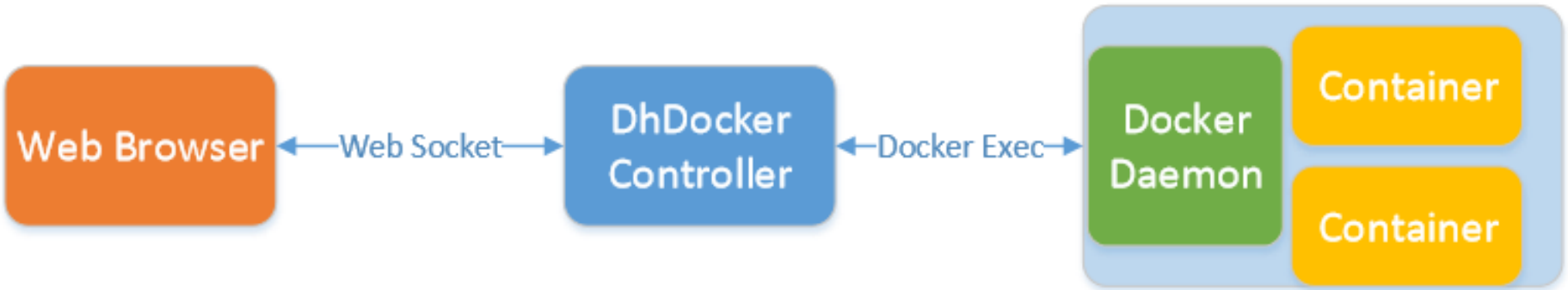
- **集群（宿主机）日志收集**
  - Cloud Watch Logs代理
    - /var/log/dmesg
    - /var/log/messages
    - /var/log/docker
    - /var/log/ecs/ecs-init.log
    - /var/log/ecs/ecs-agent.log

- **应用日志收集**
  - 宿主机磁盘卷
  - 一对一agent
  - 一对多agent

实战问题2 – 日志及排障

- 实时查看日志/状态，安全控制
  - SSH ?

别把容器当虚机使用 ！



实战问题3 – 统计数据收集

- ❑ 容器级别统计数据
- ❑ 应用级别统计数据

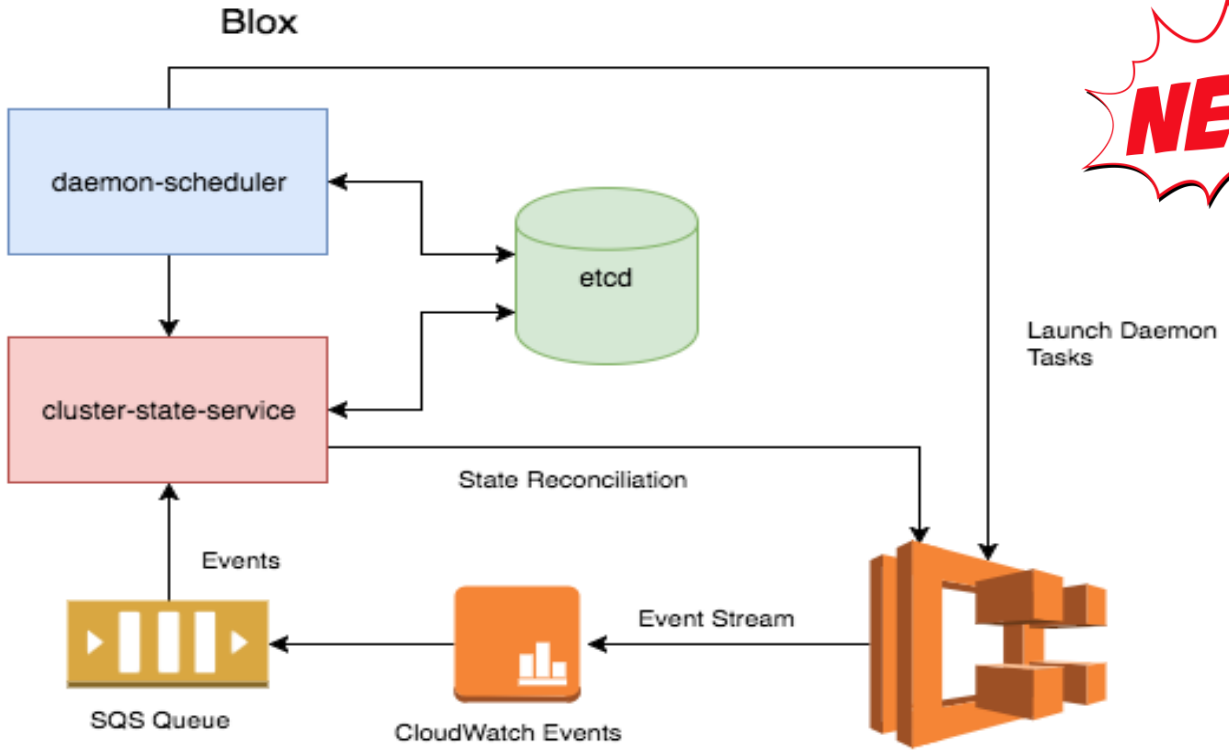
自定义指标

CloudWatch Agent

```
aws cloudwatch put-metric-data --metric-name PageViewCount --namespace
"MyService" --value 2 --timestamp 2016-10-14T12:00:00.000Z
```

- ❑ 主机级别（容器数量，状态等）

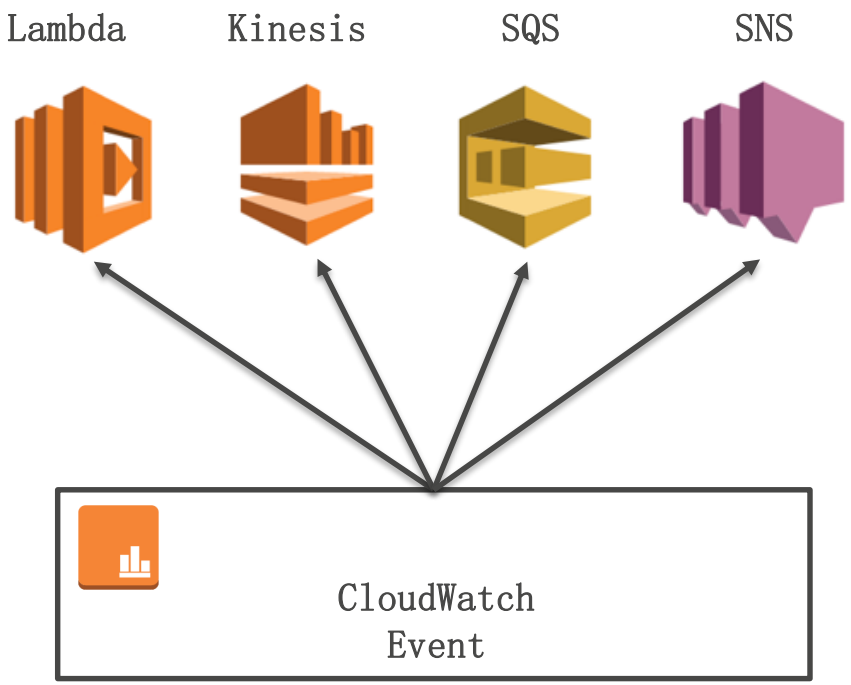
Blox

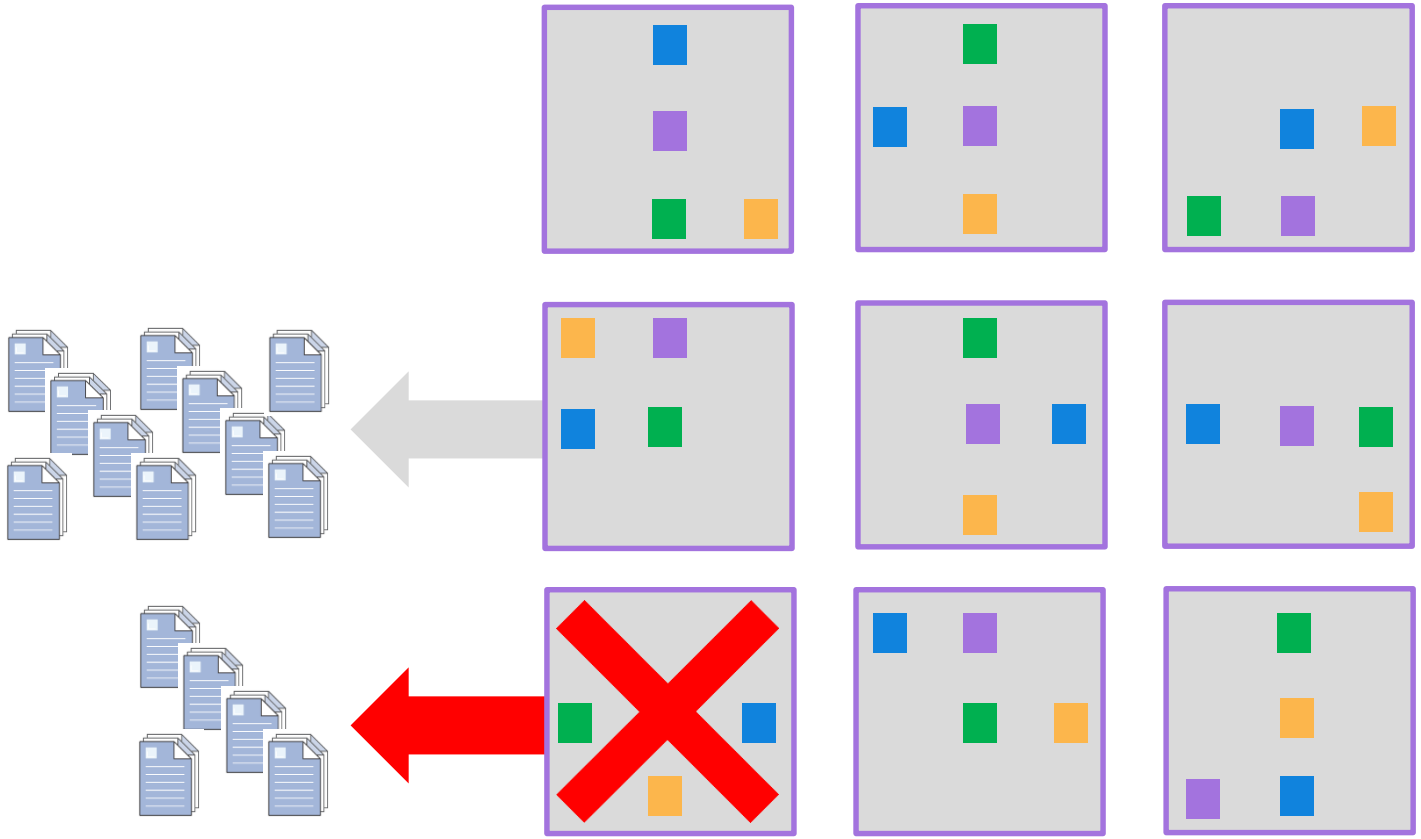


NEW!

实战问题4 – 自动化运维与审计

事件流





客户案例

Instacart ~ 国内“多点”

创立于2012

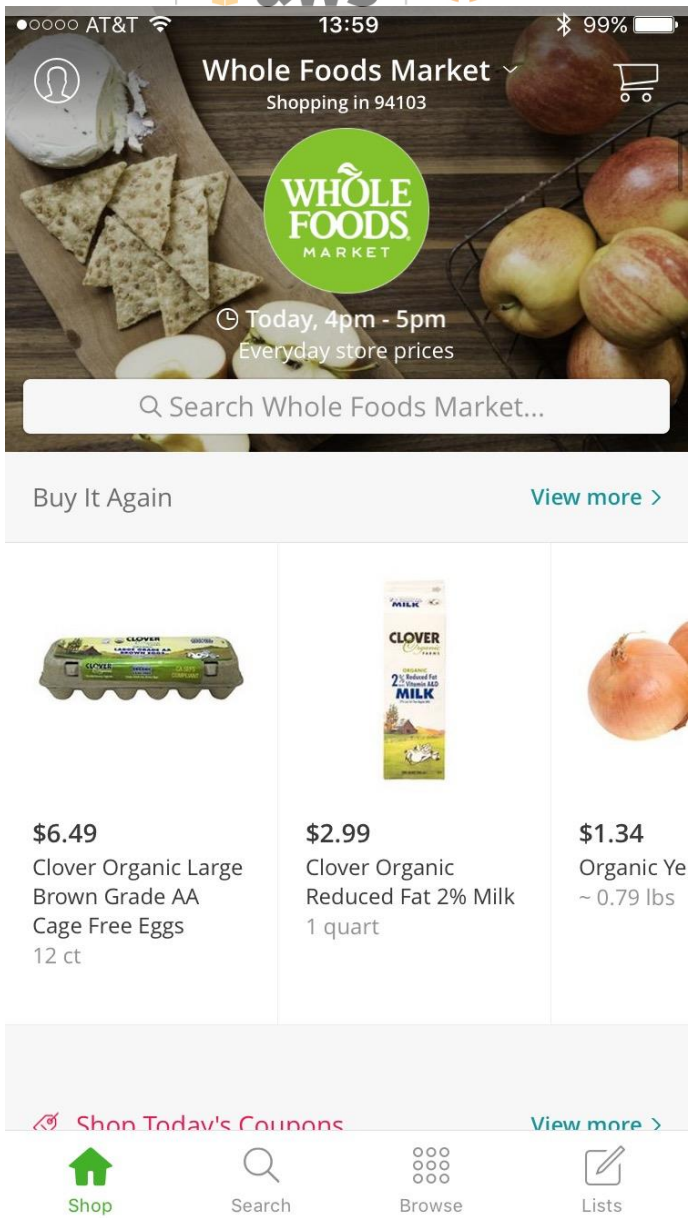
1小时到达的本地商店采购服务

135家食品零售商

28亿￥融资

235亿￥估值

很小的技术和运维团队



技术栈

- 主要采用 Ruby/Rails/JS, Python, & R
- 子域名划分, 各自拥有自己的 SLA
- 子域名又由各自的微服务组成
- Monorepo交付模型

为什么不用EC2?

- 启动相对较慢
- 部署和回滚不具备原子性
- 资源的灵活度受限于EC2所定义的实例大小
- 对开发者透明: 容器 vs. 实例

部署

- 原子性的部署通过CloudFormation
- 瞬间回滚
- 按照不同的应用类型来自定义“GC”策略
- 更符合需要部署的应用程序的规格
- 更加迅速快捷的CI/CD

蓝绿部署

- 利用ALB的注册/分离来完成蓝/绿
- 只需要保持一个或多个旧版的运行就可以立即回滚
- 利用不同的端口部署蓝

谢谢！