
AWS Summit
AWS技术峰会2015 · 上海
亚马逊
aws

蓝/绿部署在AWS上的实践模式

武杰 解决方案架构师 AWS

本话题讨论的主要内容

- 系统部署的风险分析
- 蓝/绿部署的重要概念
- 在AWS上进行蓝/绿部署的益处
- 在AWS上进行蓝/绿部署模式
- 数据层面切换的最佳实践
- 成本优化

系统部署的风险

- 面临的挑战
- 应用程序出错
- 基础架构失效
- 容量的问题
- 扩展的问题
- 人为的错误
- 流程失效
- 回滚的问题

- 对业务的影响
- 宕机时间
- 数据丢失
- 糟糕的客户体验
- 损失收入
- 业务人员的抱怨
- 员工压力太大
- 浪费时间和资源

在AWS上定义蓝/绿部署

什么是蓝/绿部署？

- “蓝”
- *(existing production environment)*

- “绿”
- *(parallel environment running a different version of the application)*

- “部署”
- *(ability to switch traffic between the two environments)*

什么是环境？

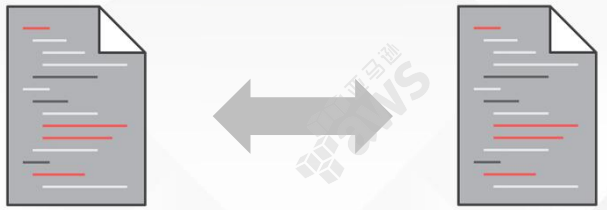
Boundary for where things changed and what needs to be deployed

示例:

App component, app tier, microservice

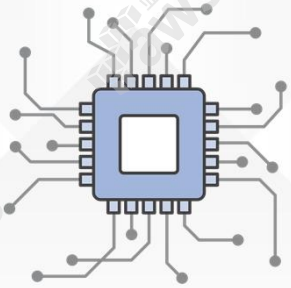
示例:

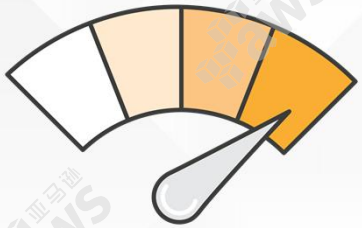
DNS, load balancer



在AWS上进行蓝/绿部署的益处

- **AWS:**
- 弹性部署
- 多样的选择
- 容量可动态调整
- 按实际用量付费
- 高效快速





在AWS上进行蓝/绿部署模式

共有的问题: 环境的自动化

- 成功的部署取决于对如下风险的克服:

- 应用程序出问题 (功能性的问题)
- 应用程序的性能问题
- 人员/流程 失效
- 基础架构失效
- 回滚的容量
- 高额的成本

自动化平台的长处

CloudFormation

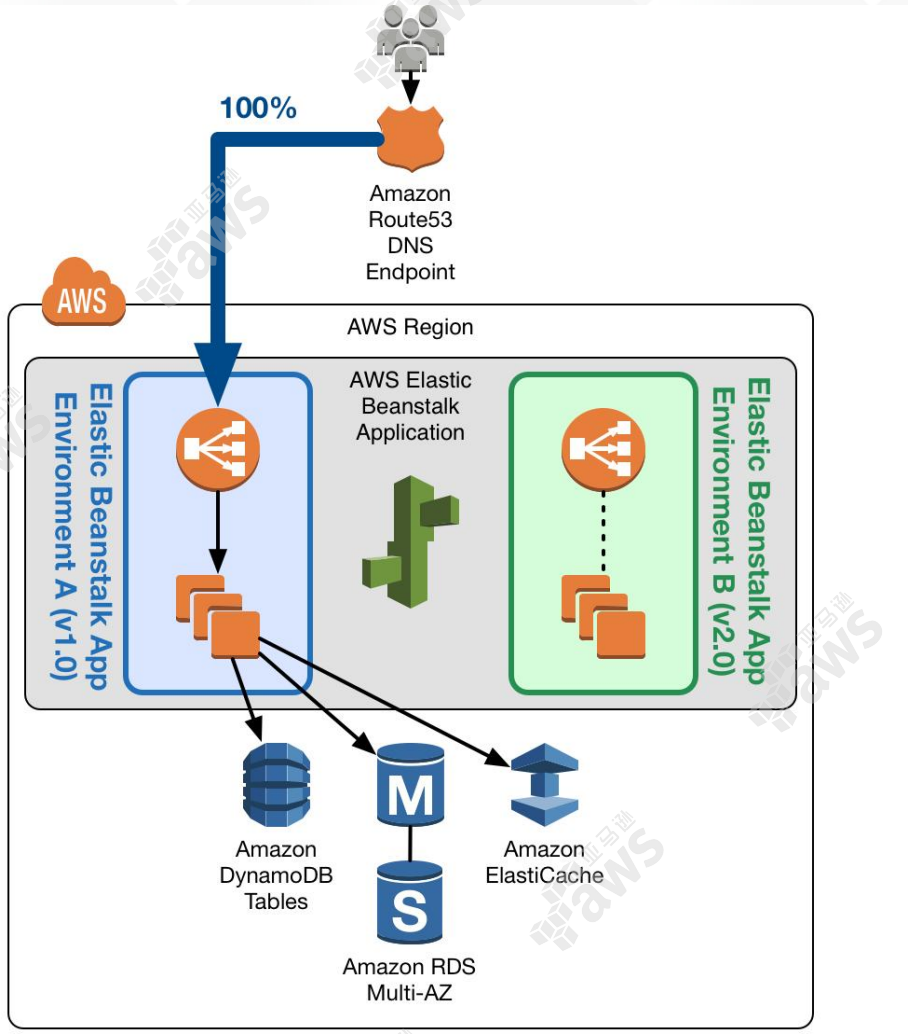
全面的自动化平台

- ✓ 定义从网络到软件的整个环境
- ✓ 控制高一级的自动化服务: Elastic Beanstalk, OpsWorks, Auto Scaling

- **蓝/绿部署的不同模式对以上这些风险有不同的处理方式**

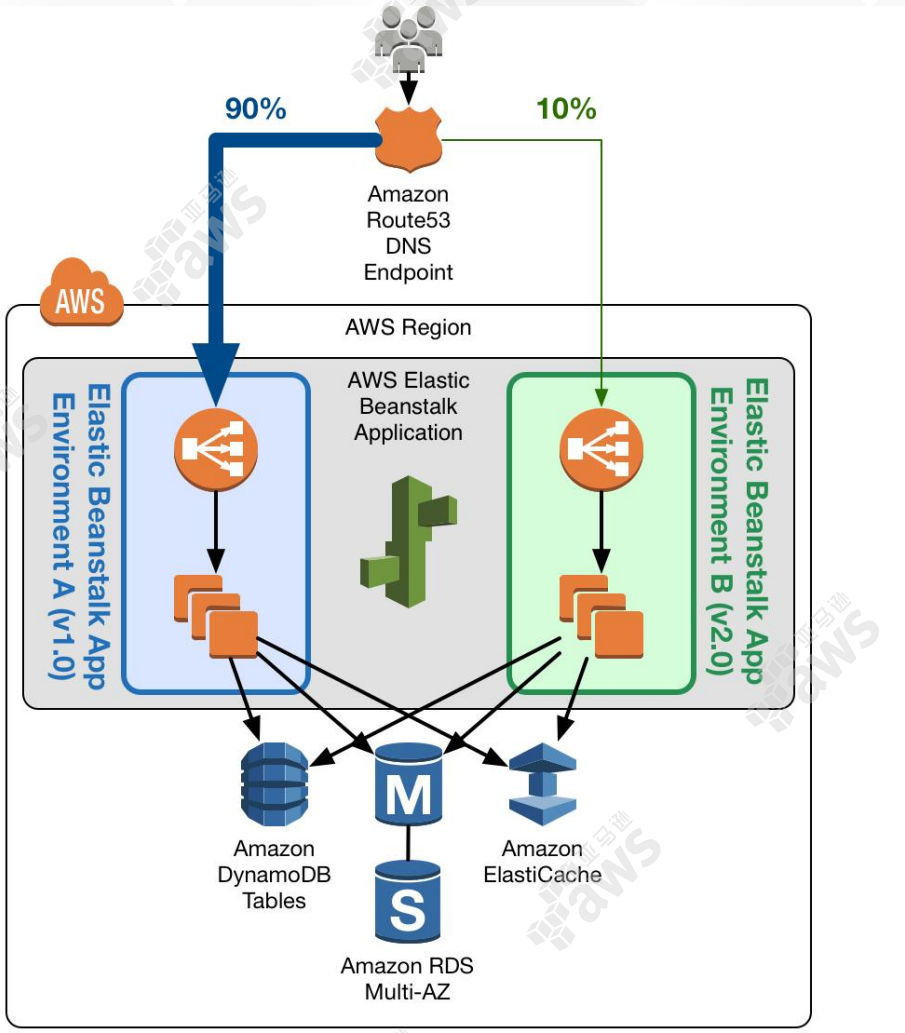
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



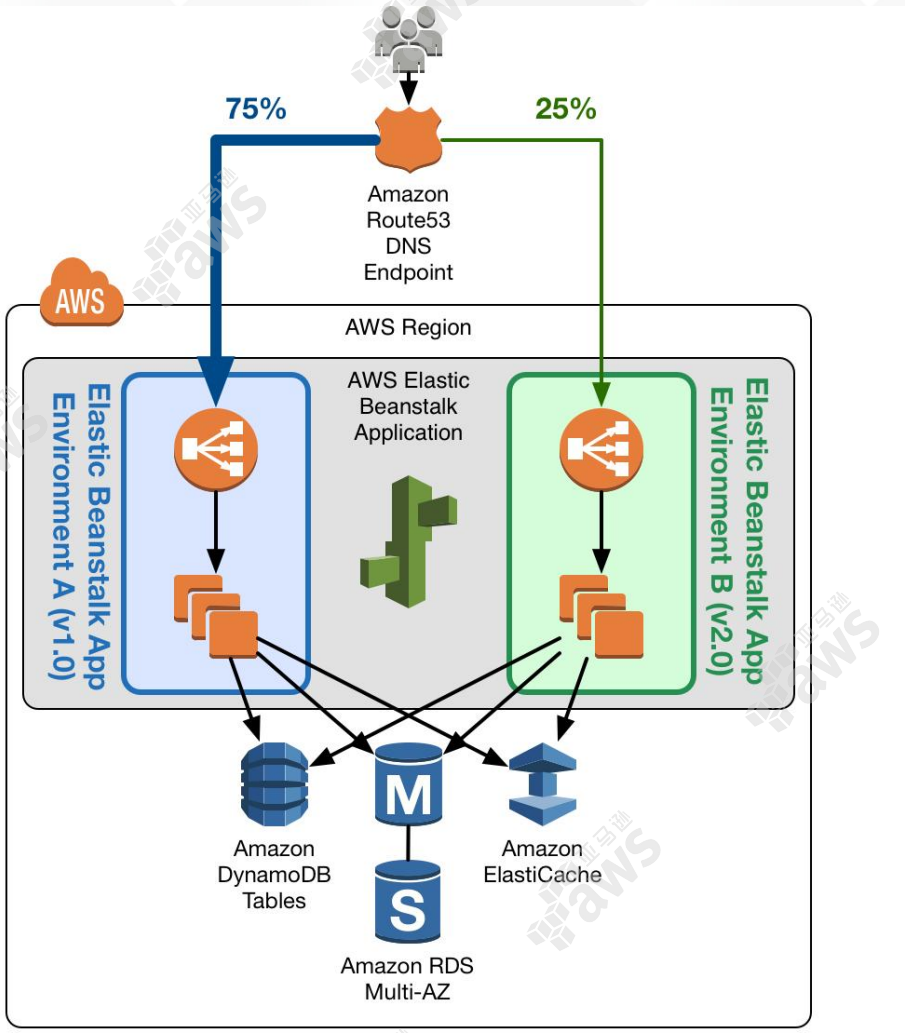
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



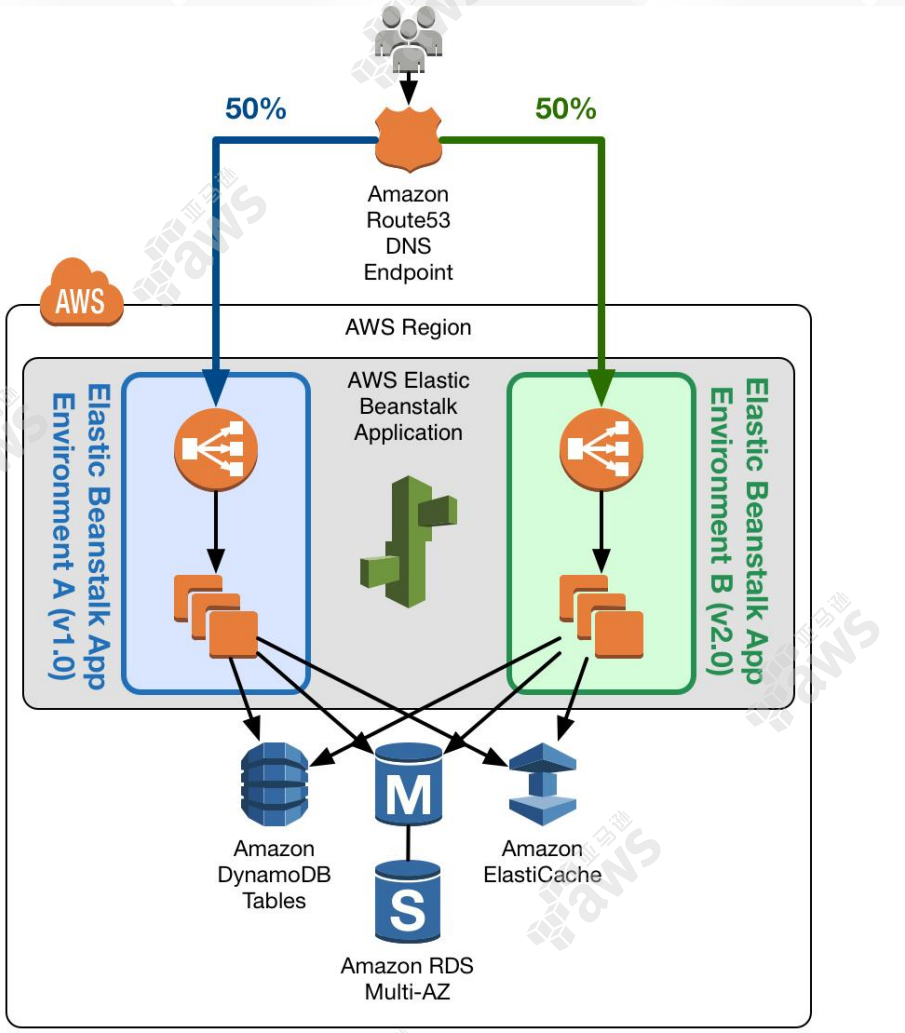
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



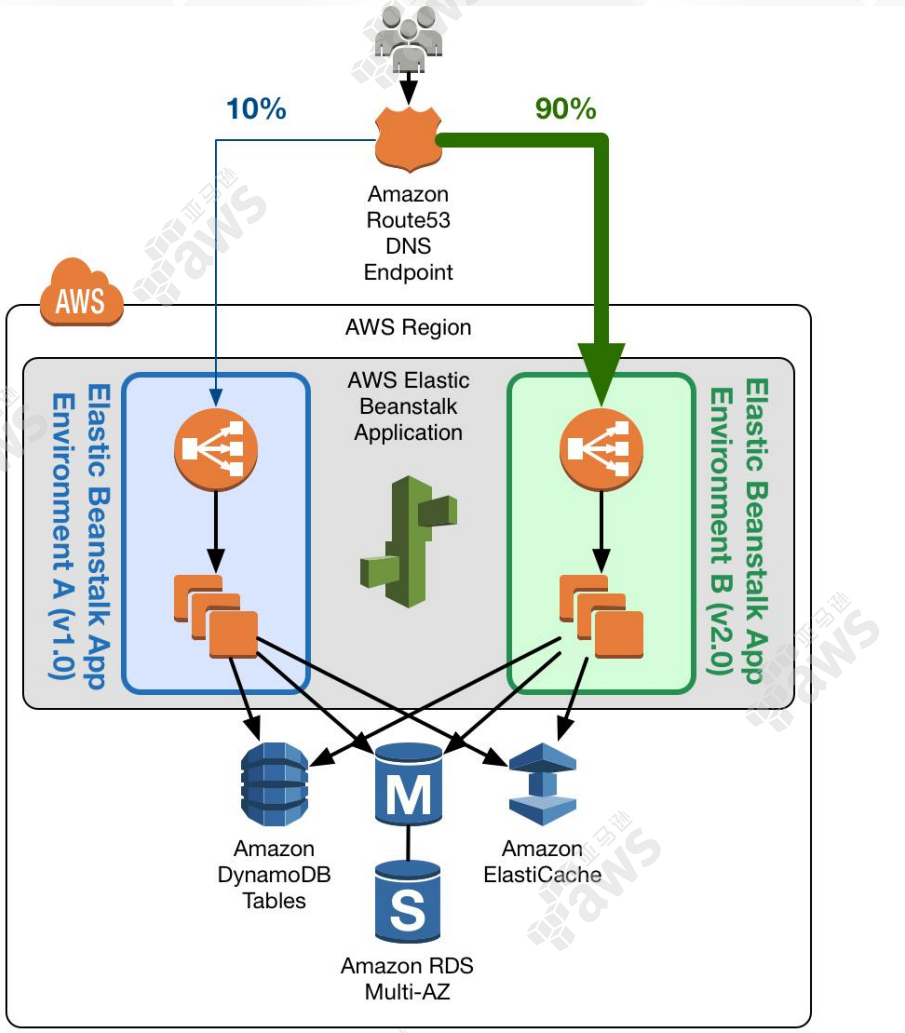
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



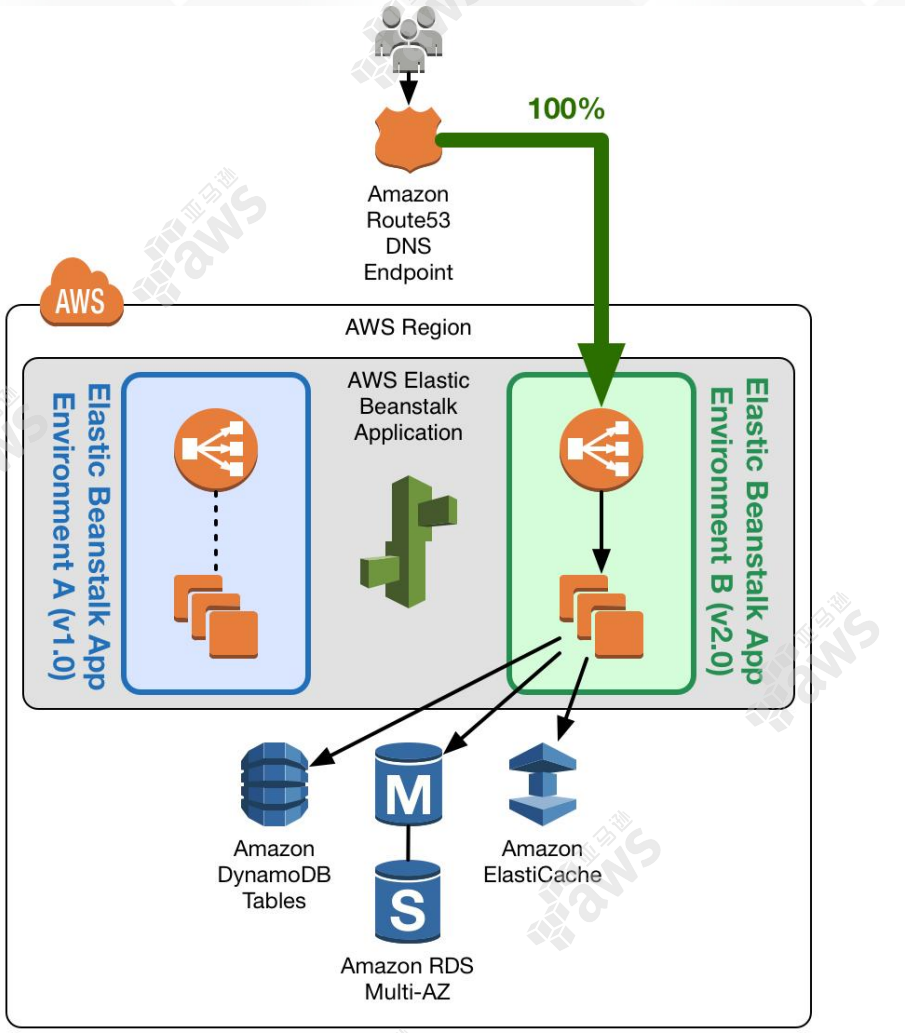
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



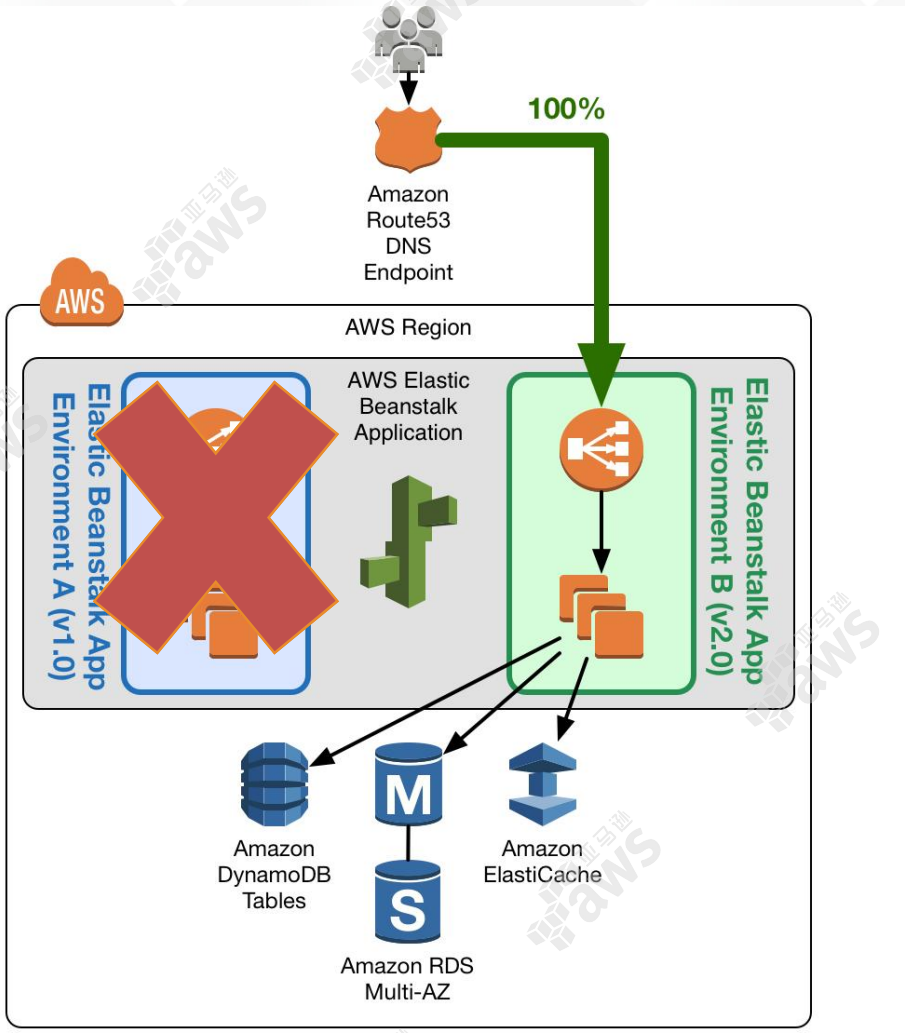
模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



模式: 经典的DNS切换

- 部署流程:
- 从已有的应用环境开始
- 部署新的应用环境
- 测试绿的应用环境
- 通过DNS逐步切换流量
- 监控你的环境
- 出现问题，回滚到蓝的应用环境



自动化你的应用环境

- Use **CloudFormation** templates to model your environment
- Version-control your templates
- Use Elastic Beanstalk or OpsWorks to model your applications inside the template
- Update CloudFormation stack from updated template containing green environment

```
"Resources": {
  "myApp": { "Type": "AWS::ElasticBeanstalk::Application"
},
  "myConfigTemplate": {
    "Type":
"AWS::ElasticBeanstalk::ConfigurationTemplate"
  },
  "myBlueAppVersion": {
    "Type": "AWS::ElasticBeanstalk::ApplicationVersion"
  },
  "myBlueEnvironment": {
    "Type":"AWS::ElasticBeanstalk::Environment"
  },
  "myGreenAppVersion": {
    "Type": "AWS::ElasticBeanstalk::ApplicationVersion"
  },
  "myGreenEnvironment": {
    "Type": "AWS::ElasticBeanstalk::Environment"
  },
  "myGreenEndpoint": {"Type": "AWS::Route53::RecordSet" }
  ...
```

Amazon Route 53 基于权重的DNS切换

- AWS Elastic Beanstalk environment endpoint swap
- DNS record time-to-live (TTL)
- Reaction time = (TTL × no. of DNS caches) + Route53 propagation time, up to 1min
- Beware of misbehaving DNS clients
- Auto Scaling and Amazon Elastic Load Balancing (ELB) need time to scale
- Measurable metrics
- ELB: Latency, SurgeQueueLength, SpillOverCount, BackendConnectionErrors
- Your application metrics
- Your deployment goals

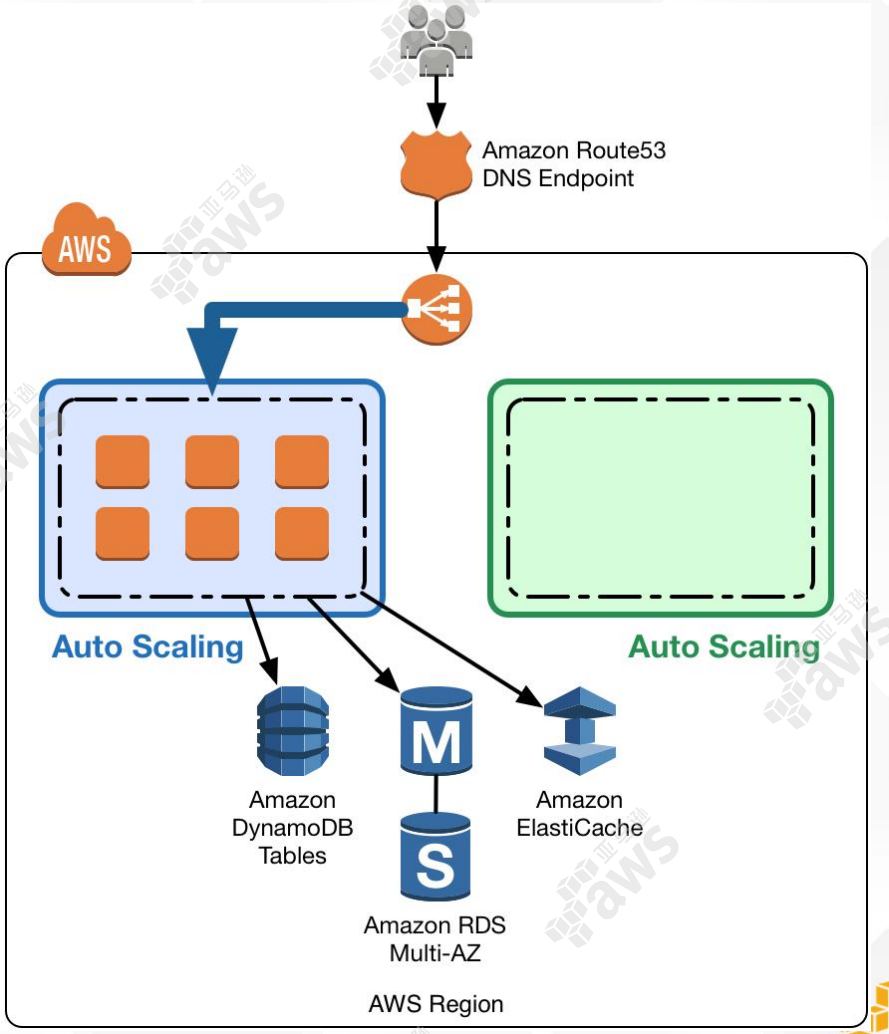
模式回顾:经典的DNS切换

风险	克服的程度	说明
程序的功能问题	优	有利于逐层剖析问题
程序的性能问题	优	平滑切换，流量分流管理
人员/流程出错	好	取决于自动化框架
基础架构失效	好	取决于自动化框架
回滚	中	DNS TTL 的复杂性 (reaction time, flip/flop)
成本	优	通过Auto Scaling来优化成本

让我们去掉DNS的切换...

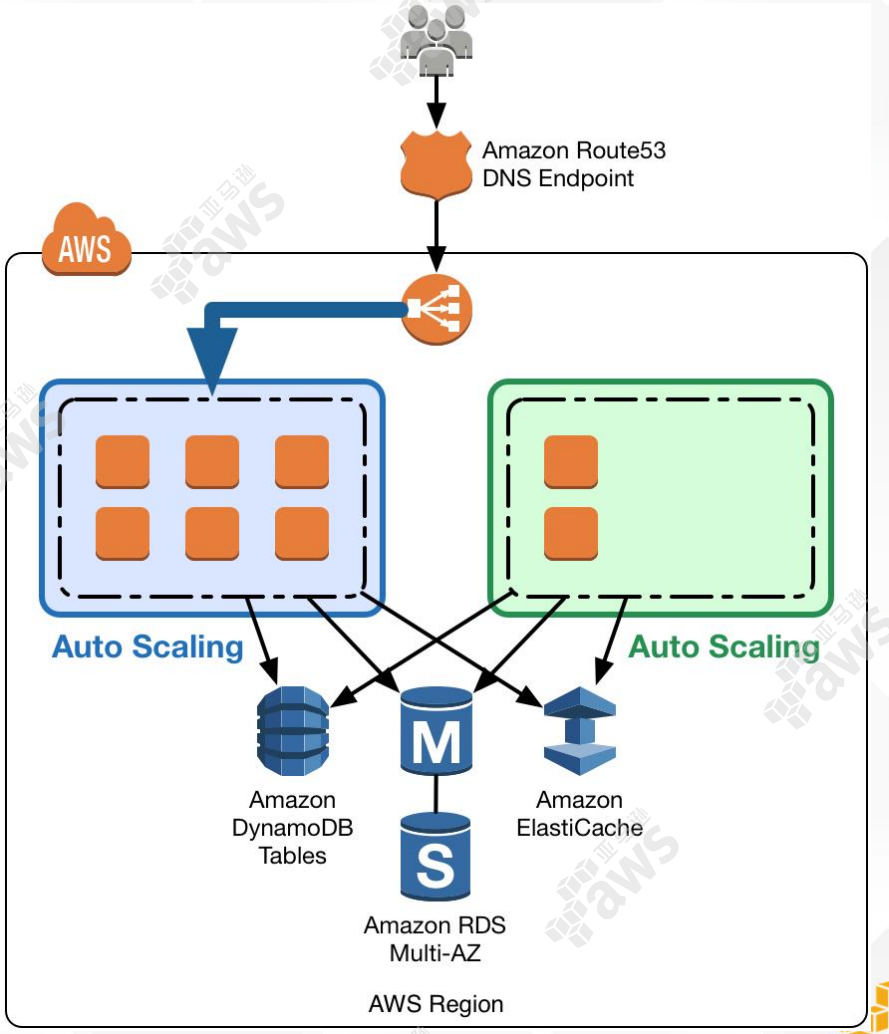
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB)，部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- 测试绿的应用环境
- 在ELB上注册绿的ASG
- 在ELB上去除蓝的ASG



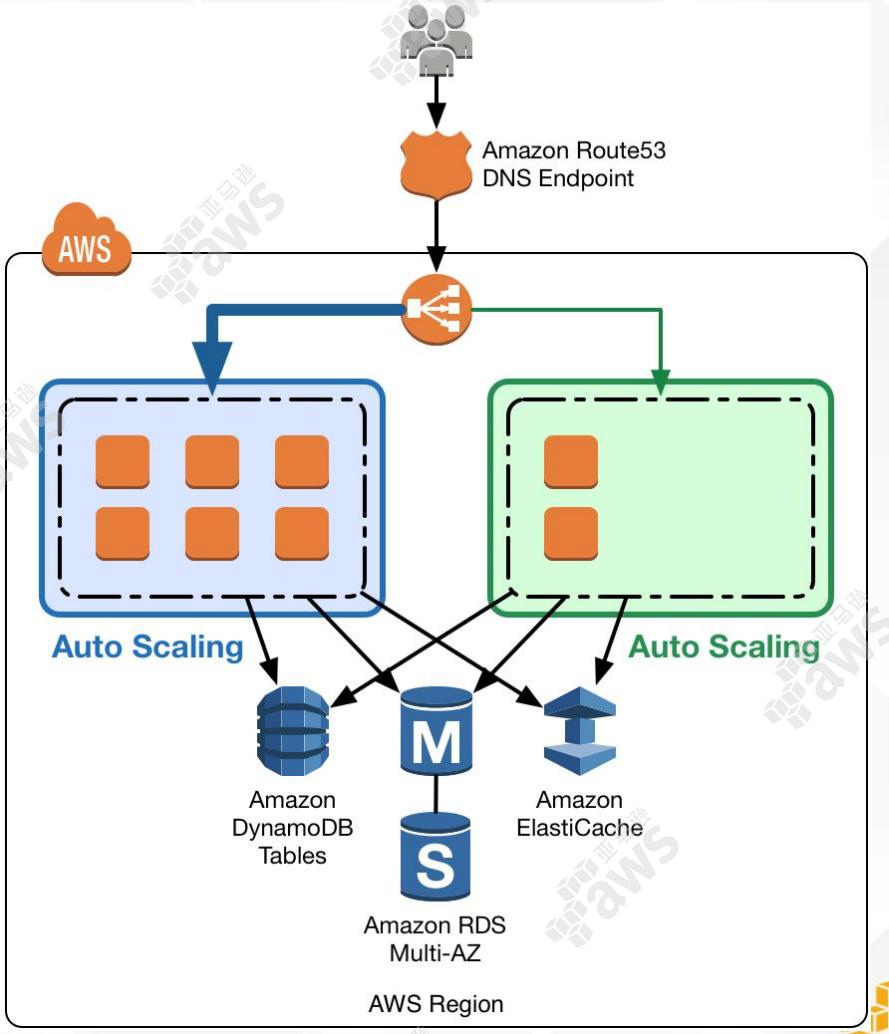
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- 测试绿的应用环境
- 在ELB上注册绿的ASG
- 在ELB上去除蓝的ASG



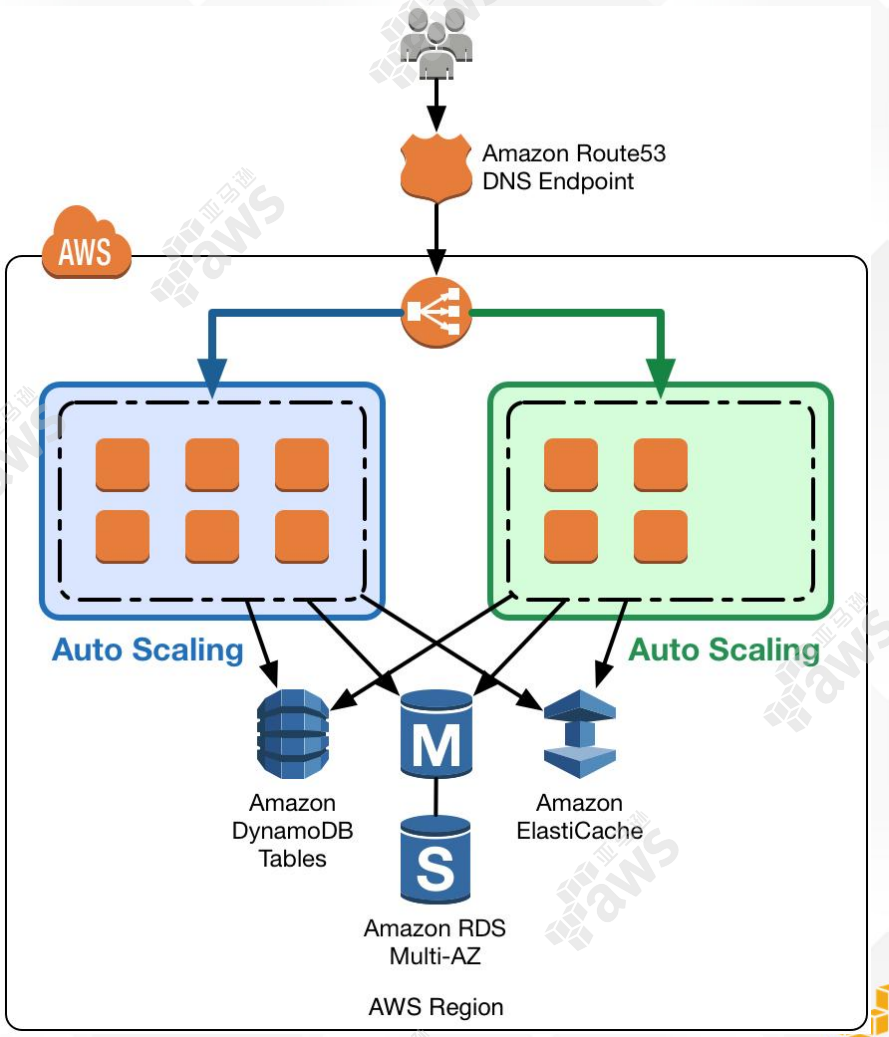
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- **测试绿的应用环境**
- **在ELB上注册绿的ASG**
- **在ELB上去除蓝的ASG**



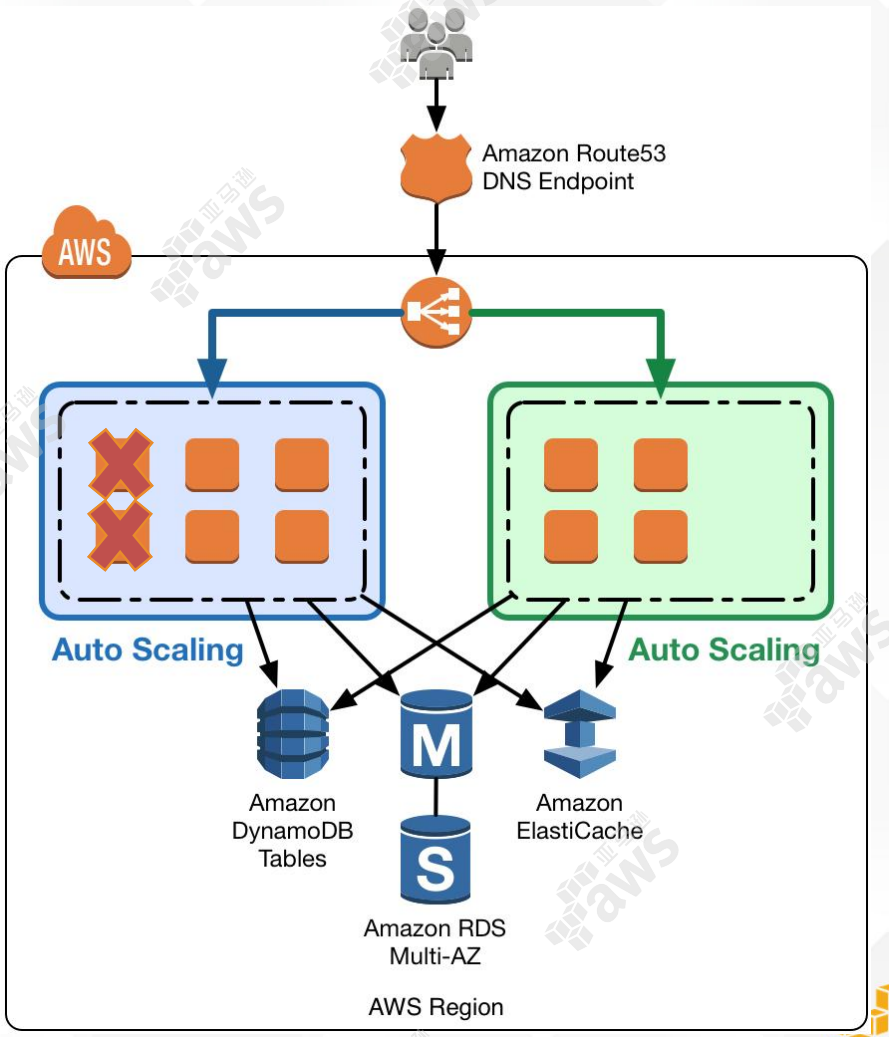
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- 测试绿的应用环境
- 在ELB上注册绿的ASG
- 在ELB上去除蓝的ASG



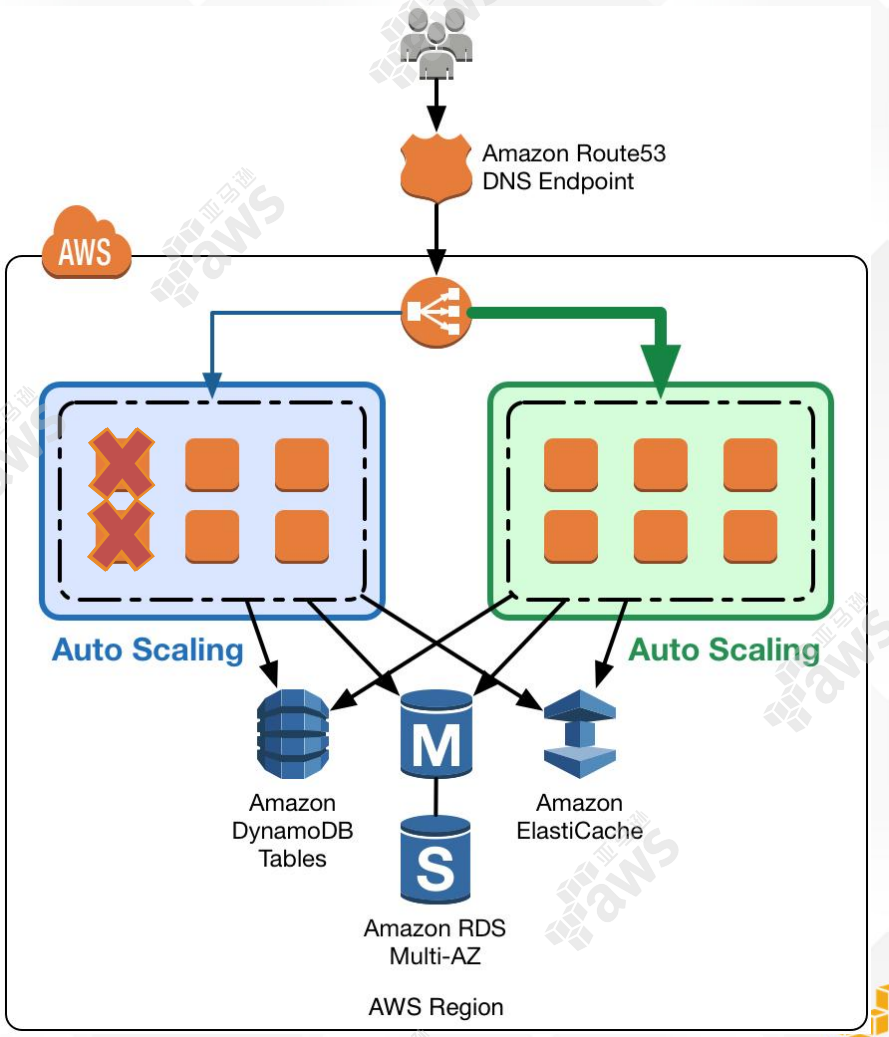
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- **测试绿的应用环境**
- **在ELB上注册绿的ASG**
- **在ELB上去除蓝的ASG**



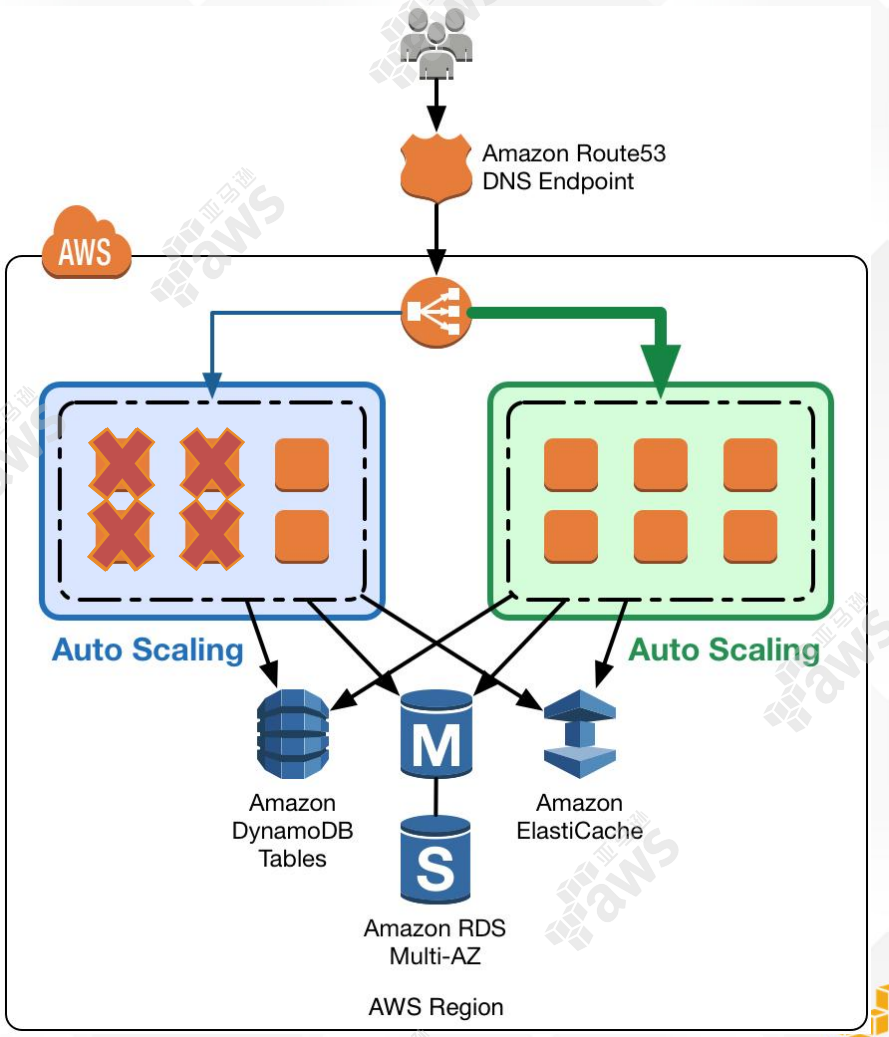
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- **测试绿的应用环境**
- **在ELB上注册绿的ASG**
- **在ELB上去除蓝的ASG**



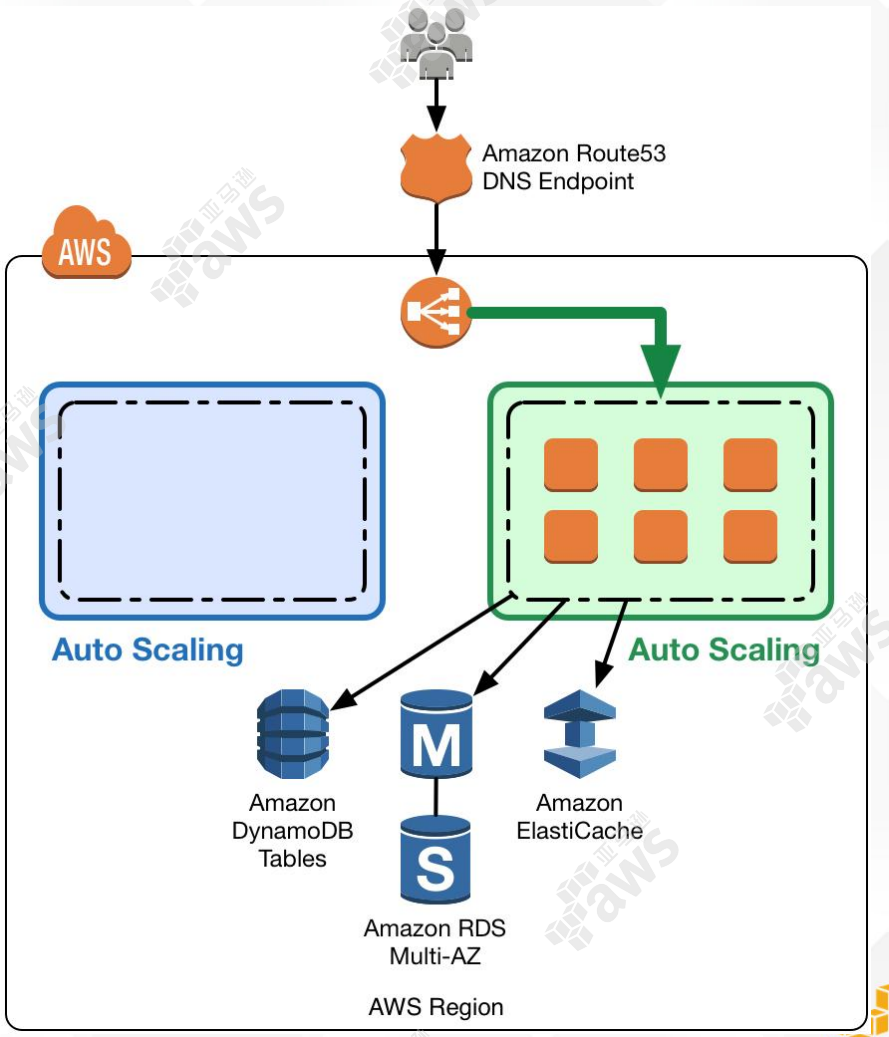
模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- **测试绿的应用环境**
- **在ELB上注册绿的ASG**
- **在ELB上去除蓝的ASG**



模式: 切换 Auto Scaling Groups

- 部署流程:
- Amazon Elastic Load Balancer (ELB) 部署在应用前段
- 从现有的 Auto Scaling Group (ASG)开始
- 部署&扩展新的ASG
- **测试绿的应用环境**
- **在ELB上注册绿的ASG**
- **在ELB上去除蓝的ASG**



在ELB后面切换 Auto Scaling groups

- Register with ELB:
  - One or more EC2 instances
  - One or more Auto Scaling groups
- Least outstanding requests algorithm favors green ASG instances for new connections
- Connection draining - gracefully stop receiving traffic
- Scale out green ASG before ELB registration
- Put blue instances in standby

```
$ aws autoscaling attach-load-balancers \
    --auto-scaling-group-name "green-asg" \
    --load-balancer-names "my-app-elb"

$ aws autoscaling set-desired-capacity \
    --auto-scaling-group-name "green-asg" \
    --desired-capacity X

$ aws autoscaling detach-load-balancers \
    --auto-scaling-group-name "blue-asg" \
    --load-balancer-names "my-app-elb"

$ aws autoscaling enter-standby \
    --instance-ids i-xxxxxxxx \
    --auto-scaling-group-name "blue-asg" \
    --should-decrement-desired-capacity
```

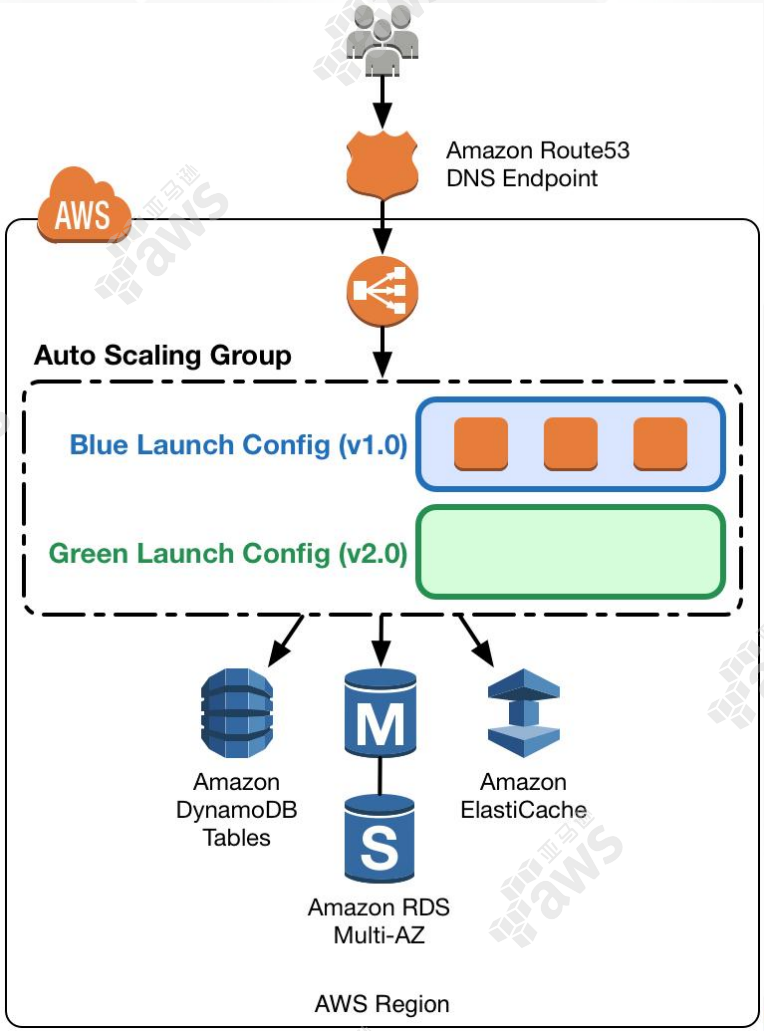
模式回顾: 切换 Auto Scaling groups

风险	克服的程度	说明
程序的功能问题	优	有利于逐层剖析问题，附加ELB
程序的性能问题	好	流量分流管理，力度较粗，ELB预热
人员/流程出错	好	取决于自动化框架
基础架构失效	优	Auto-scaling
回滚	优	没有DNS的复杂性
成本	优	通过Auto Scaling来优化成本

让我们继续减少应用环境的限制...

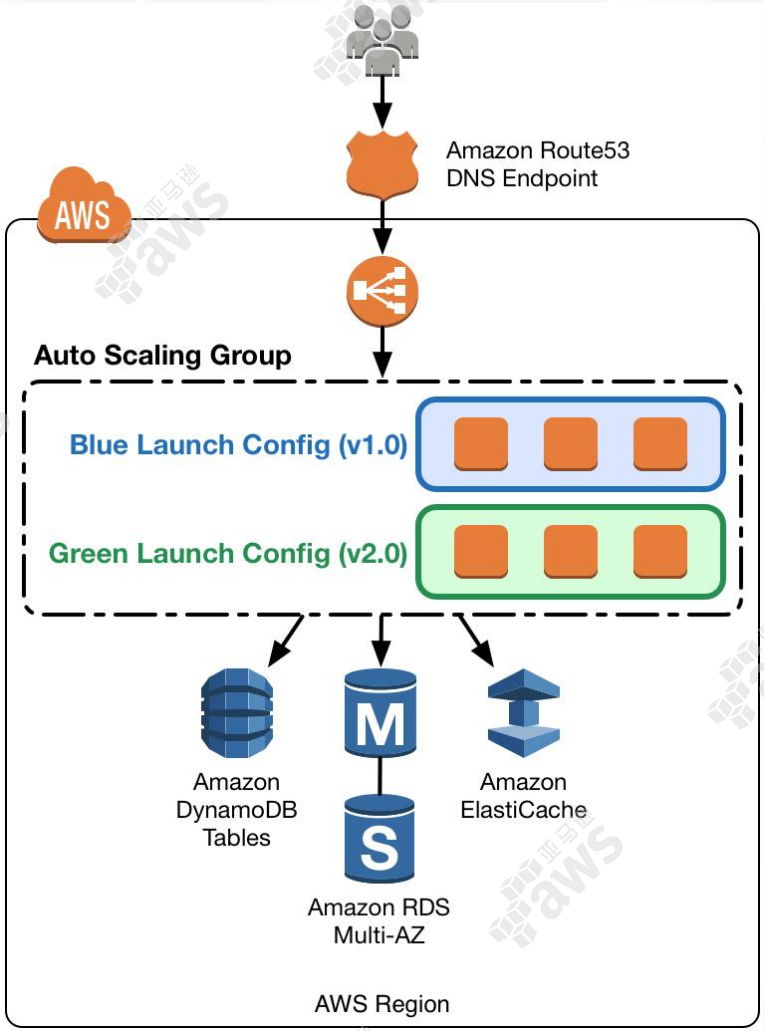
模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



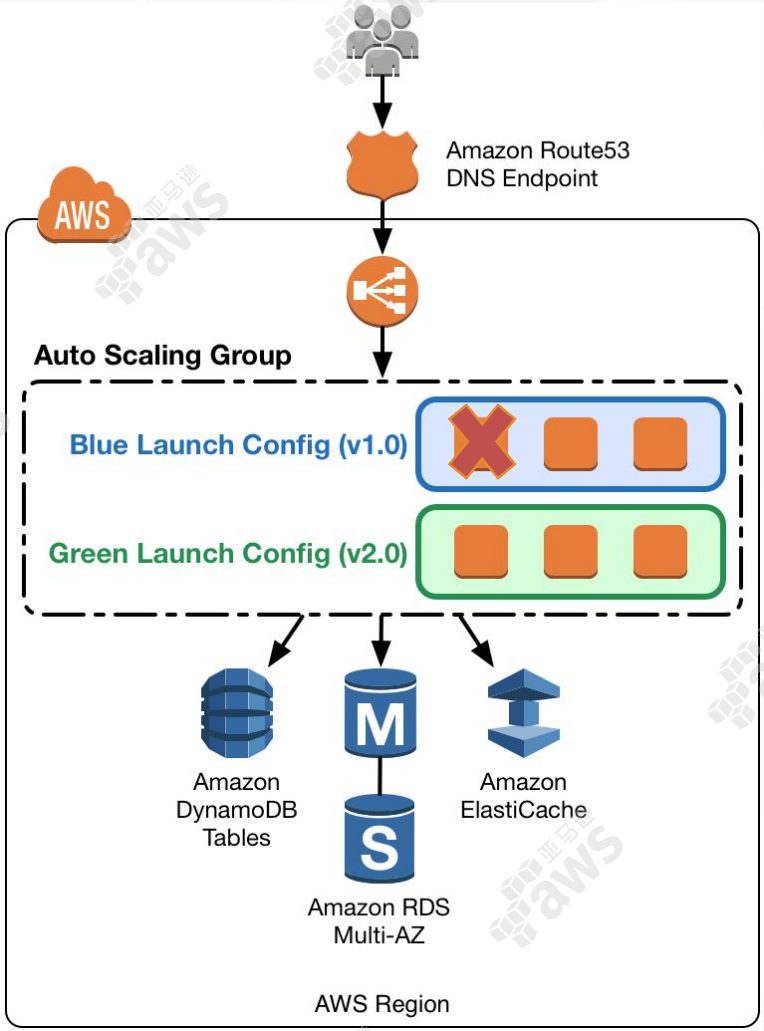
模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



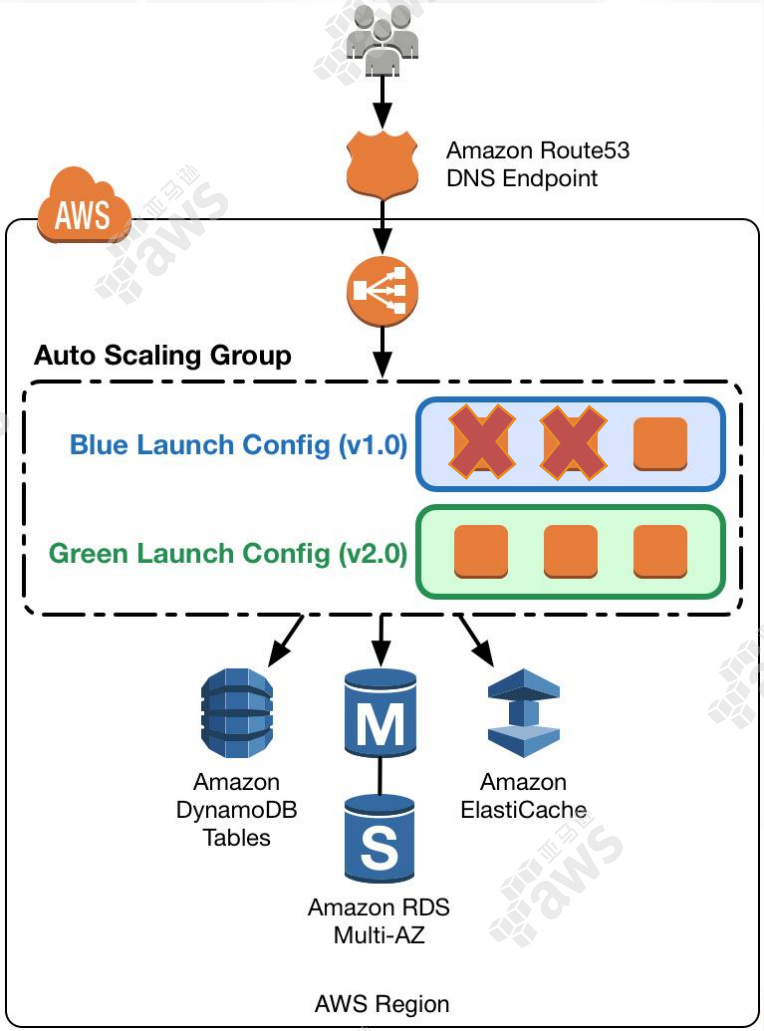
模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



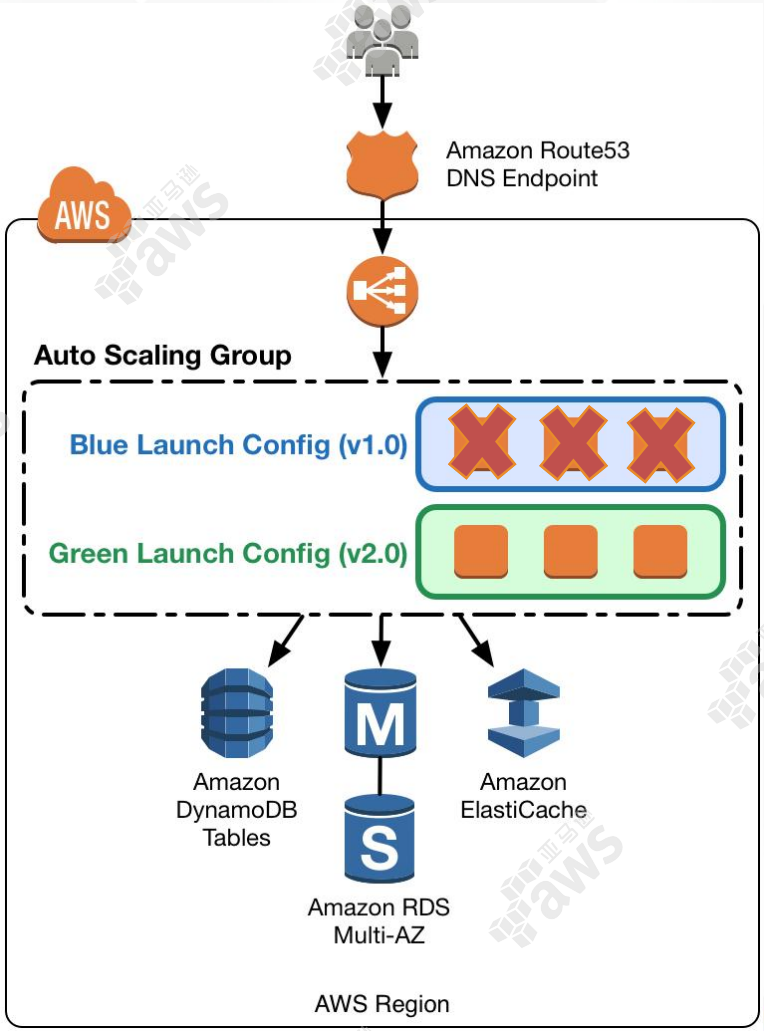
模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



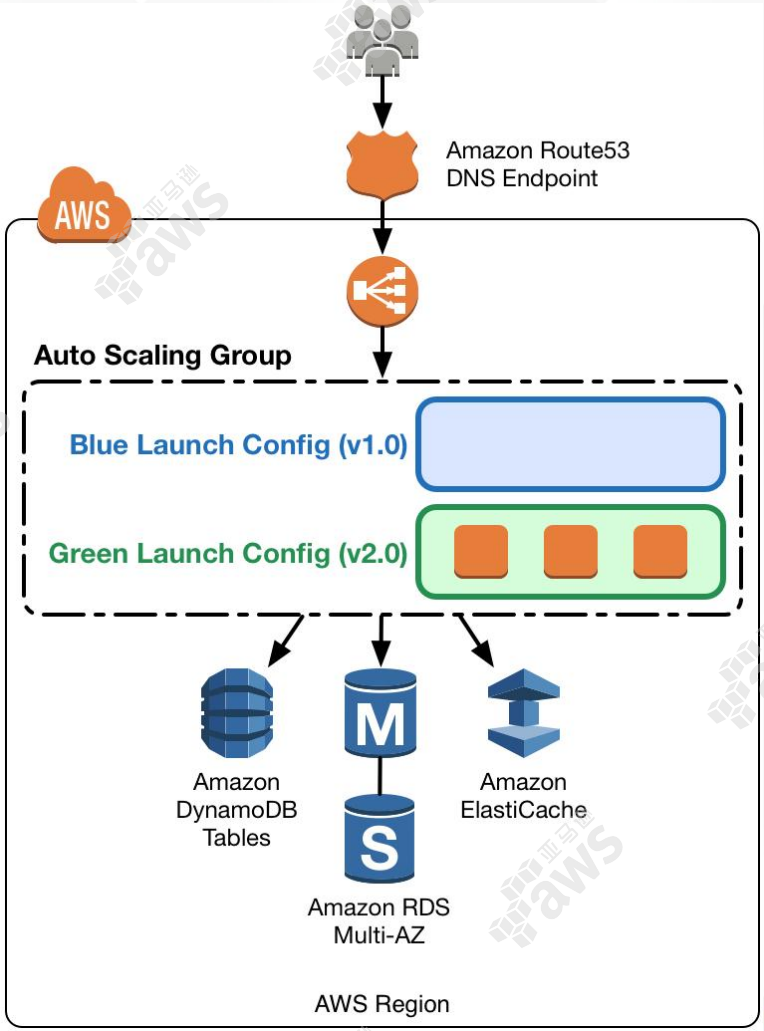
模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



模式: 切换 Launch Configurations

- 部署流程:
- 从现有的在ELB后面的 ASG & Launch Configuration 开始
- 在ASG上注册更新的绿的Launch Configuration
- 将ASG的容量逐步增加到原有容量的2倍
- 将ASG的容量减少到原有的容量
- 为增加可控性，将老的实例切换到备份状态



切换 launch configurations

- **Launch configurations:**
  Blueprints for ASG instance provisioning, each ASG points to exactly one
- **Scale-out & replacement:**
  Events will use the attached (**green**) launch configuration to provision instances
- **Scale-in:**
  ASG scale-in events will terminate instances with oldest launch configuration first **while** trying to keep capacity in AZs balanced
- May need to address AZ imbalances separately
- **Temporarily remove instances from ASG**
  Place specific ASG instances (**blue**) into standby – stop receiving traffic

模式回顾: 切换 Launch Configurations

风险	克服的程度	说明
程序的功能问题	中	在异构环境中定位错误比较复杂
程序的性能问题	中	流量分流的力度不够细, initial traffic load
人员/流程出错	好	取决于自动化框架
基础架构失效	优	Auto-Scaling
回滚	优	没有DNS的复杂性
成本	好	通过Auto Scaling来优化成本, 但是开始时需要额外的扩展

蓝/绿部署模式总结

模式 对风险的克服	经典式DNS切换	切换 Auto Scaling groups	切换 launch configs
程序的功能问题	Canary analysis	Canary analysis	Mixed fleet
程序的性能问题	Granular traffic switch	Instance- level granularity	Mixed fleet
人员/流程的错误	Automation: Use CloudFormation with Elastic Beanstalk, OpsWork, third party		
基础架构失效	Automation framework	Auto Scaling, ELB	Auto Scaling, ELB
回滚的能力	DNS	ELB	ELB
成本管理	Gradual scaling	Gradual scaling	Some over-provisioning
部署的复杂性	Simple, DNS weights	Auto Scaling control	Scale-in adjustments

Thank You