

# **Cloud-Powered Continuous Integration and Deployment**

**Jinesh Varia**  
**[jvaria@amazon.com](mailto:jvaria@amazon.com)**  
**@jinman**

# About Me



**Jinesh Varia**

@jinman

[jvaria@amazon.com](mailto:jvaria@amazon.com)

[linkedin/in/jinman](https://www.linkedin.com/in/jinman)

<http://jinesh.varia.in>



# “Cloud Architectures”

## Cloud Architectures

Jinesh Varia  
Technology Evangelist  
Amazon Web Services  
(jvaria@amazon.com)

Amazon Web Services - Architecting for The Cloud: Best Practices

January 2010

### Introduction

# “Cloud Best Practices aws”

pattern-matching across millions of a parallel computation on them can the virtual servers releasing all its r cost for the caller.

In the second section, we discuss : Amazon SimpleDB and Amazon EC2

### Keywords

Amazon Web Services, Amazon S3,



## Architecting for the Cloud: Best Practices

January 2010

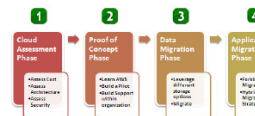
Amazon Web Services - Migrating Your Existing Applications to the AWS Cloud

October 2010

### A Phased Strategy for Migration: Step By Step Guide

Amazon Web Services - Migration Scenario: Backend Processing Pipeline

October 2010



### Migration Scenario: Migrating Backend Processing Pipeline to the AWS Cloud

Figure 1: The Phase Drives Approach to Cloud Migration

Phases	Backend Business
<b>Cloud Assessment</b> <ul style="list-style-type: none"><li>Financial Assessment (TCO calculation)</li><li>Security and Compliance Assessment</li><li>Technical Assessment (Classify application types)</li><li>Identify the tools that can be reused and the tools that need to be built</li><li>Migrate licensed products</li><li>Create a plan and measure success</li></ul>	Identify legacy web services
<b>Proof of Concept</b> <ul style="list-style-type: none"><li>Get your feet wet with AWS</li><li>Build a pilot and validate the technology</li><li>Test existing software in the cloud</li></ul>	Build or migrate program
<b>Moving your Data</b> <ul style="list-style-type: none"><li>Understand different storage options in the AWS cloud</li><li>Migrate fileservers to Amazon S3</li><li>Migrate commercial RDBMS to EC2 + EBS</li><li>Migrate MySQL to Amazon RDS</li></ul>	Backend Storage
<b>Moving your Apps</b> <ul style="list-style-type: none"><li>Finalize migration strategy</li><li>Hybrid migration strategy</li><li>Build "cloud aware" layers of code as needed</li><li>Create AMIs for each component</li></ul>	Backend Autoscale
<b>Leveraging the Cloud</b> <ul style="list-style-type: none"><li>Leverage other AWS services</li><li>Automate elasticity and SDLC</li><li>Harden security</li><li>Create dashboard to manage AWS resources</li></ul>	Backend Autoscale, Backend Higher
<b>Optimization</b> <ul style="list-style-type: none"><li>Optimize usage based on demand</li><li>Improve efficiency</li><li>Implement advanced monitoring and telemetry</li><li>Re-engineer your application</li><li>Decompose your relational databases</li></ul>	Backend Higher, Backend Higher

Page 1 of 23

Page 4 of 23

# “Cloud Migration aws”

Amazon Web Services Migration Scenario: Web Application Architecture

October 2010

### Migration Scenario: Migrating Web Applications to the AWS Cloud

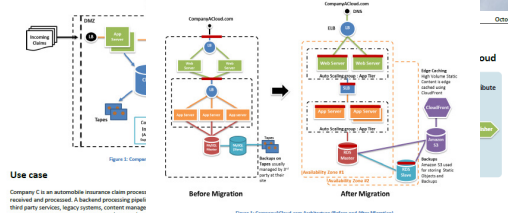


Figure 1: CompanyACloud.com Architecture (Before and After Migration)

CompanyACloud.com is a customer-facing web application of company A, which serves as a marketing portal and a customer management system. Customers, partners and employees use the web application to collaborate with each other using a rich web interface that can be viewed in a standard internet browser. CompanyACloud.com lists a complete catalog of products and their details. As new product announcements are made, marketing campaigns generate substantial amounts of traffic to the site resulting in periodic spikes. Outside of the timeframes created by these spikes, CompanyACloud.com experiences a fairly steady and predictable traffic load, which is characterized by high on weekdays and low on weekends. The website is currently hosted on dedicated infrastructure at the company's headquarters.

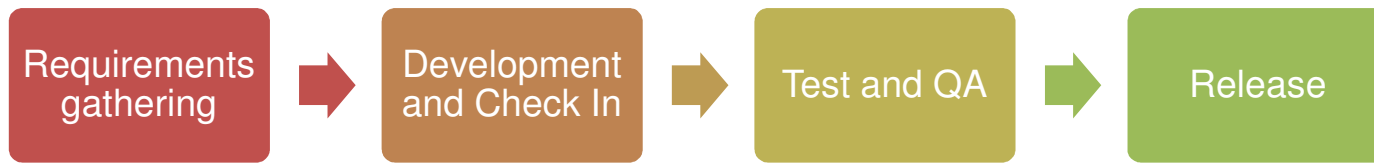
Using a standard 3-tier application architecture, the company deploys a front-end hardware based load balancer, which manages traffic across two Apache web servers each running on a separate physical box. The application is running

each processing infrastructure was hosted on-premise and consisted of 30 medium servers for transcoding and watermarking along with 5 high-end servers for encrypting digital rights management (DRM). The scheduling and monitoring components of the processing pipeline would queue the jobs in a database and kept track of job status.

each processing infrastructure was hosted on-premise and consisted of 30 medium servers for transcoding and watermarking along with 5 high-end servers for encrypting digital rights management (DRM). The scheduling and monitoring components of the processing pipeline would queue the jobs in a database and kept track of job status.



Customer is the center of our universe



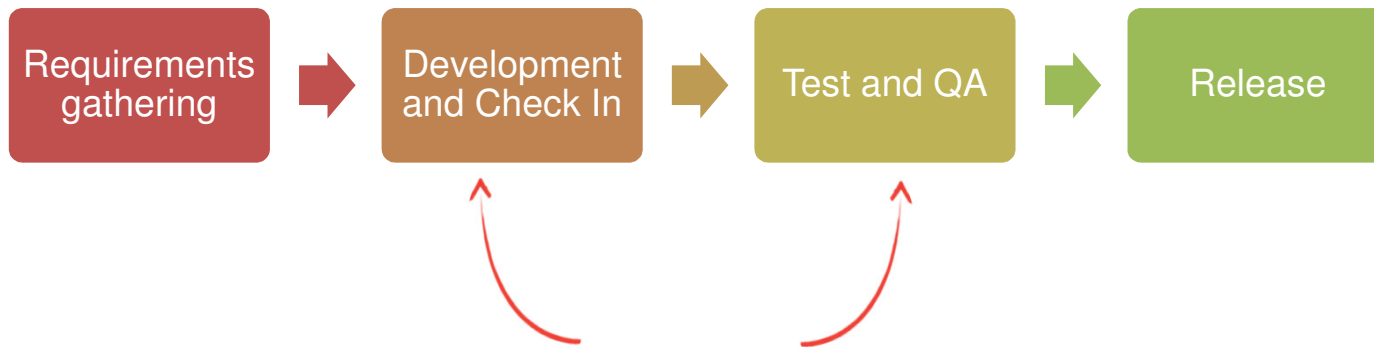
# Learning from customers

*Lots of Learning*



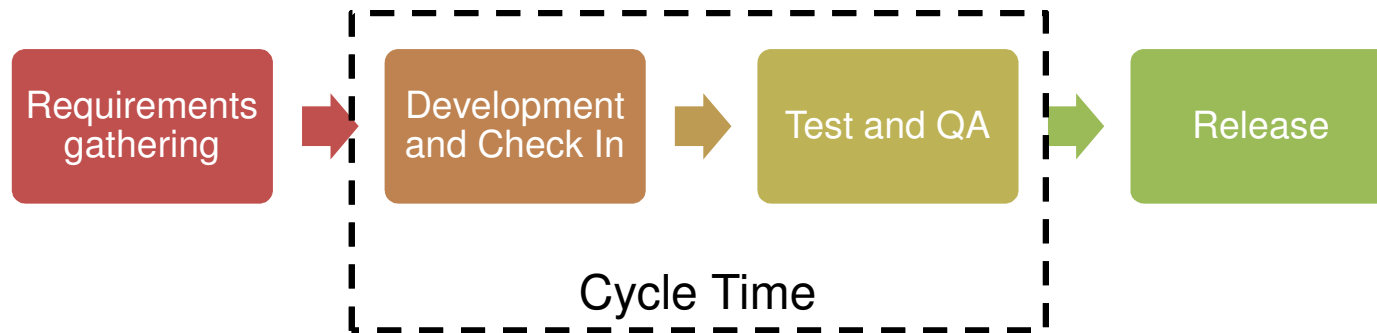
*Some Learning*

# Learning from customers



*Little or no Learning*

# Learning from customers





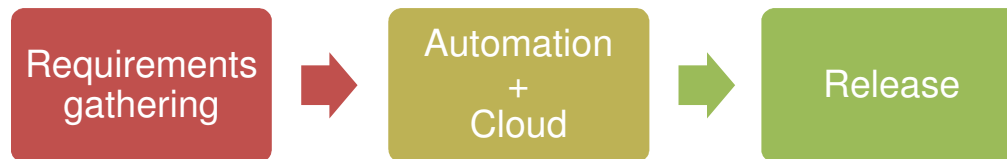
# Learning from customers



# Reduce the cycle time



Learn faster



... while keeping your costs low

# Cloud-powered Continuous Delivery



**Continuous  
Integration**



**Continuous  
Deployment**



**Continuous  
Optimization**

# Cloud-powered Continuous Integration



**Continuous  
Integration**

Goal: to have a working state of the code at any given time

Benefit: Fix bugs earlier when they are cheaper to fix

Metric: New guy can check out and compile at first day at job

Poka yo-ke (ポカヨケ)






Martin Fowler  
Paul Duvall  
Jez Humble  
David Farley  
Matt Graham  
Michael Nygard

.....

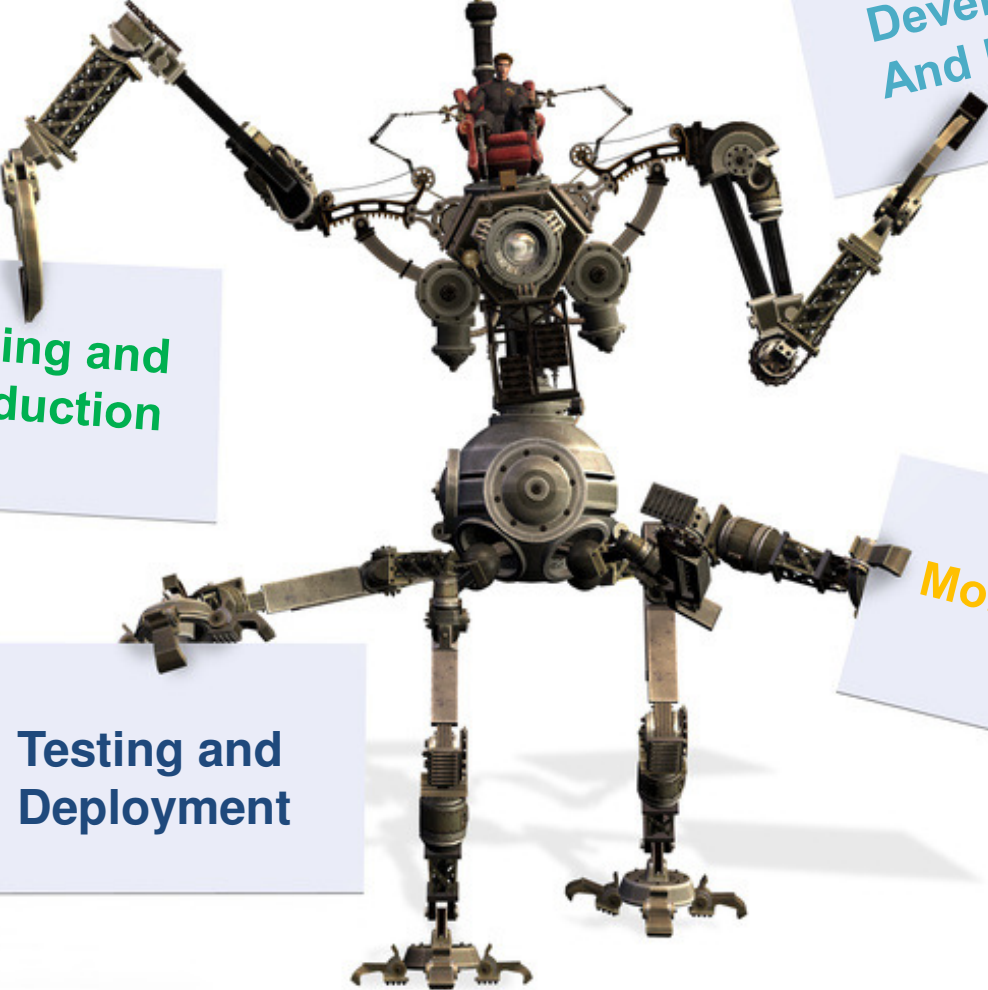
.....



Back to Sp

1. Keep absolutely everything in version control
2. Commit early and commit often
3. Always check in to trunk and avoid branching
4. Take responsibility if your check in breaks the build
5. Automate the build, test, deploy process
6. Be prepared to stop the mainline when/if build breaks
7. Create a comprehensive automated test suite
8. Only one way deploy and everybody uses that same way
9. Be prepared to revert to the previous revision
10. Continuously improve collaboration and increase speed of feedback





Hello, I am  
Mr. Automate

Development  
And Build

Staging and  
Production

Testing and  
Deployment

Monitoring

# Automate Everything

# Stress Reduction Kit



## Directions:

1. Place kit on FIRM surface.
2. Follow directions in circle of kit.
3. Repeat step 2 as necessary, or until unconscious.
4. If unconscious, cease stress reduction activity.

**Application Containers** - [JBoss](#), [Tomcat](#), [IIS](#), [Mongrel](#). *NOTE: there are so many app containers, I'm not going to try to list all of them.*

**Build Tools** - [Ant](#), [AntContrib](#), [NAnt](#), [MSBuild](#), [Buildr](#), [Gant](#), [Gradle](#), [make](#), [Maven](#), [Rake](#)

**Code Review** - [Crucible](#)

**Code Insight** - [Fisheye](#)

**Continuous Integration** - [Bamboo](#), [Jenkins](#), [AntHill Pro](#), [Go](#), [TeamCity](#), [TFS 2010](#)

**Database** - [Hibernate](#), [MySQL](#), [Liquibase](#), [Oracle](#), [PostgreSQL](#), [SQL Server](#), [SimpleDB](#), [SQL Azure](#), [Ant](#), [MongoDB](#)

**Database Change Management** - [dbdeploy](#), [Liquibase](#)

**Data Center Configuration Automation** - [Capistrano](#), [Cobbler](#), [BMC Bladelogic](#), [CFEngine](#), [IBM Tivoli Provisioning Manager](#), [Puppet](#), [Chef](#), [Bcfg2](#), [AWS Cloud Formation](#), [Windows Azure AppFabric](#) *NOTE: There are many names and overlap for this tool "category".*

**Dependency Management** - [Ivy](#), [Archiva](#), [Nexus](#), [Artifactory](#), [Bundler](#)

**Deployment Automation** - [Java Secure Channel](#), [ControlTier](#), [Altiris](#), [Capistrano](#), [Fabric](#), [Func](#)

**Information Sharing** - [Confluence](#), [Google Apps](#)

**Installer** - [InstallShield](#), [IzPack](#)

**Integrated Development Environment (IDE)** - [Eclipse](#), [IDEA](#), [Visual Studio](#)

**Issue Tracking** - [Greenhopper](#), [JIRA](#)

**Multi-Type** - [rPath](#)

**Passwords** - [PassPack](#), [PasswordSafe](#)

**Protected Configuration** - [ESCAPE](#), [ConfigGen](#)

**Project Management** - [JIRA](#), [Pivotal Tracker](#), [SmartSheet](#)

**Provisioning** - [JEOS](#), [BoxGrinder](#), [CLIP](#), [Eucalyptus](#), [AppLogic](#)

**Reporting/Documentation** - [Doxygen](#), [Grand](#), [GraphViz](#), [JavaDoc](#), [NDoc](#), [SchemaSpy](#), [UmlGraph](#)

**Static Analysis** - [CheckStyle](#), [Clover](#), [Cobertura](#), [FindBugs](#), [FxCop](#), [JavaNCSS](#), [JDepend](#), [PMD](#), [Sonar](#), [Simian](#)

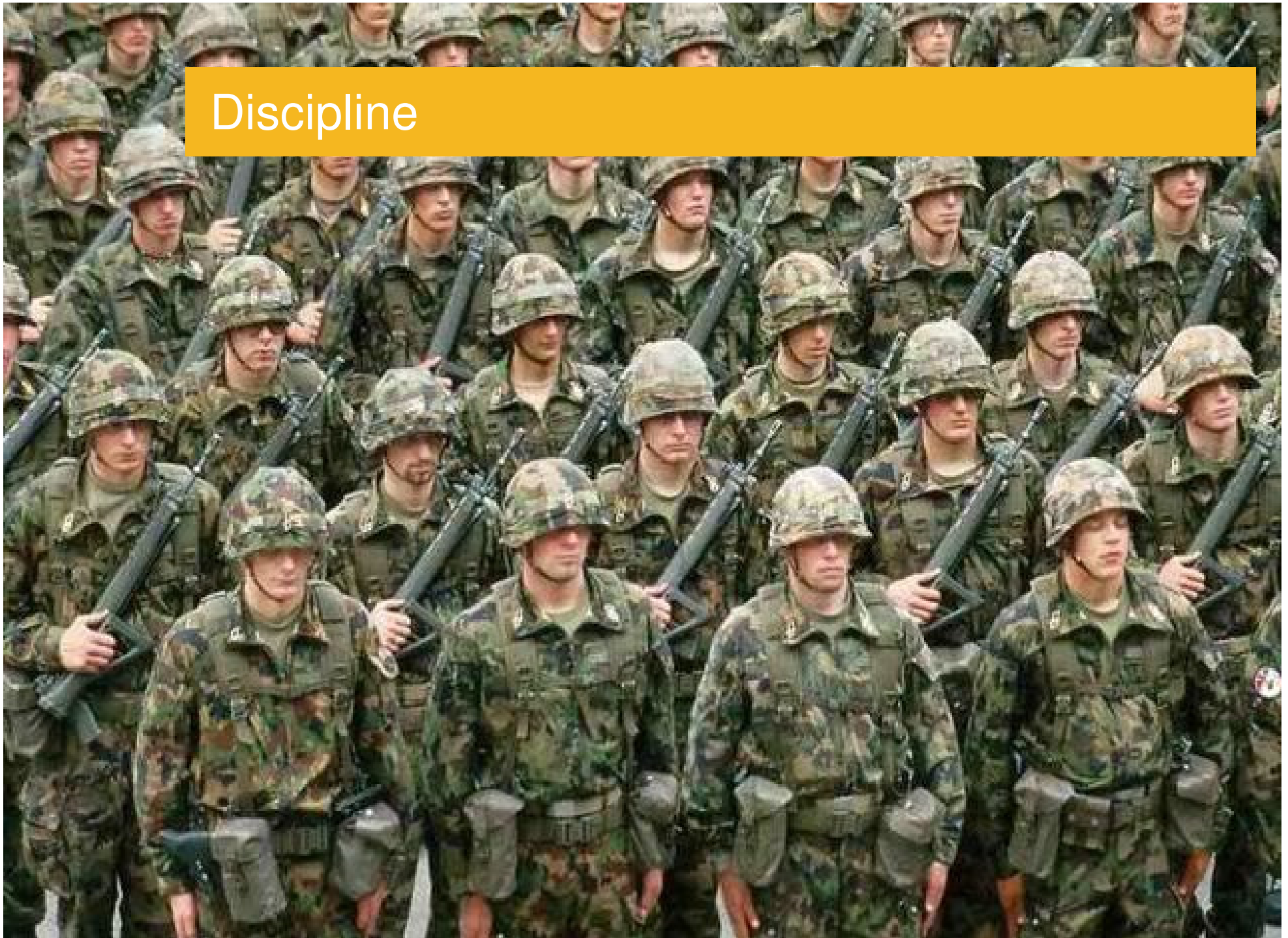
**Systems Monitoring** - [CloudKick](#), [Nagios](#), [Zabbix](#), [Zenoss](#)

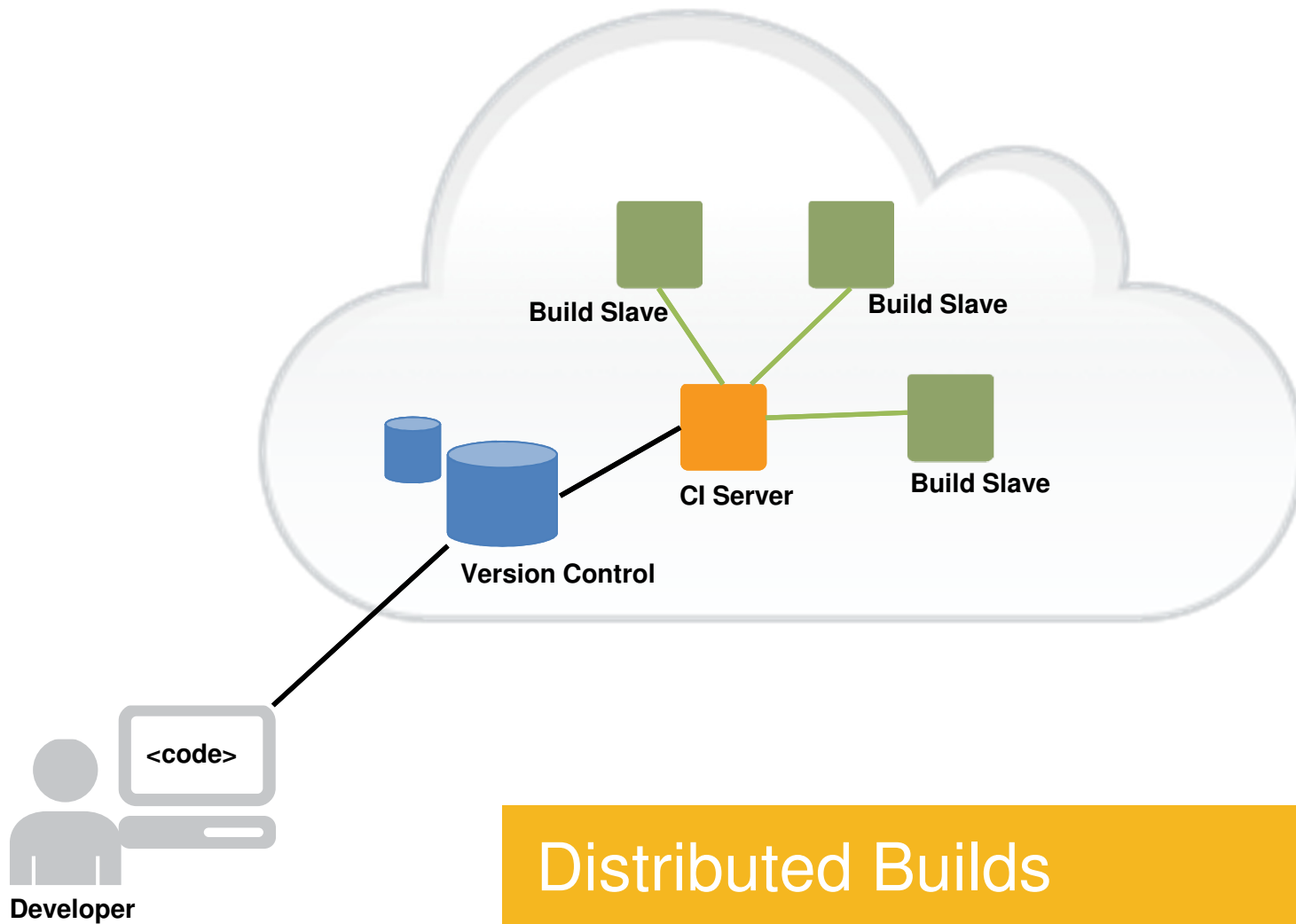
**Testing** [AntUnit](#), [Cucumber](#), [DbUnit](#), [webrat](#), [easyb](#), [Fitnesse](#), [JMeter](#), [JUnit](#), [NBehave](#), [SoapUI](#), [Selenium](#), [RSpec](#), [SauceLabs](#)

**Version-Control System** - [SVN/Subversion](#), [git](#), [Perforce](#)



# Discipline



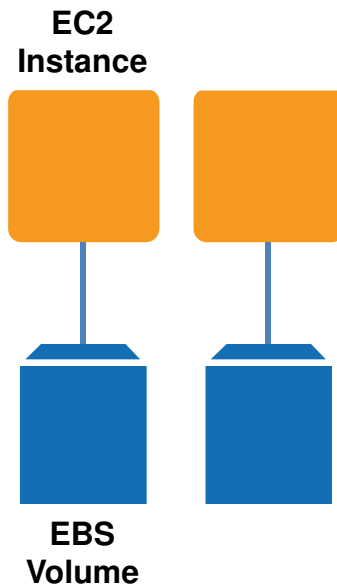


# Stop v/s Terminate



`createImage ()`

AMI



`RunInstances ()`



`StopInstances ()`  
`StartInstances ()`



`TerminateInstances ()`




# Jenkins

 cruisecontrol.

**CruiseControl.rb**  
Continuous Integration for Ruby

*continuum*

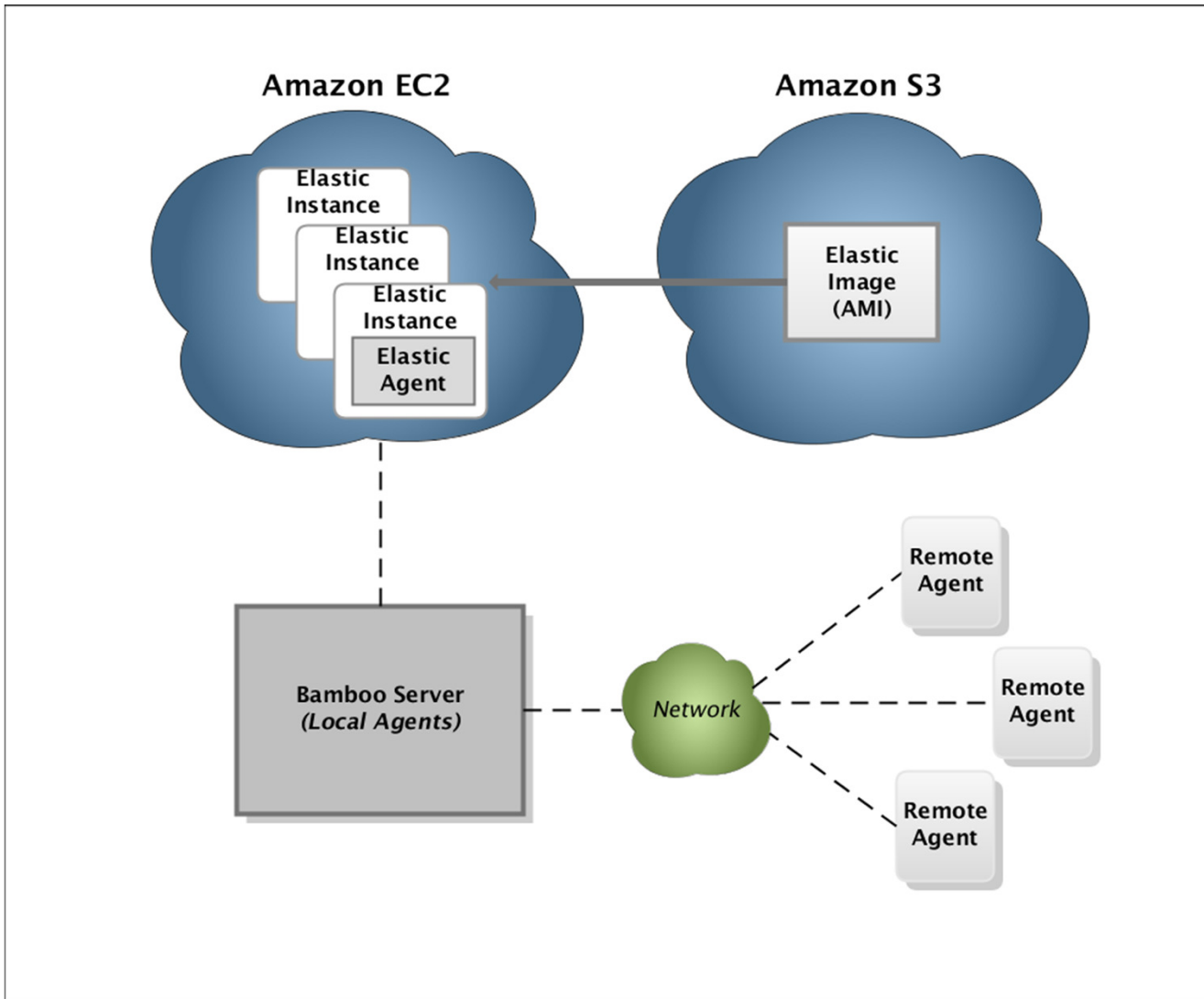
 **Bamboo**

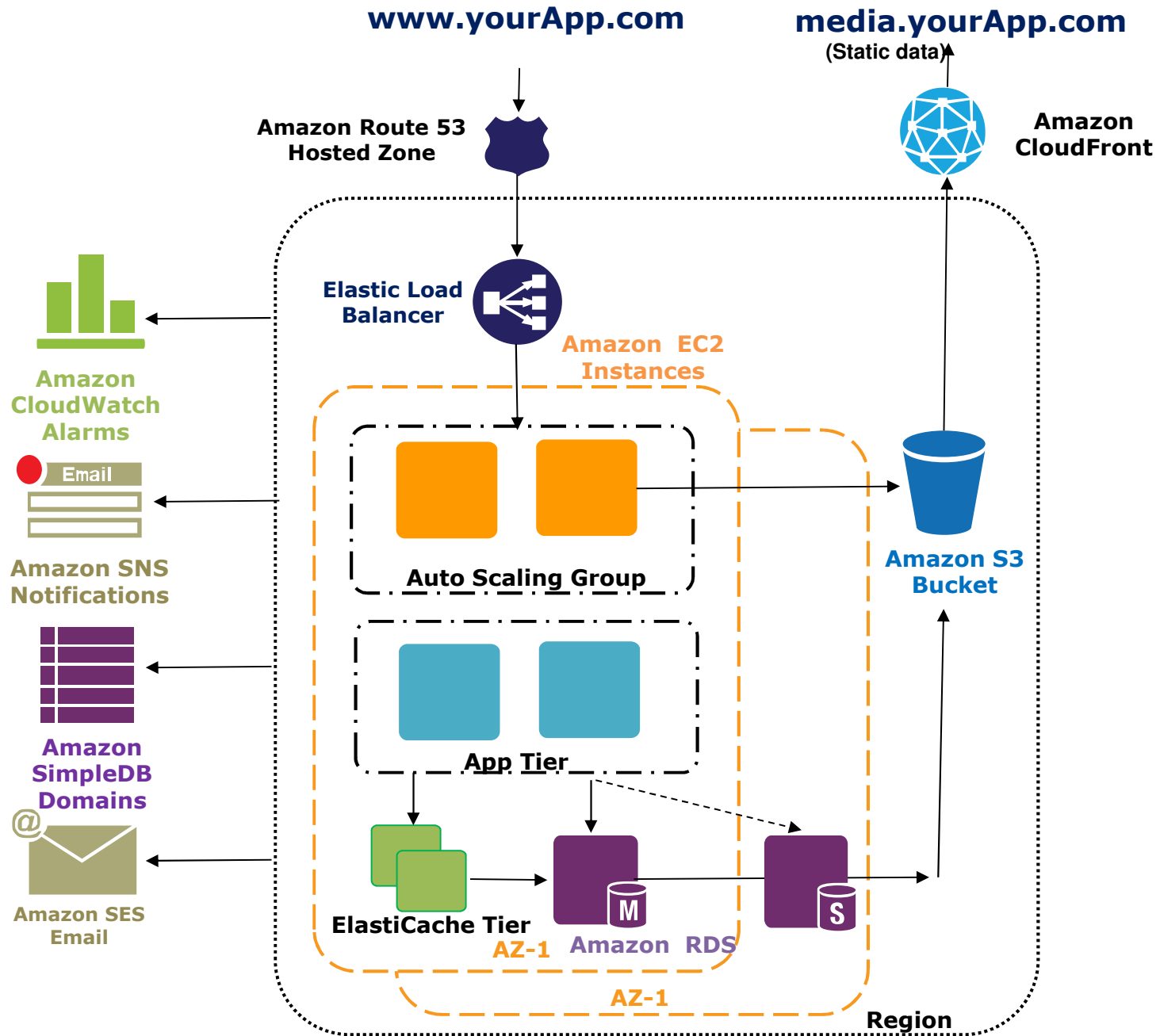
 ravis

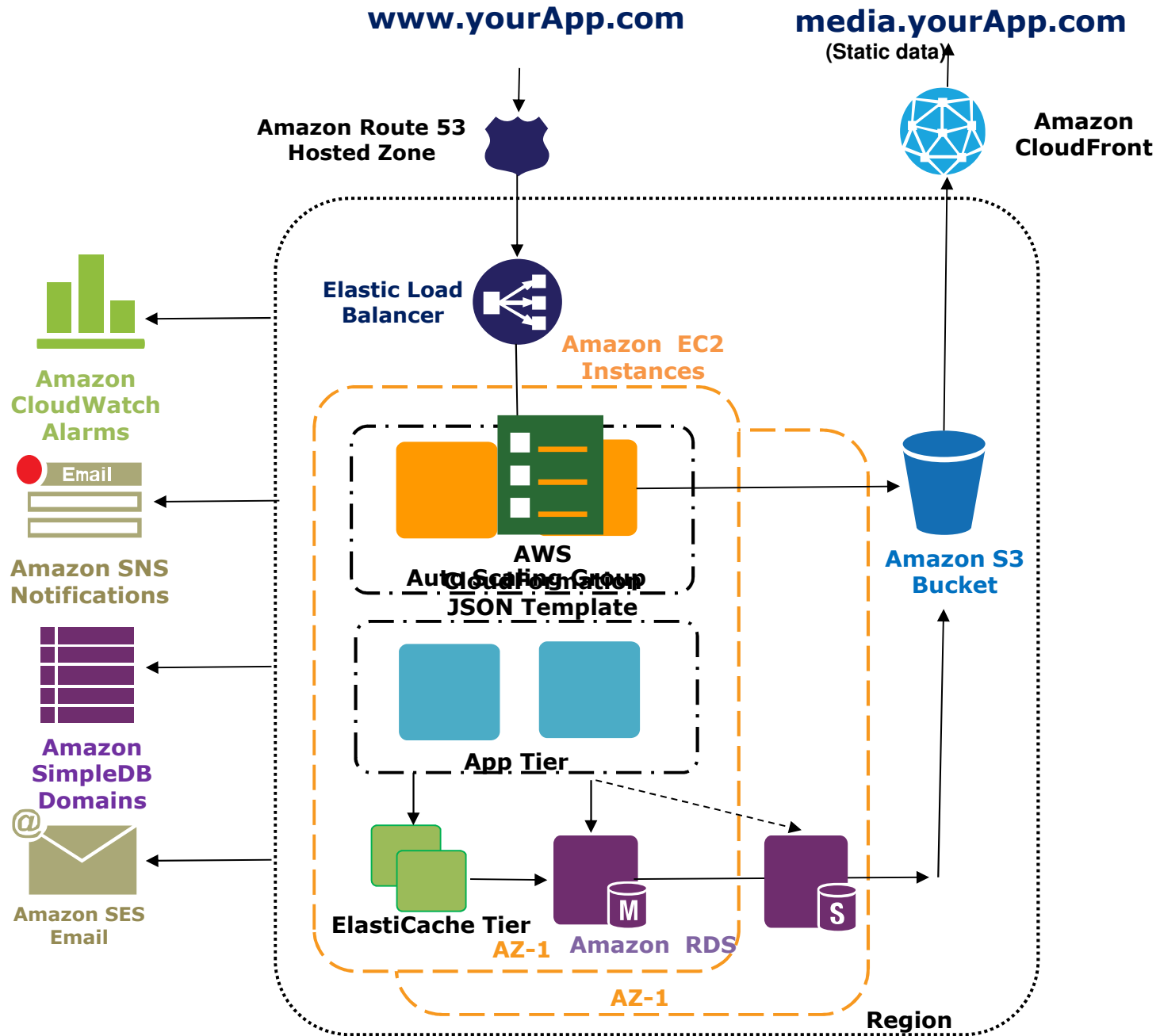
pm  ase

 **TeamCity**









**www.yourApp.com**

**media.yourApp.com**  
(Static data)

Amazon Route 53  
Hosted Zone

Amazon  
CloudFront

Elastic Load  
Balancer

Amazon EC2  
Instances

Amazon  
CloudWatch  
Alarms

Email

Amazon SNS  
Notifications

Amazon  
SimpleDB  
Domains

Amazon SES  
Email

AWS  
Auto Scaling Group  
JSON Template

Amazon S3  
Bucket

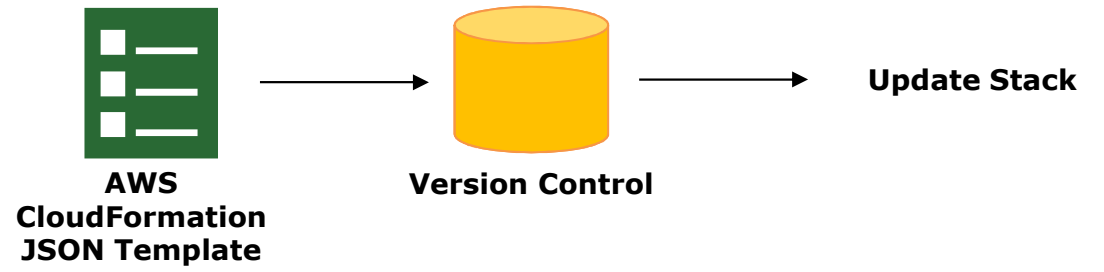
App Tier

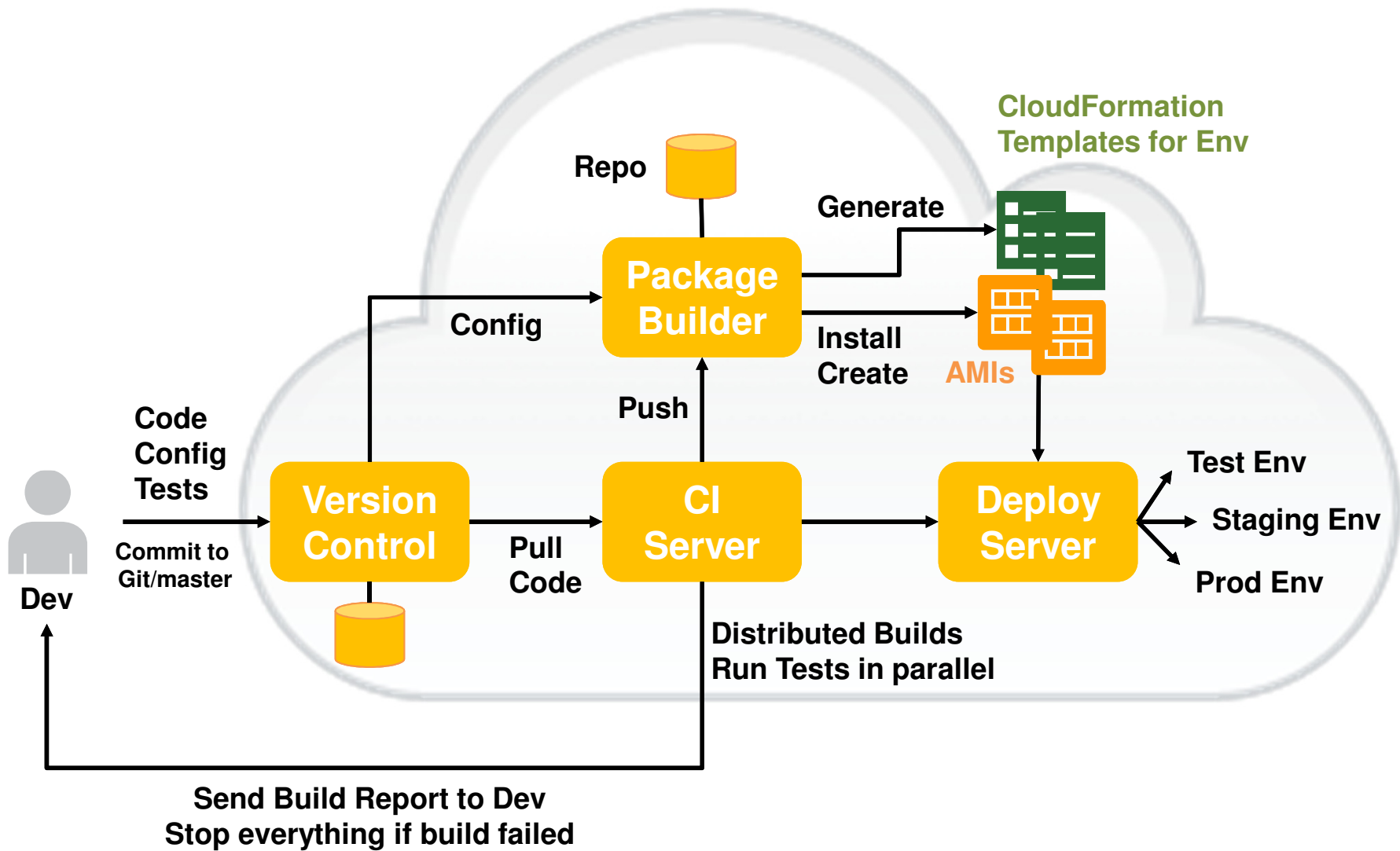
ElasticCache Tier

AZ-1 Amazon RDS

AZ-1

Region



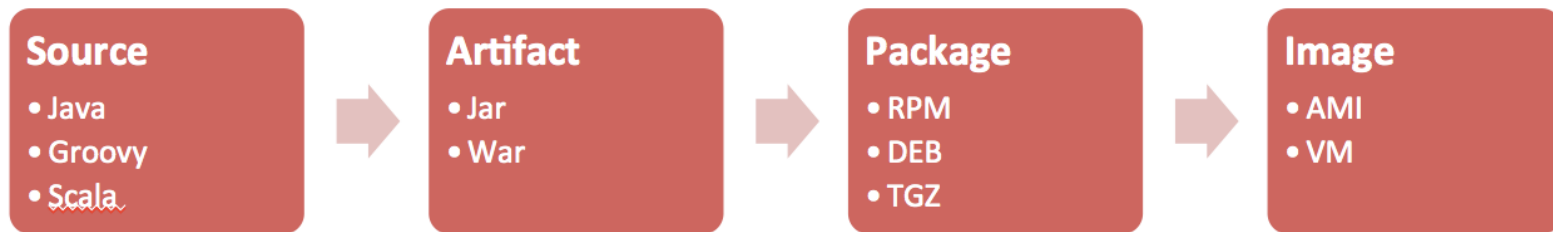


# Cloud Continuous Integration

# Cloud Continuous Integration

- Know exactly which version (and configuration) is currently in Test, Staging and Prod.
- Fast deployments with pre-configured AMIs and CF templates
- Easy to reproduce bugs by cloning the deployment into Test Environment
- Easy to spin up environments for ad-hoc requests from sales and marketing
- Poka-yoke design: Only one way to deploy

# NETFLIX



Build Job does the following:  
build the artifact,  
publish it to Artifactory,  
build the package,  
publish the package to the repo.

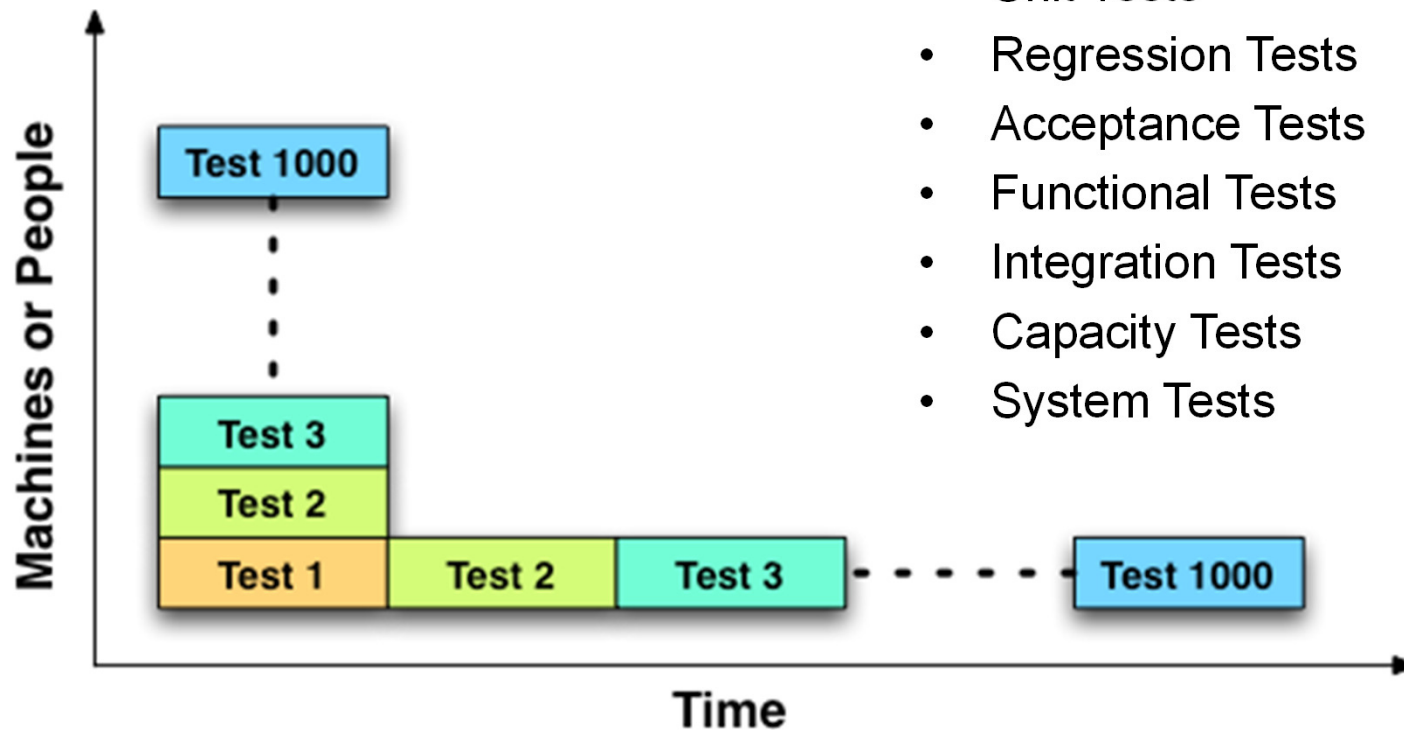
Then there is a follow on job that mounts a base OS image, installs the packages and then creates the final AMI.

Source: <http://techblog.netflix.com/2011/08/building-with-legos.html>

“TestOps”



# Automated Testing in Parallel



- Load Tests
- Web Performance Tests
- Usability Tests
- Unit Tests
- Regression Tests
- Acceptance Tests
- Functional Tests
- Integration Tests
- Capacity Tests
- System Tests

# RDS Snapshots – Test something quickly

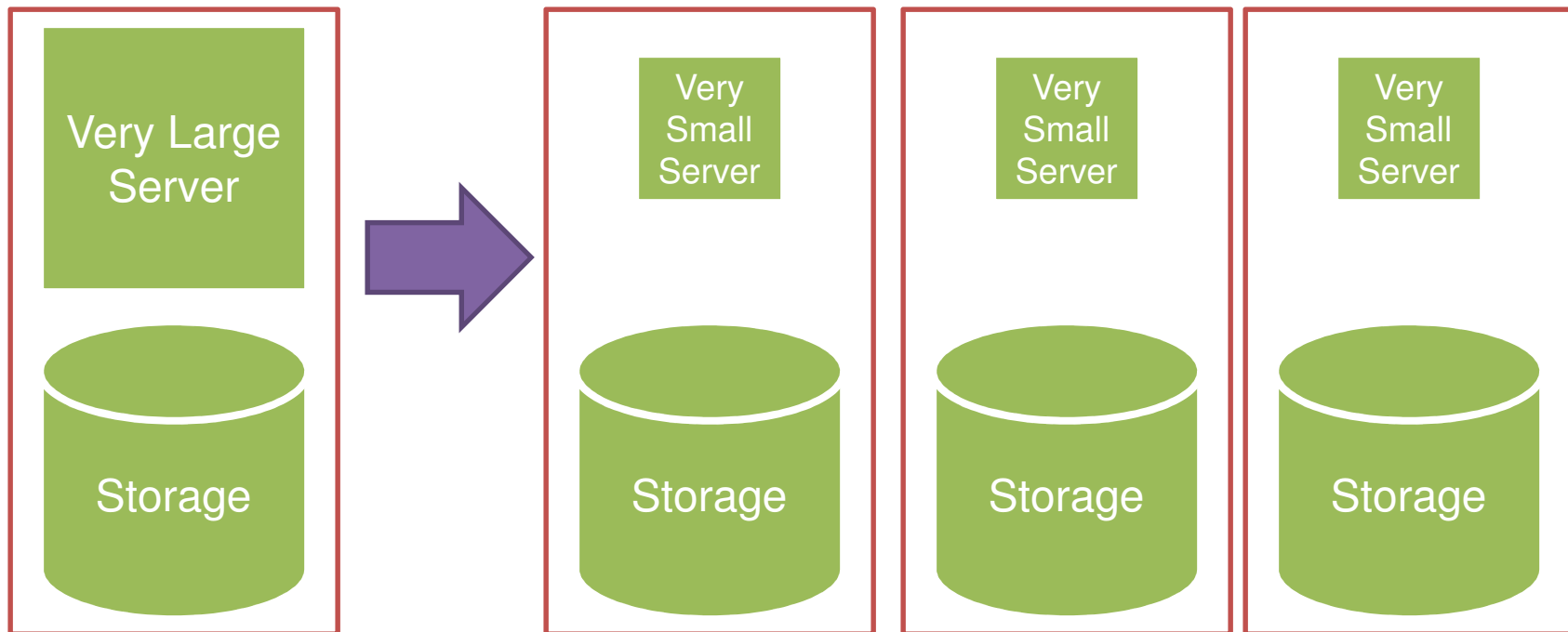
**ARM**

PROD

Bob Test

Ted Test

Mary Test



# Cloud-powered Continuous Deployment



**Continuous  
Integration**



**Continuous  
Deployment**

# Cloud-powered Continuous Deployment



**Continuous  
Deployment**

Goal: 1-click deploy and 1-click rollback

Benefit: Release early, release often and iterate quickly

Metric: fast feedback of your feature



Software Inventory is lost revenue





# Amazon May Continuous Deployment Stats

(production hosts and environments only)

**11.6**  
Seconds

**Mean Time between deployments (weekday)**

**1079**

**Max # of deployments in a single hour**

**10000**

**Mean # of hosts simultaneously receiving a deployment**

**30000**

**Max # of hosts simultaneously receiving a deployment**



The need for **speed**



**« Want to increase innovation?  
Lower the cost of failure »»**

Joli Ito



Break your problem into small batches

Small deployments

Incremental changes

Easy rollbacks

# Virtual Images = Real Productivity Gain

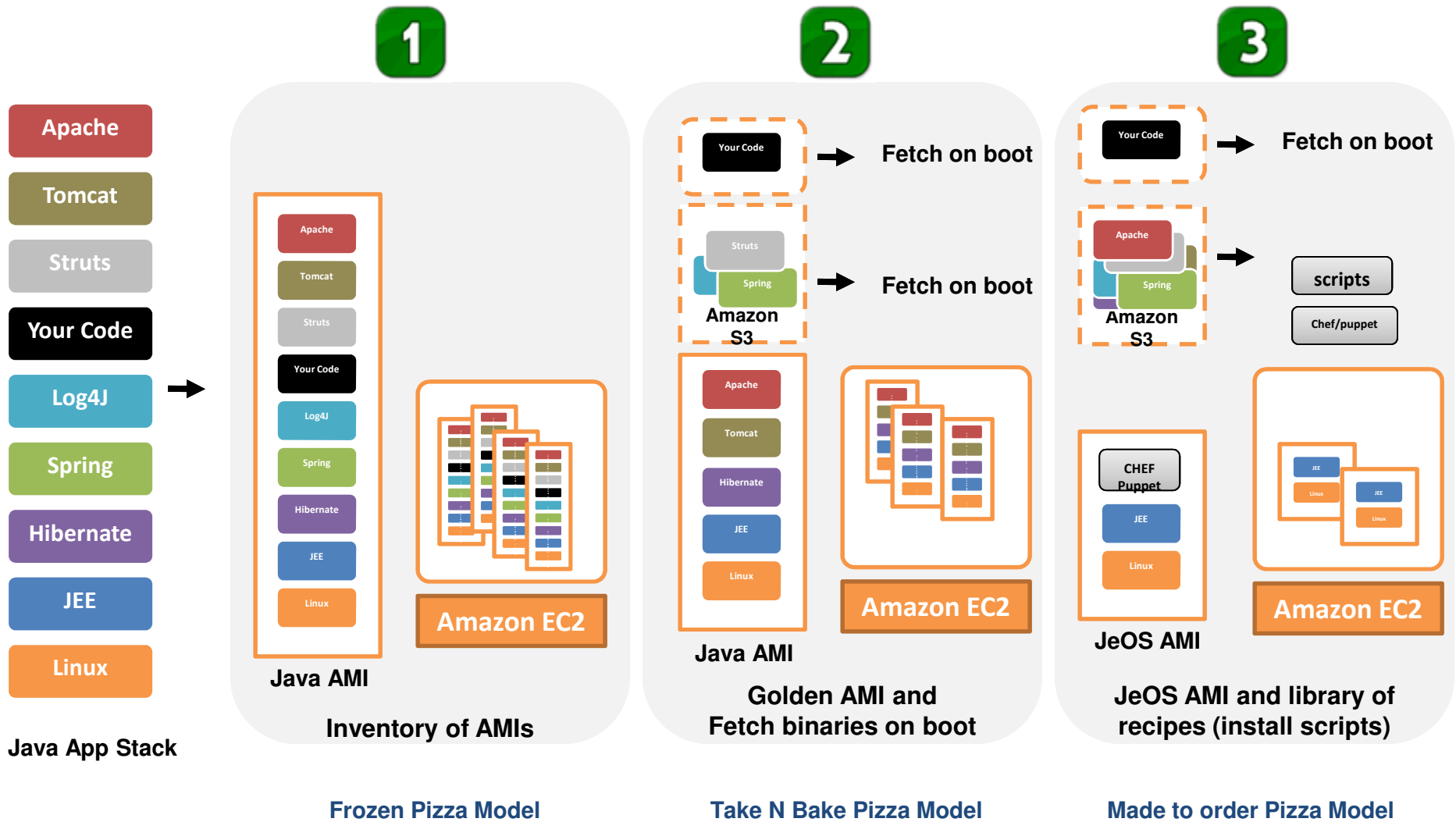


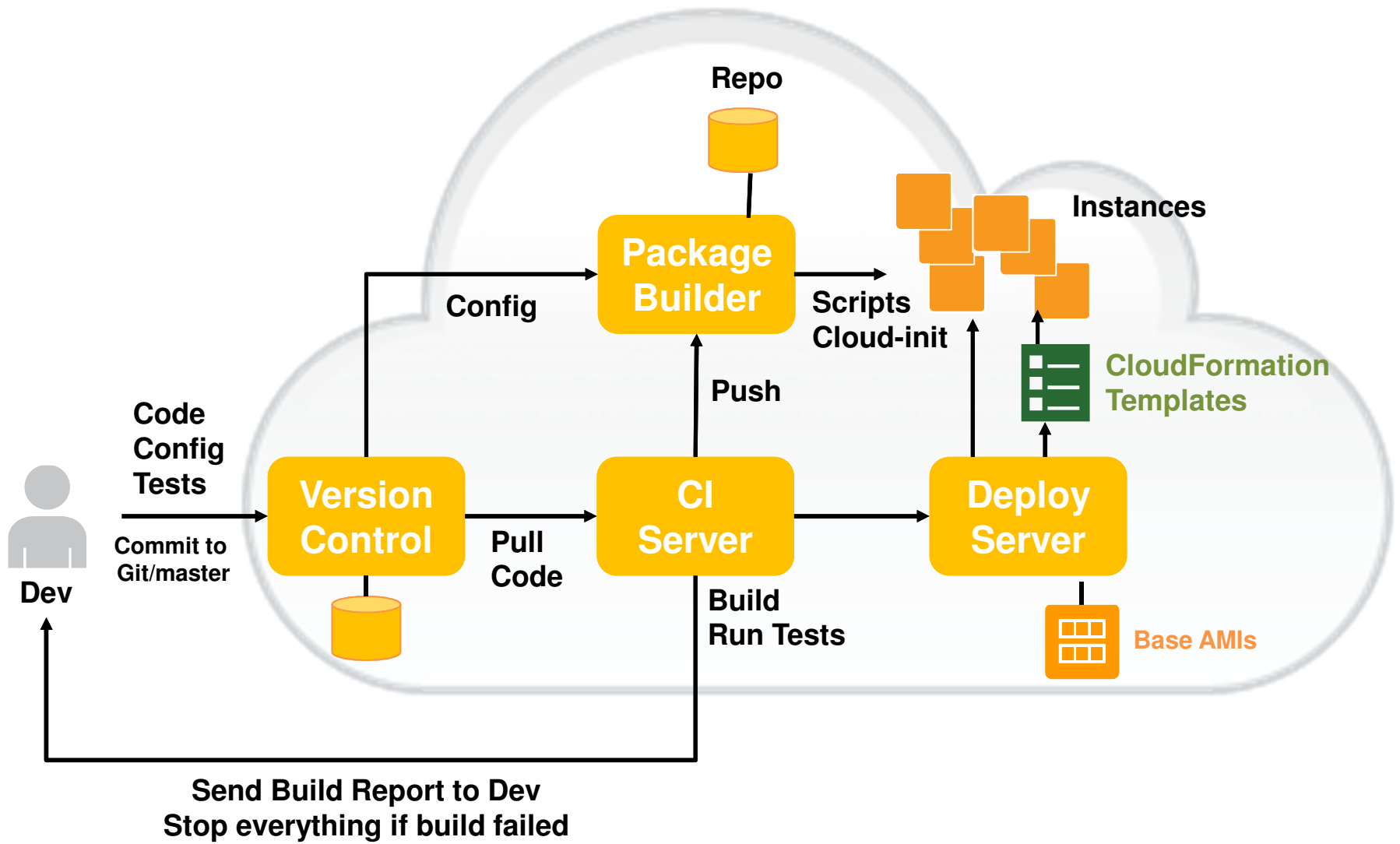
**Java Stack**

**.NET Stack**

**RoR stack**

# Push to an AMI or Pull from an Instance





## Cloud Continuous Deployment (Simple)

```
"UserData": {
  "Fn::Base64": {
    "Fn::Join": [
      "",
      [
        "#!/bin/bash -ex\n",
        "yum -y install git-core\n",
        "yum -y install php-pear\n",
        "pear install Crypt_HMAC2-1.0.0\n",
        "pear install HTTP_Request-1.4.4\n",
        "pear channel-discover
pear.amazonwebservices.com\n",
        "pear install aws/sdk\n",
```

*Bootstrap using User Data*

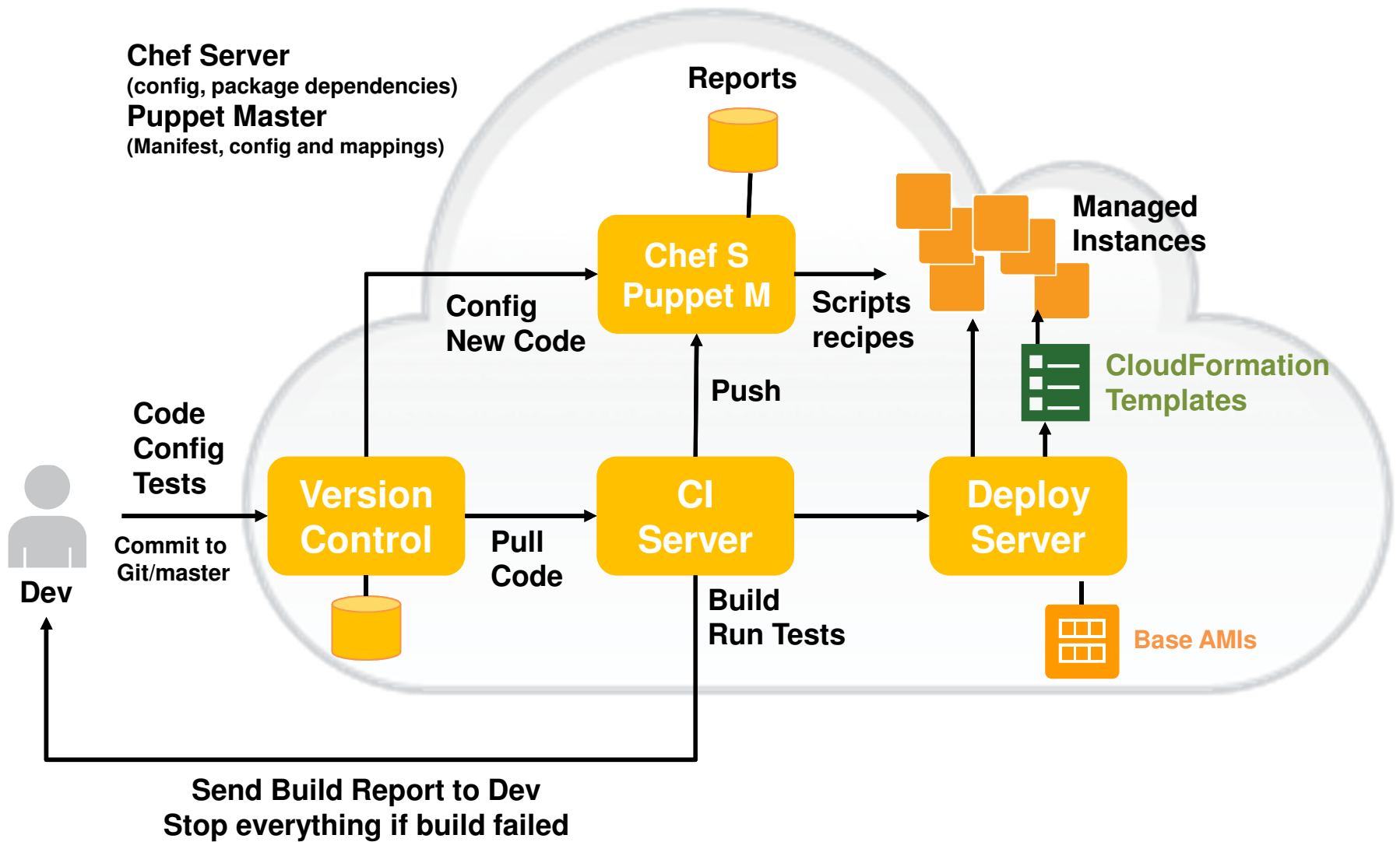


```
"AppDatabase": {"Type":  
"AWS::CloudFormation::Stack",  
"Metadata": { ... },  
"Properties": {  
  "TemplateURL": {  
    "Fn::Join": [  
      "/",  
      [  
        { ... },  
        "RDS_MySQL_55.template"  
      ]  
    }  
  }  
},
```

*Embedded Stacks*

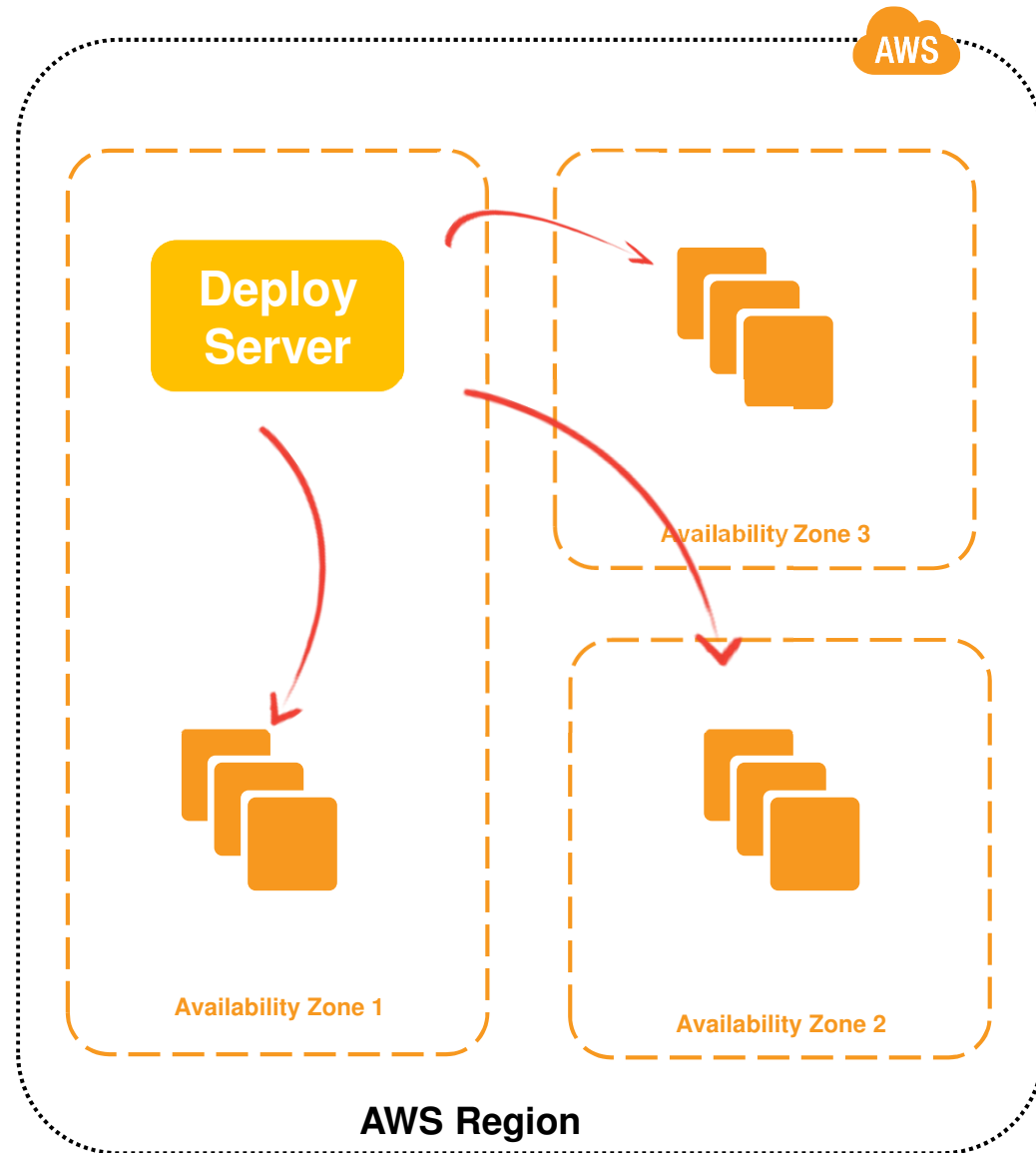






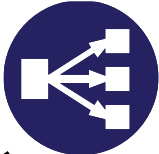
# Cloud Continuous Deployment

Automate deployment  
to multiple  
Availability Zones  
(Fault Tolerant Zones)



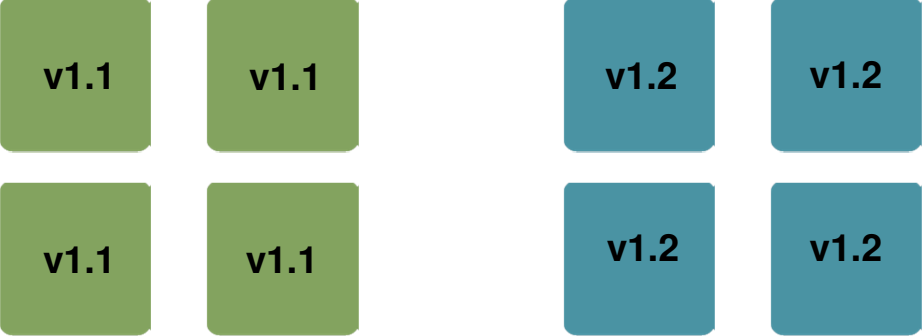
# Blue Green Deployments

**Load Balancing**  
(ELB)

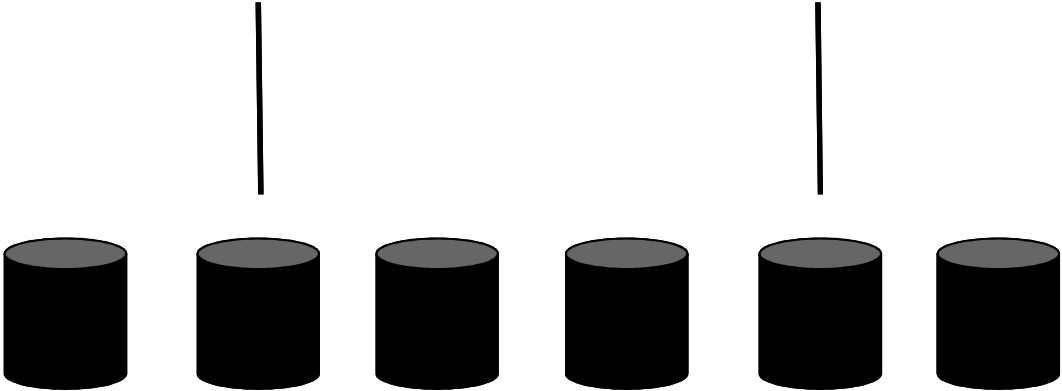


**High Error Rate**  
 **Monitoring**  
(CloudWatch)

**Web Server Fleet**  
(Amazon EC2)



**Database Fleet**  
(RDS or DB on EC2)



# Auto Scaling

**Auto Scaling Group** (Min, Max # of instances, Availability Zones .. )

**Health Check** (Maintain Min # active...)

**Launch Configuration** (AMIID, Instance type, UserData, Security Groups..)

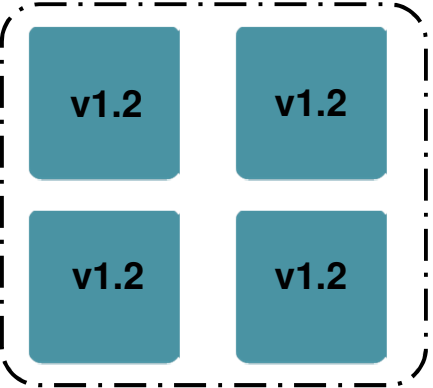
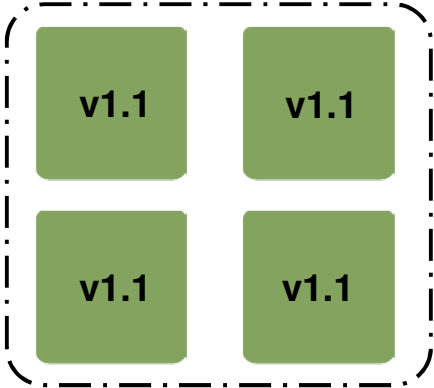
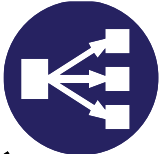
**Scaling Trigger** (Metric, Upper Threshold, Lower Threshold, Time interval ...)

**Types of Scaling** (Scale by Schedule, Scale by Policy)

**Alarm** (Notification Email, SMS, SQS, HTTP)

**Availability Zones and Regions**

**Load Balancing**  
(ELB)

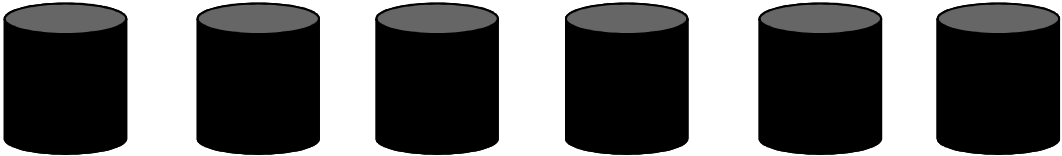


**Auto scaling**

- Max instances
- Min instances
- Scaling Trigger
- Custom Metrics
- Upper Threshold
- Lower Threshold
- Increment by

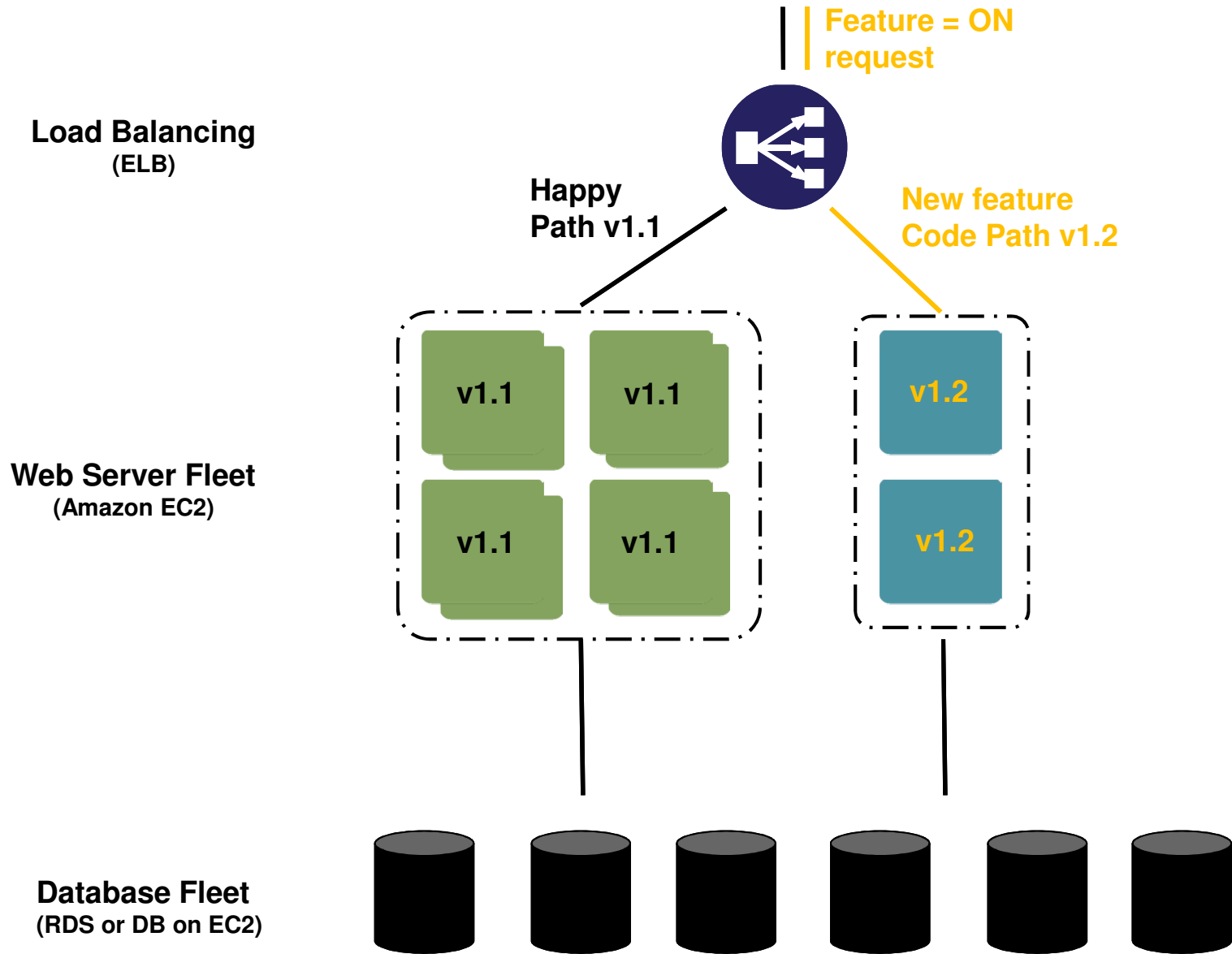
**“Auto scaling”**  
**Web Server Fleet**  
(Amazon EC2)

**Database Fleet**  
(RDS or DB on EC2)



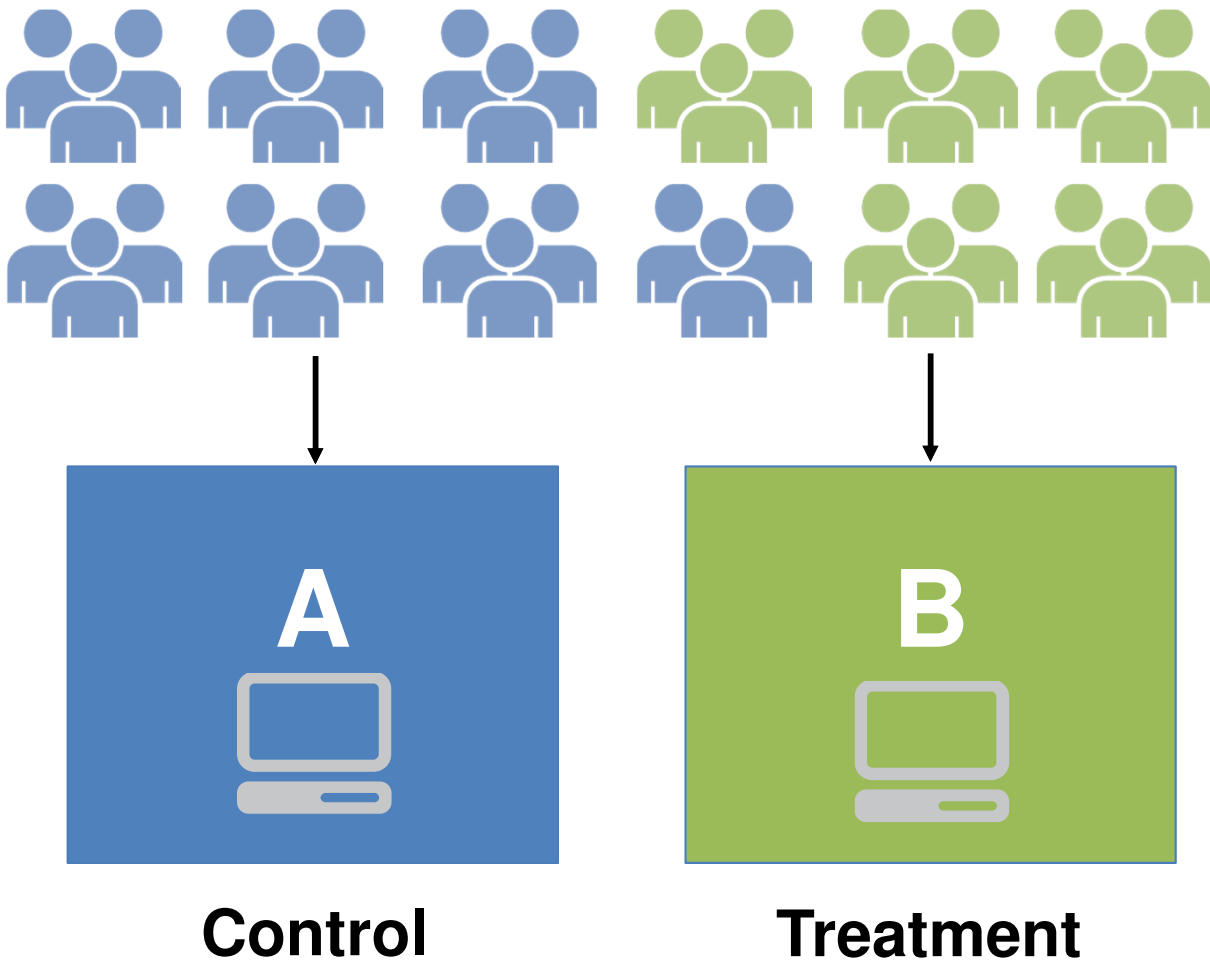
## Dark Launches with feature flags

**Deploy  $\neq$  Product Launch**





# Dialing up



**Load Balancing  
(ELB)**



↓99%

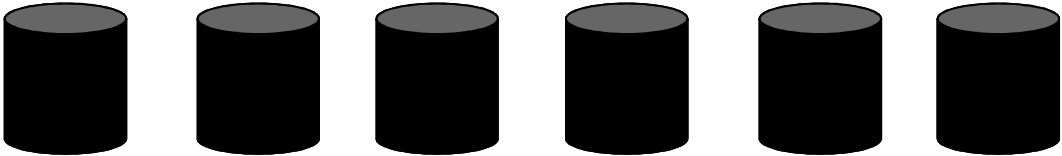
↑1%

**Web Server Fleet  
(Amazon EC2)**



**A/B Testing  
Service**

**Database Fleet  
(RDS or DB on EC2)**



**Load Balancing  
(ELB)**



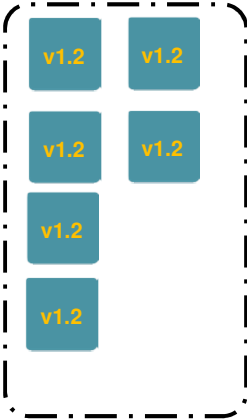
**↓90%**

**↑10%**

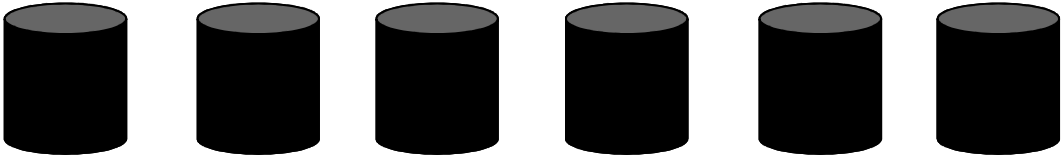


**A/B Testing  
Service**

**Web Server Fleet  
(Amazon EC2)**



**Database Fleet  
(RDS or DB on EC2)**



**Load Balancing  
(ELB)**



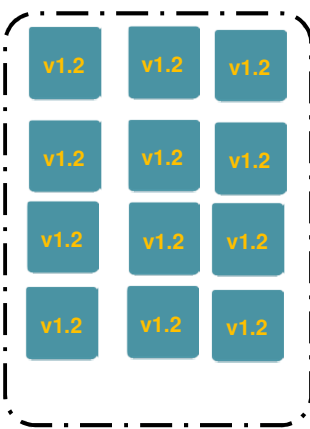
**↓70%**

**↑30%**

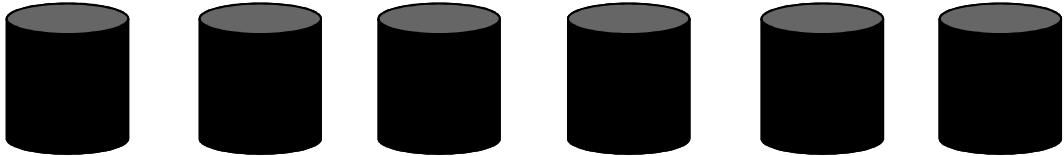


**A/B Testing  
Service**

**Web Server Fleet  
(Amazon EC2)**



**Database Fleet  
(RDS or DB on EC2)**

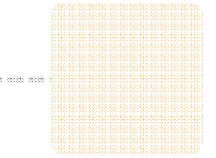


# Rollback



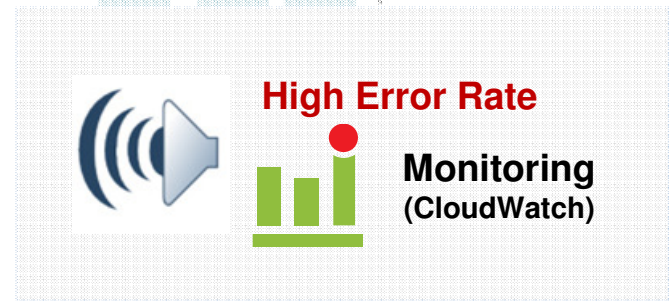
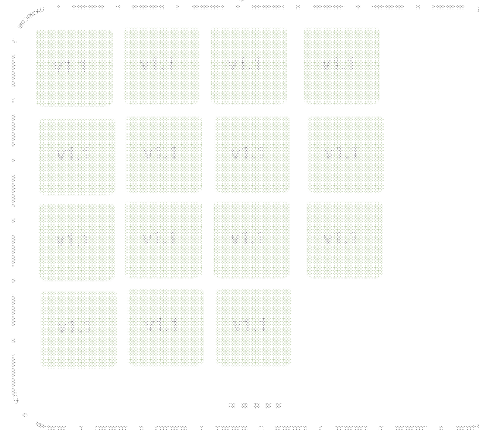
↓70%

↑30%

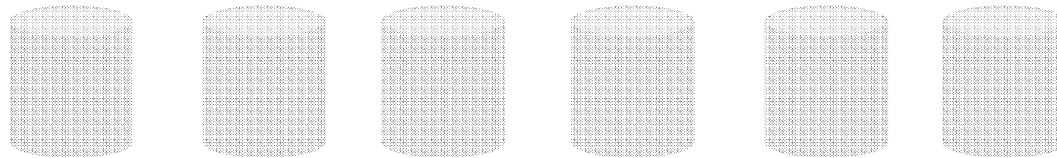


A/B Testing Service

Web Server Fleet  
(Amazon EC2)



Database Fleet  
(RDS or DB on EC2)



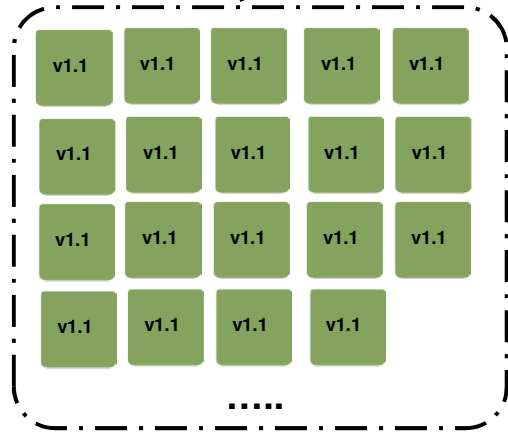
# Rollback



↑90%

↓10%

Web Server Fleet  
(Amazon EC2)

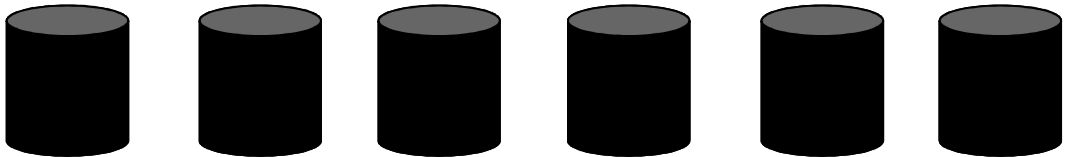


A/B Testing Service



Dev, Test

Database Fleet  
(RDS or DB on EC2)



**Load Balancing  
(ELB)**



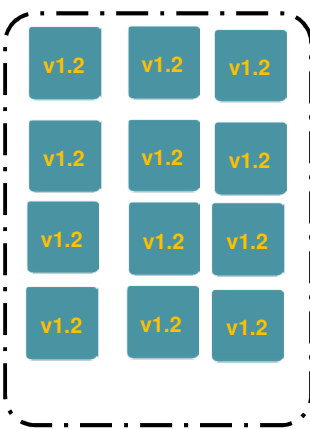
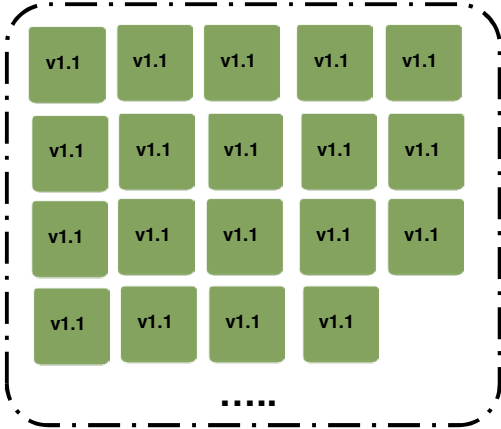
**↓70%**

**↑30%**

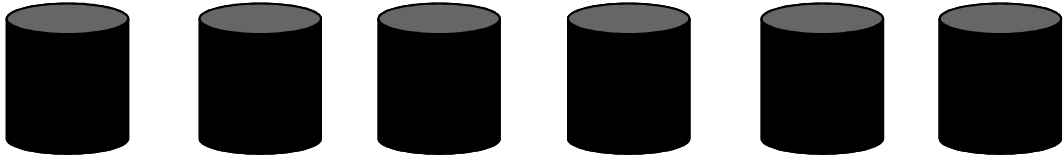


**A/B Testing  
Service**

**Web Server Fleet  
(Amazon EC2)**



**Database Fleet  
(RDS or DB on EC2)**



**Load Balancing (ELB)**



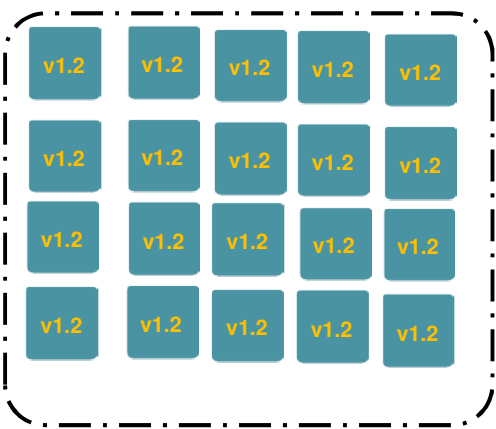
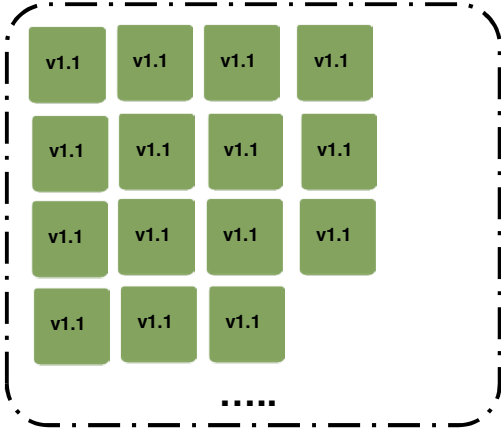
**↓50%**

**↑50%**

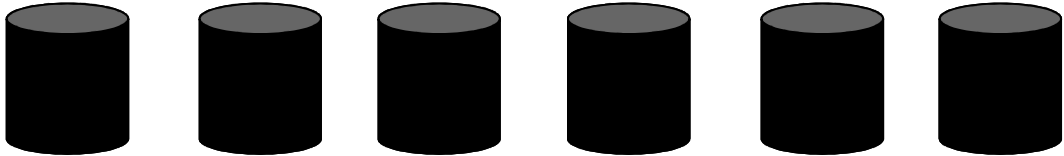


**A/B Testing Service**

**Web Server Fleet (Amazon EC2)**



**Database Fleet (RDS or DB on EC2)**





**Load Balancing  
(ELB)**



**↓30%**

**↑70%**

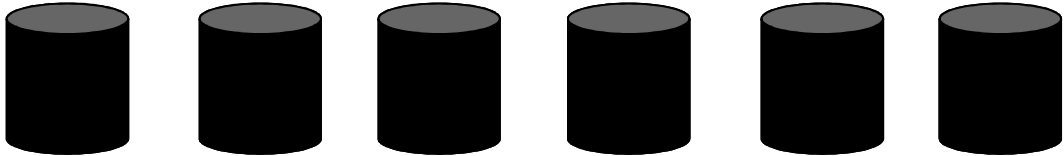


**A/B Testing  
Service**

**Web Server Fleet  
(Amazon EC2)**



**Database Fleet  
(RDS or DB on EC2)**



**Load Balancing  
(ELB)**



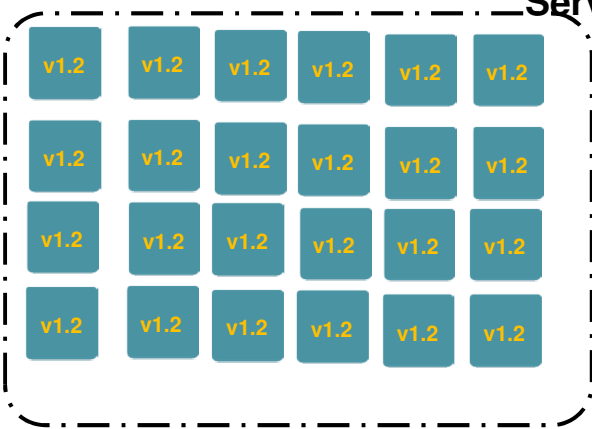
**↓5%**

**↑95%**

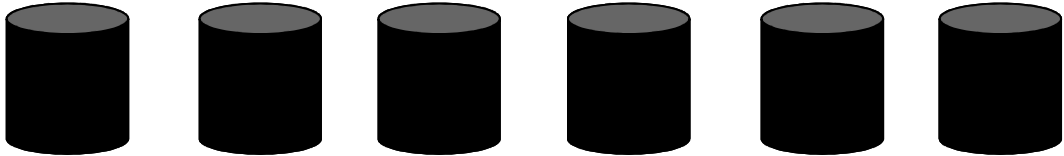


**A/B Testing  
Service**

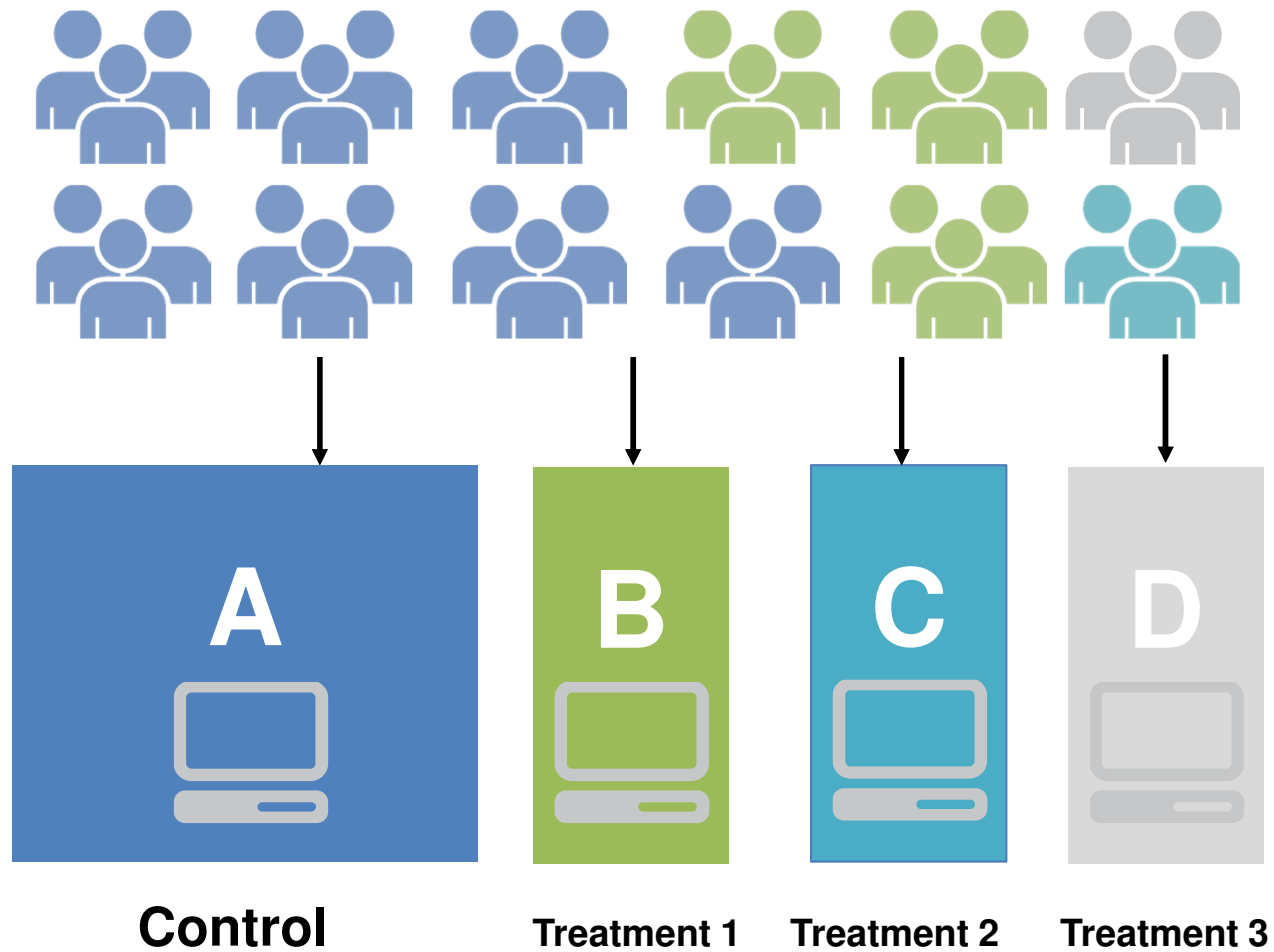
**Web Server Fleet  
(Amazon EC2)**



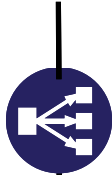
**Database Fleet  
(RDS or DB on EC2)**



# Split Users by multiple experiments



**Load Balancing  
(ELB)**



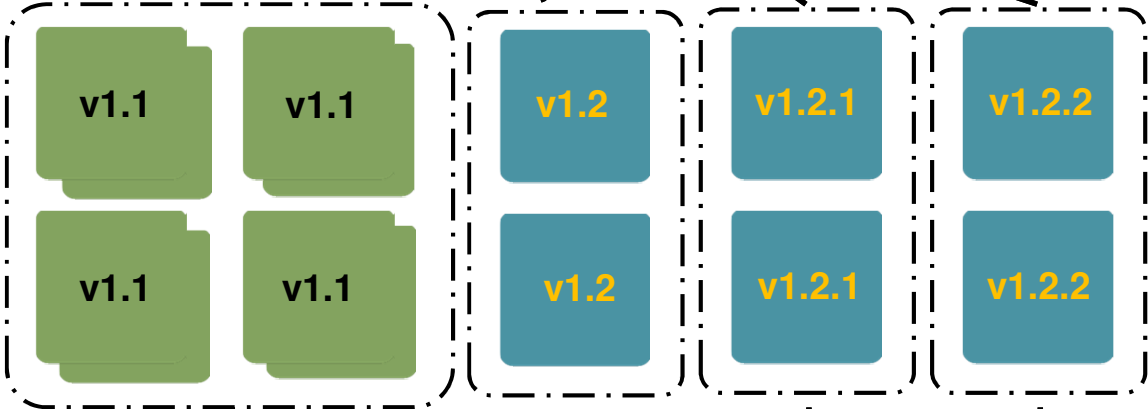
**90%**

**5%**

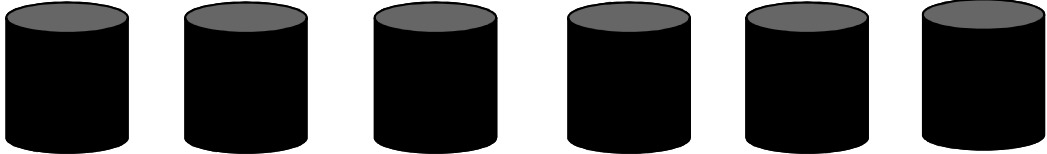
**3%**

**2%**

**Web Server Fleet  
(Amazon EC2)**



**Database Fleet  
(RDS or DB on EC2)**



# Blue Green Deployments : RDBMS?

## Mastering the Tradeoffs:

1. Hot deployment techniques
2. Simple Backup and Restore (RDS User-initiated Snapshots)
3. Break huge jobs into series of multiple backward-compatible tasks and decouple Application deployment and DB migration

# Example

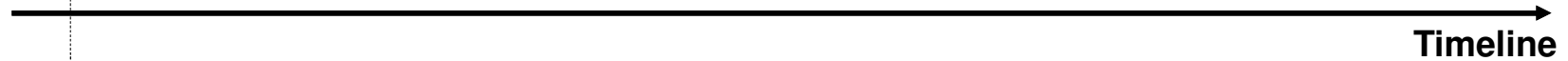
Timeline

ID	NAME	ADDRESS	ORDERID (Char)
23234	Joe Doe	xxx	333424
45322	Rob Smith	xxxx	234
2342342	Jane Smith	xxxx	23424
2342265	Anne Lee	xxxx	2342425

# Example

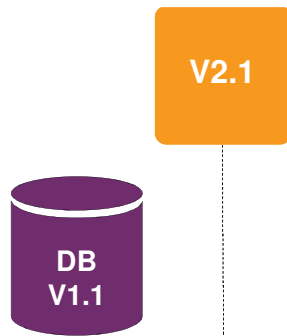
## 6 Steps

1. Add a new column



ID	NAME	ADDRESS	ORDERID	ORDERID_INT
23234	Joe Doe	xxx	333424	
45322	Rob Smith	xxxx	234	
2342342	Jane Smith	xxxx	23424	
2342265	Anne Lee	xxxx	2342425	

# Example



## 6 Steps

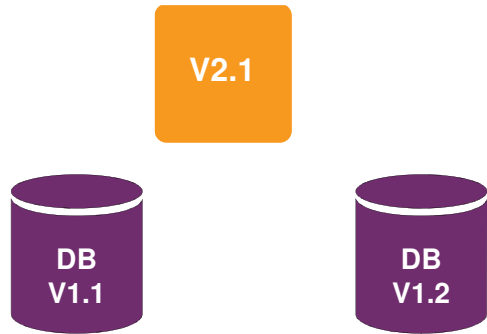
1. Add a new column
2. Change all software to write to both columns (old and new)

Deployment Timeline →

ID	NAME	ADDRESS	ORDERID	ORDERID_INT
23234	Joe Doe	xxx	333424	
45322	Rob Smith	xxxx	234	
2342342	Jane Smith	xxxx	23424	
2342265	Anne Lee	xxxx	2342425	
3632342	Mark Fox	cxxxx	<b>567890</b>	<b>567890</b>



# Example



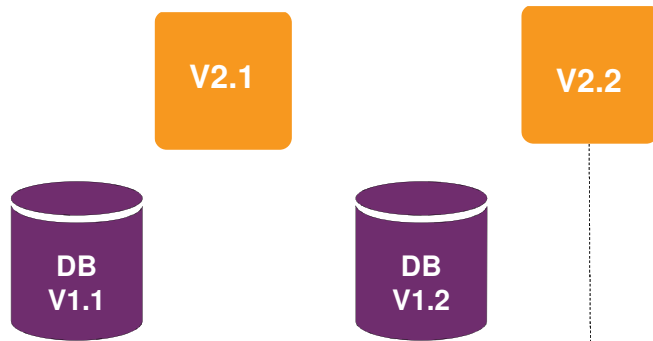
## 6 Steps

1. Add a new column
2. Change all software to write to both columns (old and new)
3. Migrate the old column data to new column (for older rows)

Deployment Timeline →

ID	NAME	ADDRESS	ORDERID	ORDERID_INT
23234	Joe Doe	xxx	333424	333424
45322	Rob Smith	xxxx	234	234
2342342	Jane Smith	xxxx	23424	23424
2342265	Anne Lee	xxxx	2342425	2342425
3632342	Mark Fox	cxxxx	<b>567890</b>	<b>567890</b>

# Example



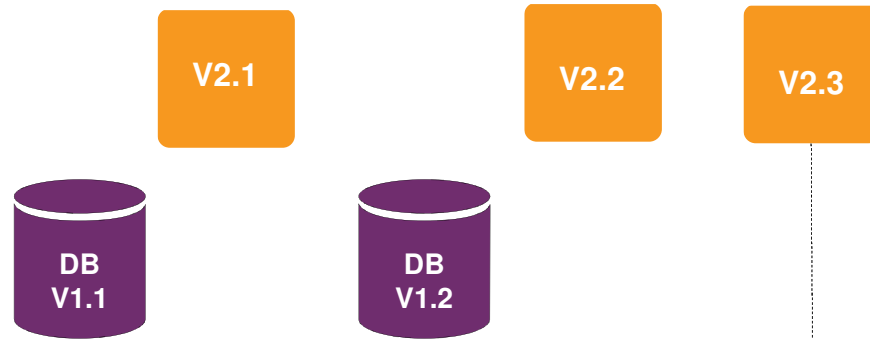
## 6 Steps

1. Add a new column
2. Change all software to write to both columns (old and new)
3. Migrate the old column data to new column (for older rows)
4. Change all software to read from the new column

Deployment Timeline

ID	NAME	ADDRESS	ORDERID	ORDERID_INT
23234	Joe Doe	xxx	333424	333424
45322	Rob Smith	xxxx	234	234
2342342	Jane Smith	xxxx	23424	23424
2342265	Anne Lee	xxxx	2342425	2342425
3632342	Mark Fox	cxxxx	<b>567890</b>	<b>567890</b>

# Example

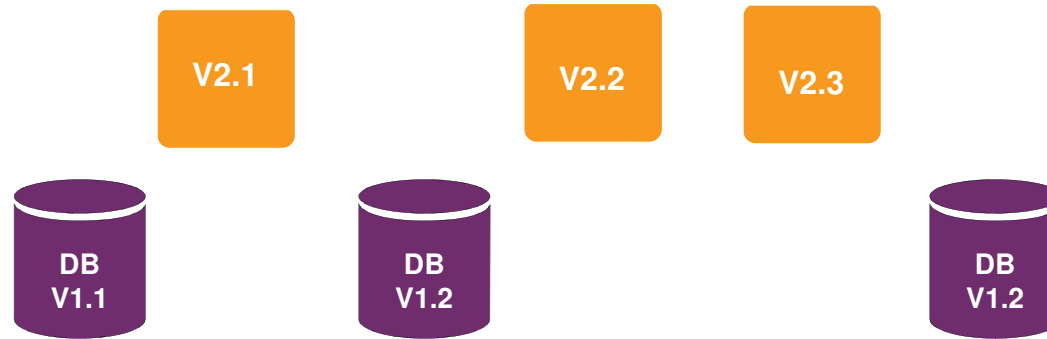


## 6 Steps

1. Add a new column
2. Change all software to write to both columns (old and new)
3. Migrate the old column data to new column (for older rows)
4. Change all software to read from the new column
5. Change all the software to only write to the new column

ID	NAME	ADDRESS	ORDERID	ORDERID_INT
23234	Joe Doe	xxx	333424	333424
45322	Rob Smith	xxxx	234	234
2342342	Jane Smith	xxxx	23424	23424
2342265	Anne Lee	xxxx	2342425	2342425
3632342	Mark Fox	cxxxx	567890	567890
433453	Mark Lee	cxxxxx		<b>4352342422</b>

# Example

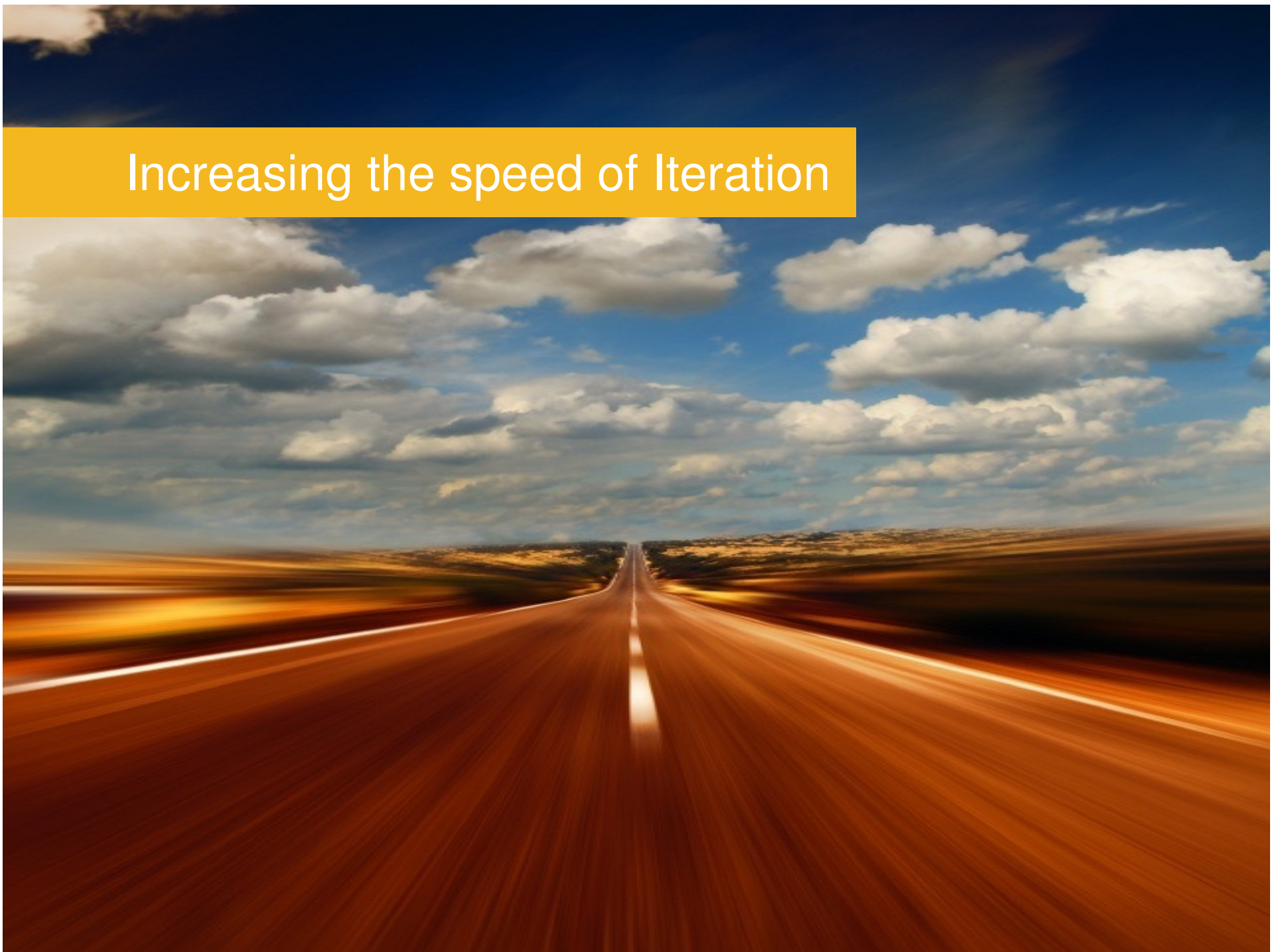


## 6 Steps

1. Add a new column
2. Change all software to write to both columns (old and new)
3. Migrate the old column data to new column (for older rows)
4. Change all software to read from the new column
5. Change all the software to only write to the new column
6. Drop the old column

ID	NAME	ADDRESS	ORDERID_INT
23234	Joe Doe	xxx	333424
45322	Rob Smith	xxxx	234
2342342	Jane Smith	xxxx	23424
2342265	Anne Lee	xxxx	2342425
3632342	Mark Fox	cxxxx	567890
433453	Mark Lee	cxxxxx	<b>4352342422</b>

# Increasing the speed of Iteration





Break your problem into small batches

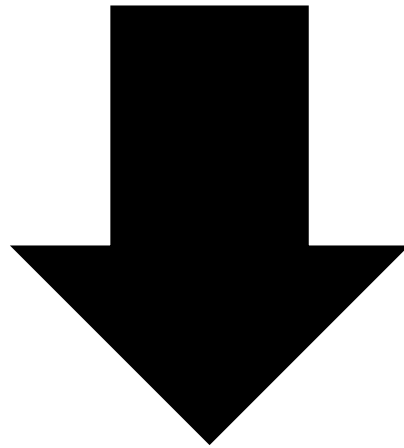
Stream of small deployments

Incremental changes

Easy rollbacks

Deploy new software quickly

Revert a bad change quickly

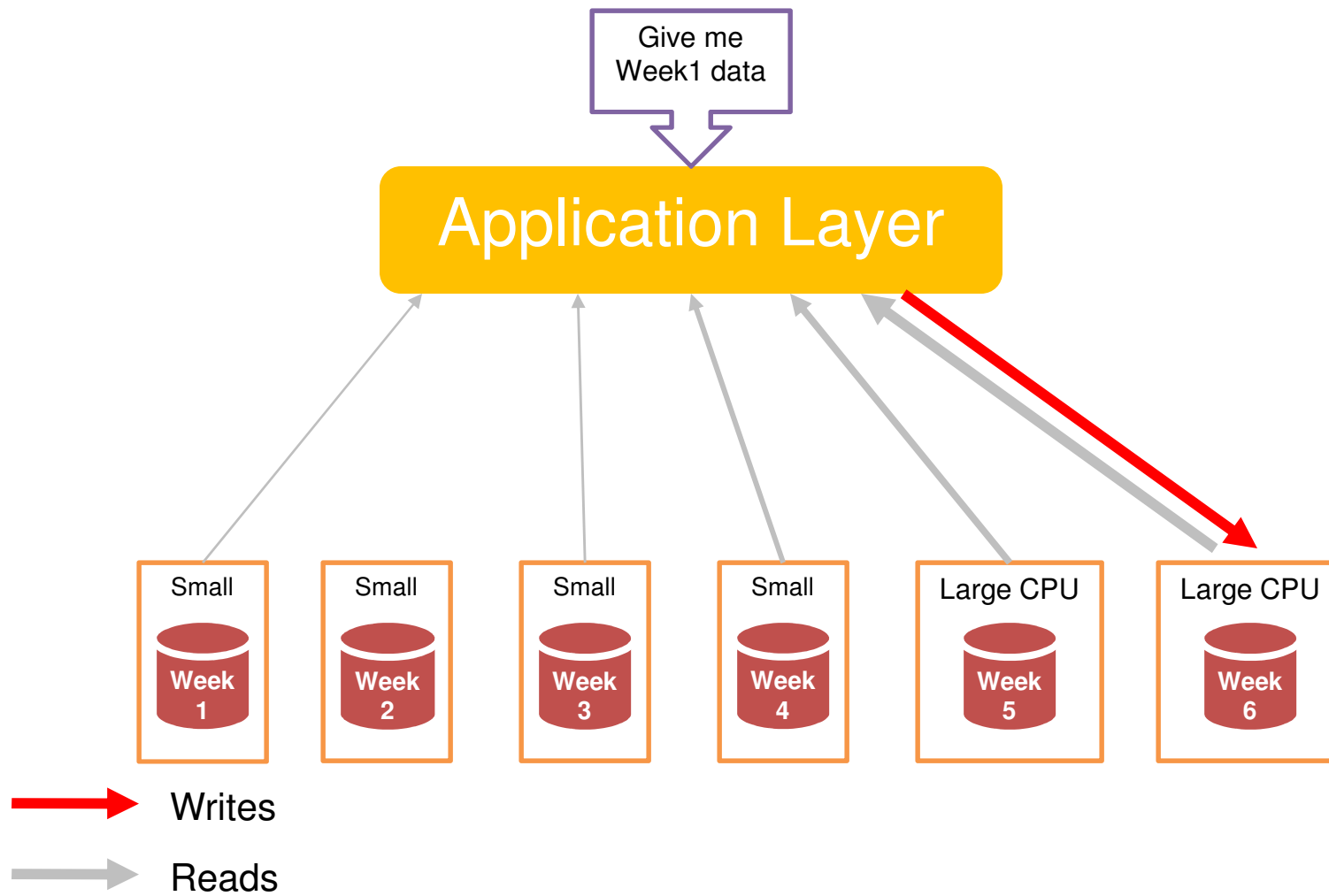


**Cost of  
Mistakes**



Dynamism of the cloud makes its easy





# Cloud-powered Continuous Delivery



**Continuous  
Integration**



**Continuous  
Deployment**



**Continuous  
Optimization**

# Cloud-powered Continuous Optimization



**Continuous  
Optimization**

Goal: Optimize on multiple dimensions : Cost, Performance, HA, Security, Response Time

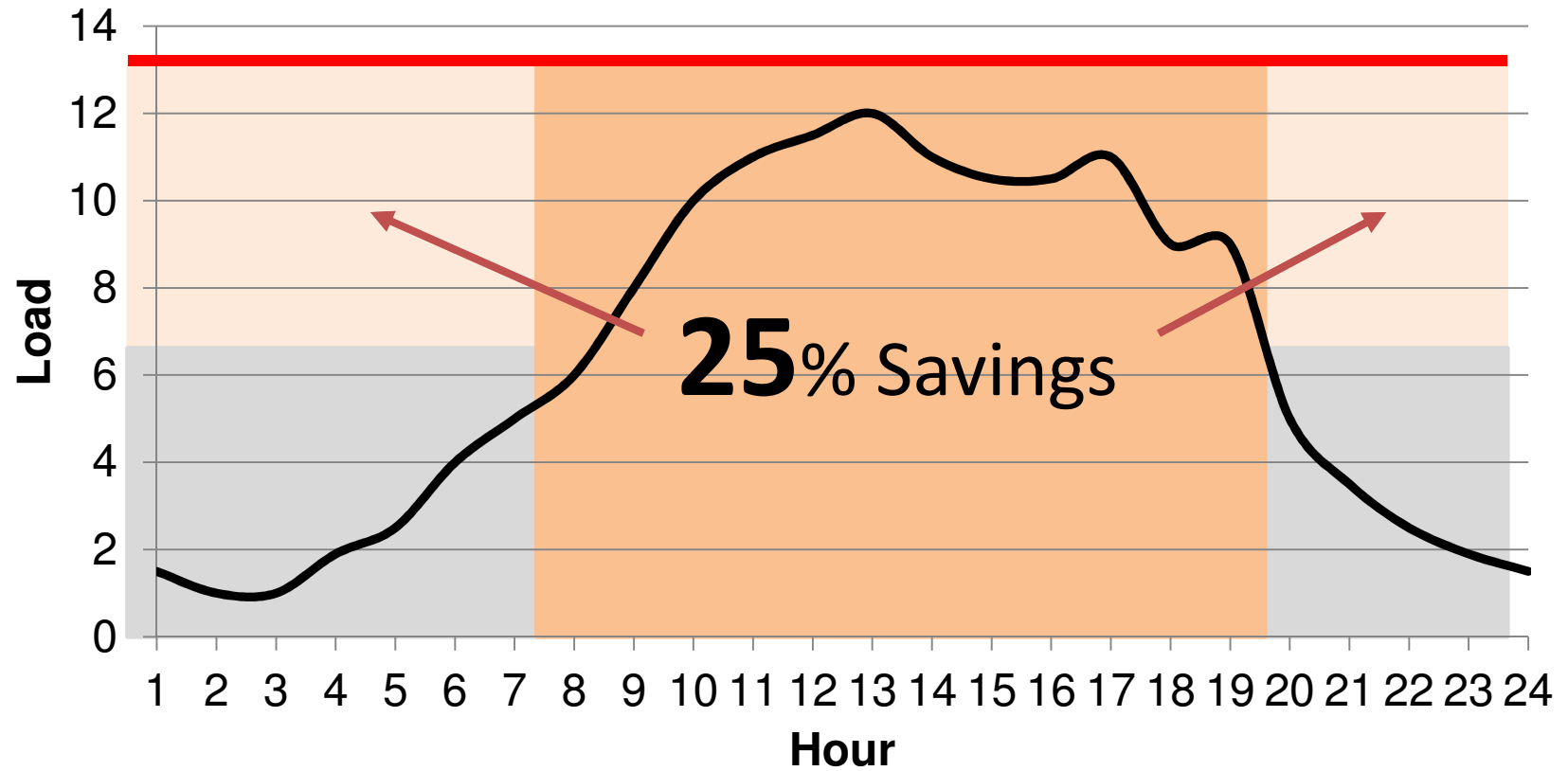
Benefit on optimize for cost - Immediate recurring cost savings

Metric: understanding utilization



When you turn off your cloud resources,  
you actually **stop paying for them**

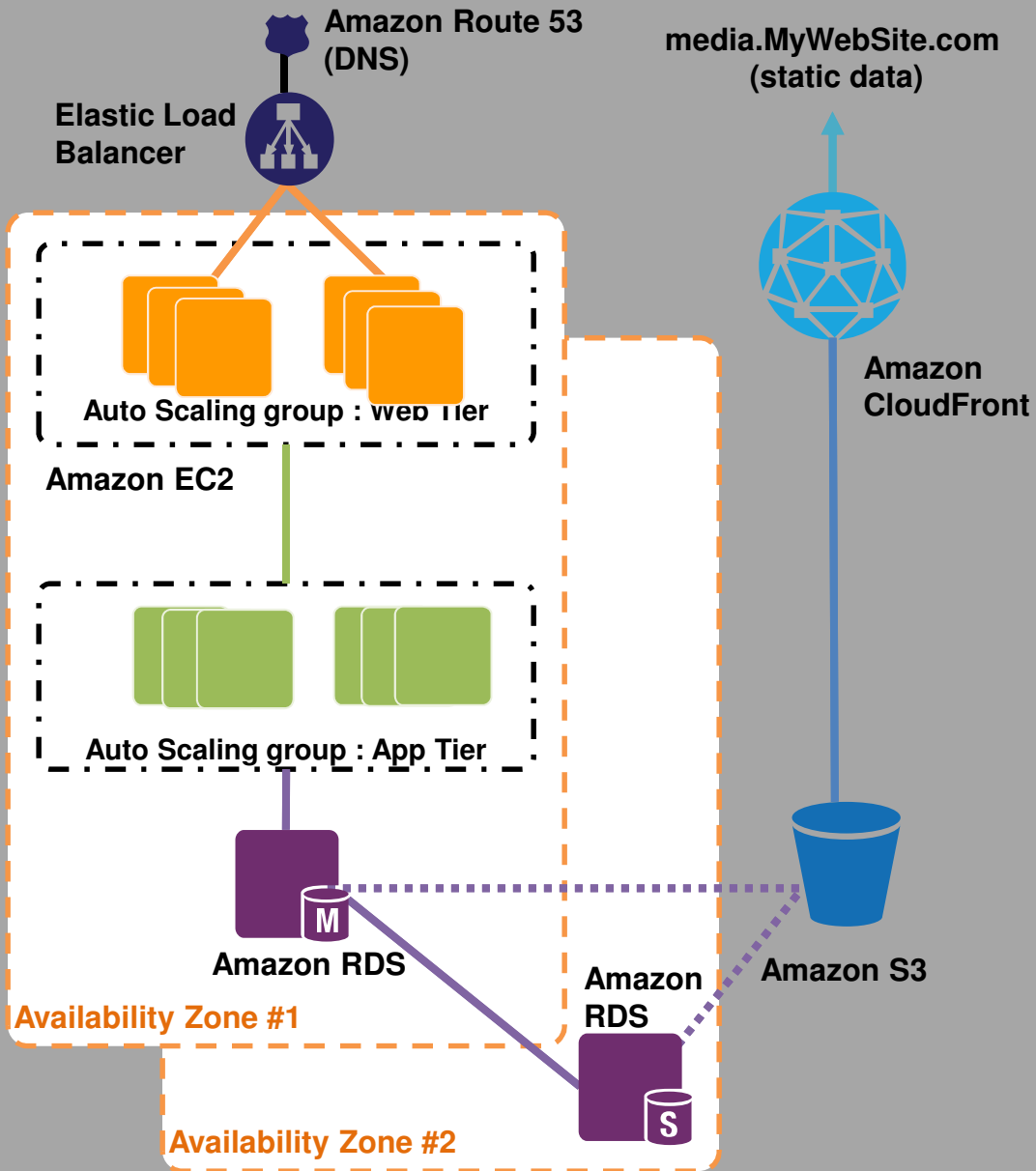
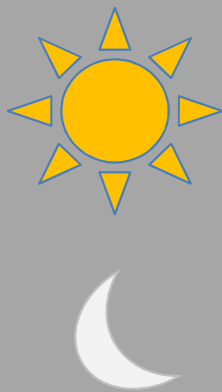
## Daily CPU Load



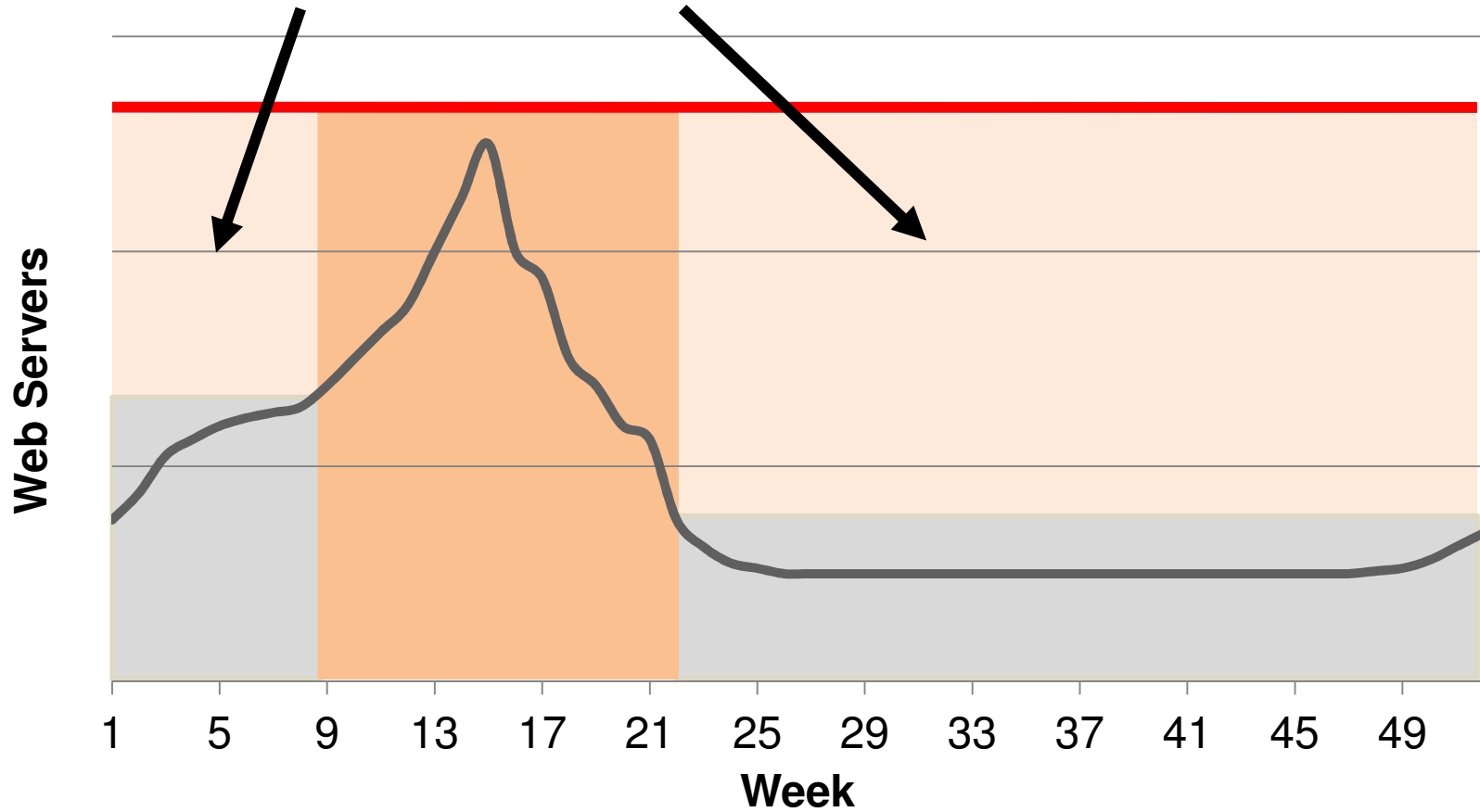
By the time of day

www.MyWebSite.com  
(dynamic data)

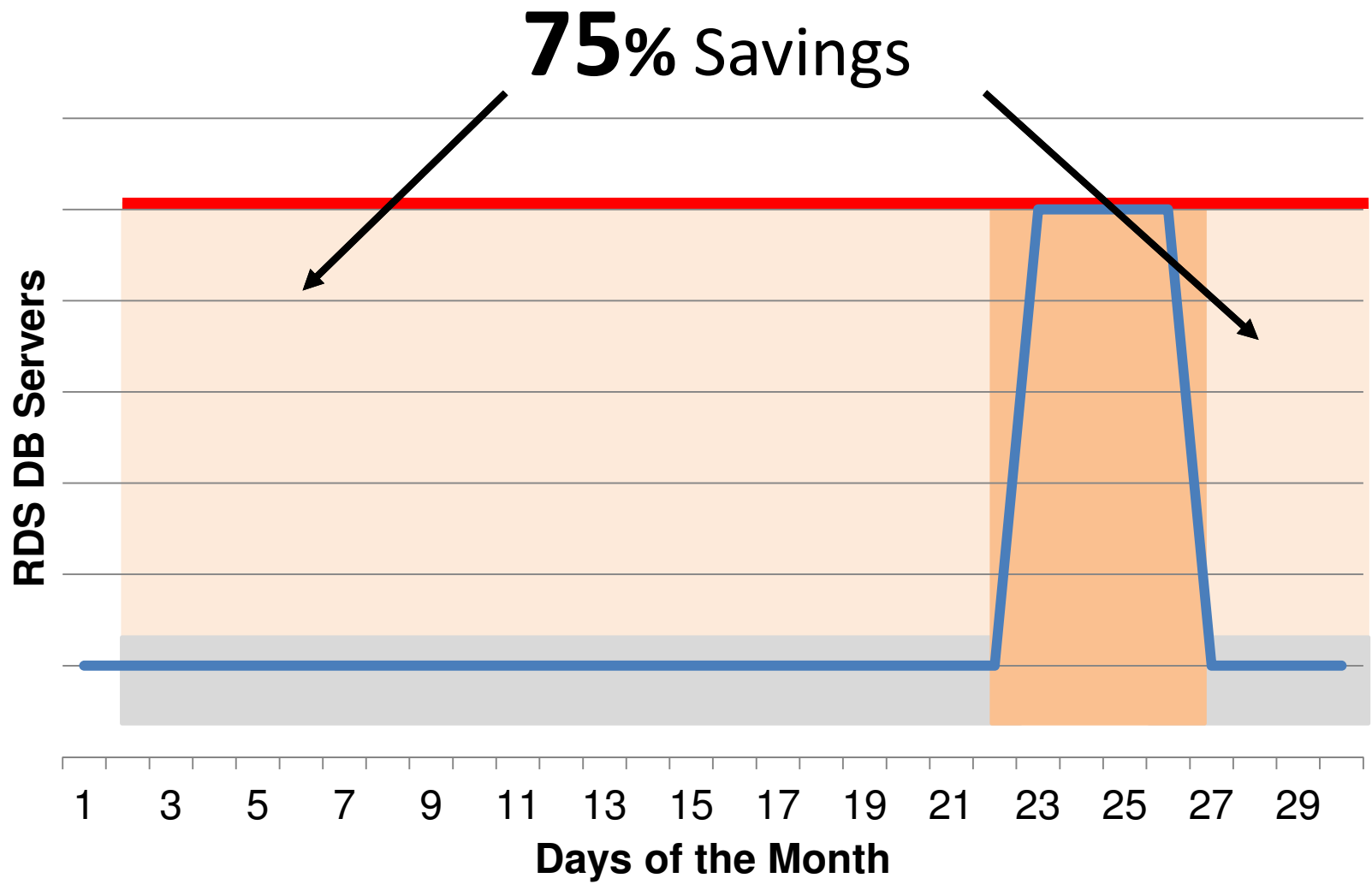
media.MyWebSite.com  
(static data)



# 50% Savings



during a year



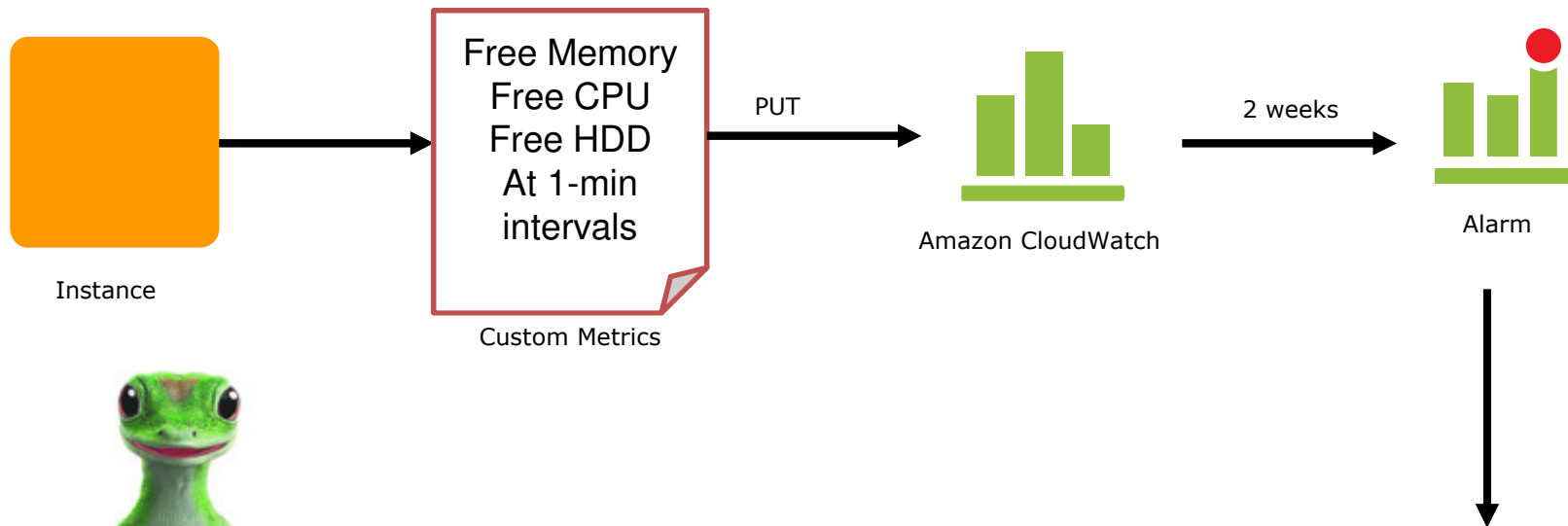
during a month



**Continuous optimization in  
your architecture and  
cloud infrastructure results  
in recurring savings in  
your next month's bill**



# Cost-aware instances



“You could save a bunch of money by switching to a **small instance**, Click on [CloudFormation Script](#) to Save”

# Choosing the right pricing model



On-demand Instances  
Reserved Instances  
Spot Instances

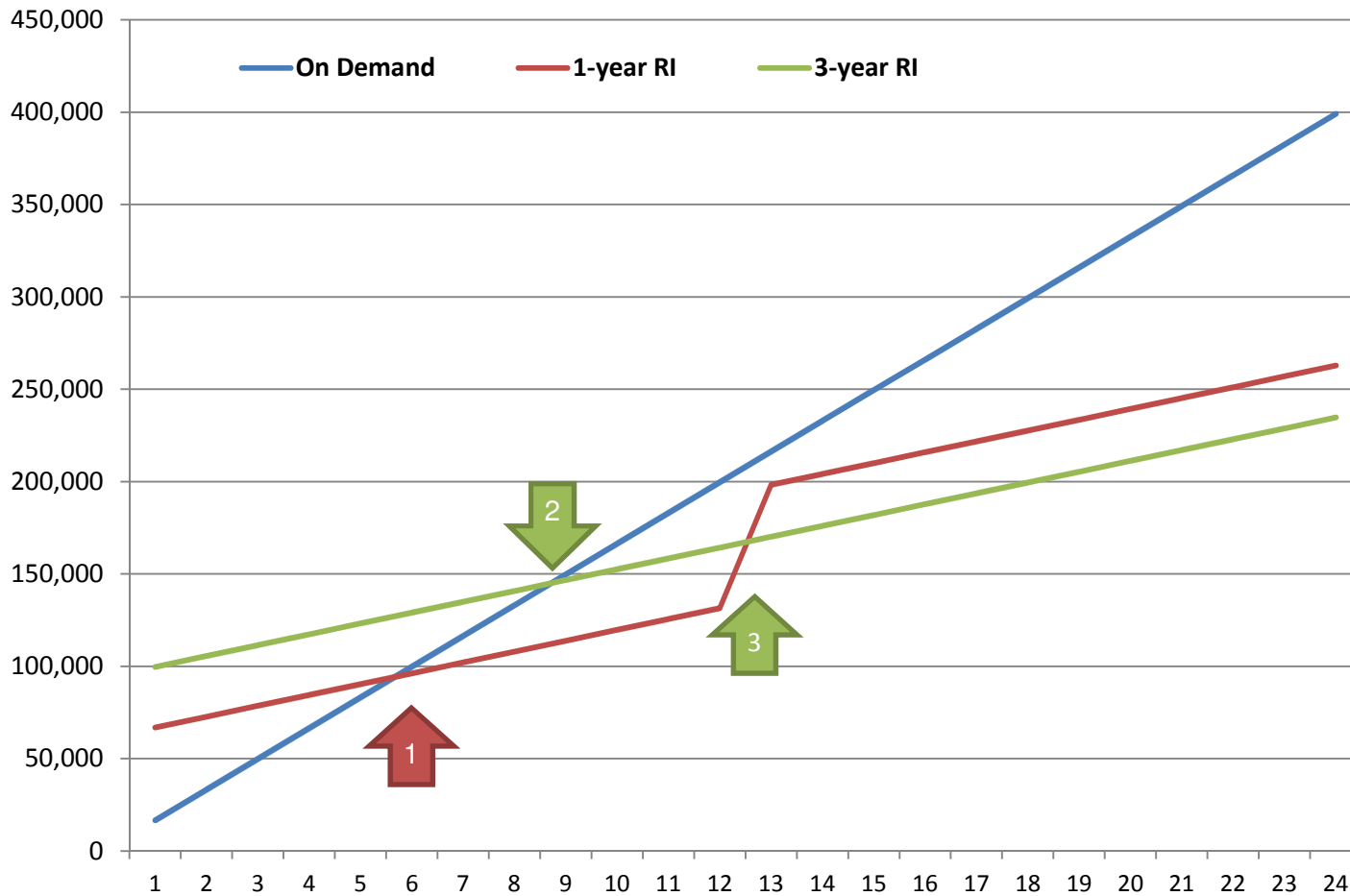
# Steady State Usage




## Total Cost for 1 Year-term of 2 application servers

	Usage Fee	One-time Fee	Total	Savings
<b>Option 1</b> On-Demand only	\$1493	-	\$1493	-
<b>Option 2</b> On-Demand + Reserved	\$1008	\$227	\$1234	~ <b>20%</b>
<b>Option 3</b> All reserved	\$528	\$455	\$983	~ <b>35%</b>

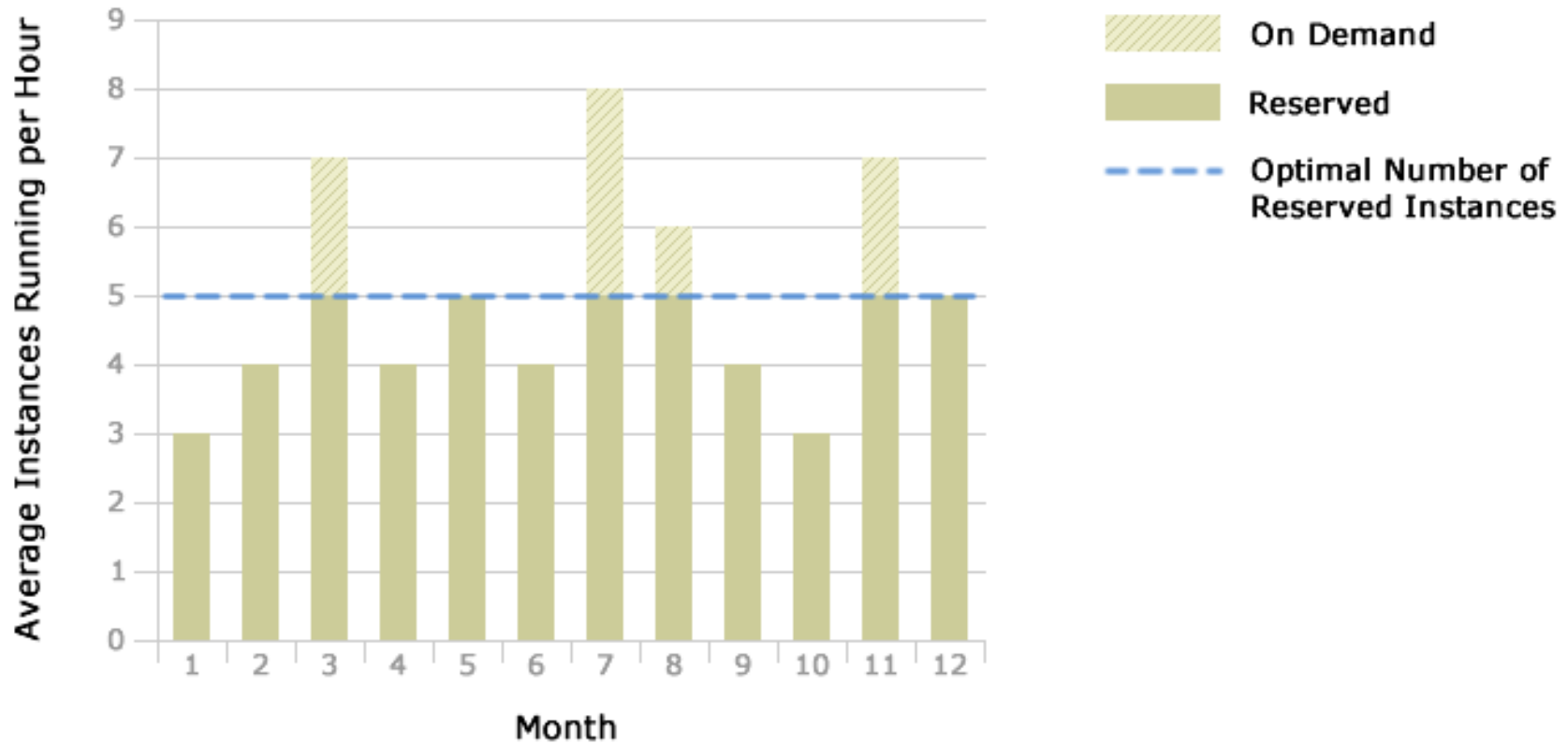
## Total Cost for 3 Year-term of the same 2 application servers

	Usage Fee	One-time Fee	Total	Savings
<b>Option 1</b> On-Demand only	\$4479	-	\$4479	-
<b>Option 2</b> On-Demand + Reserved	\$3024	\$350	\$3374	~ <b>30%</b>
<b>Option 3</b> All reserved	\$1584	\$700	\$2284	~ <b>50%</b>



- 
 1-year RI versus On Demand:  
 cost savings realized after first 6 months of usage
- 
 3-year RI versus On Demand:  
 cost savings realized after first 9 months of usage.
- 
 3-year RI versus 1-year RI:  
 Net savings of 3-year RI versus 1-year RI begin by month 13 and continue throughout the RI term (additional 23 months of savings)

# Common Pattern: Reserved + On-Demand



**Elasticity** is one of the  
fundamental properties of the cloud  
that drives many of its economic  
benefits

# Cloud-powered Continuous Delivery



**Continuous  
Integration**



**Continuous  
Deployment**



**Continuous  
Optimization**



# Summary



Invest in areas where you get to learn from your customers quickly

Automate everything else

Cloud Continuous Integration

- Release early, Release often, Iterate Quickly

- Get fast feedback

- Distributed Builds, Automated Tests in Parallel

Cloud Continuous Deployment

- Reduce the cost of mistakes

- Increase the speed of iteration

- Leverage cloud for Blue Green Deployments

Cloud Continuous Optimization

- Keep Optimizing and further reduce costs of infrastructure



**Thank you!**

**Jinesh Varia**

[jvaria@amazon.com](mailto:jvaria@amazon.com)

Twitter: [@jinman](https://twitter.com/@jinman)

Credits to Jon Jenkins, Paul Duvall and several engineers at Amazon

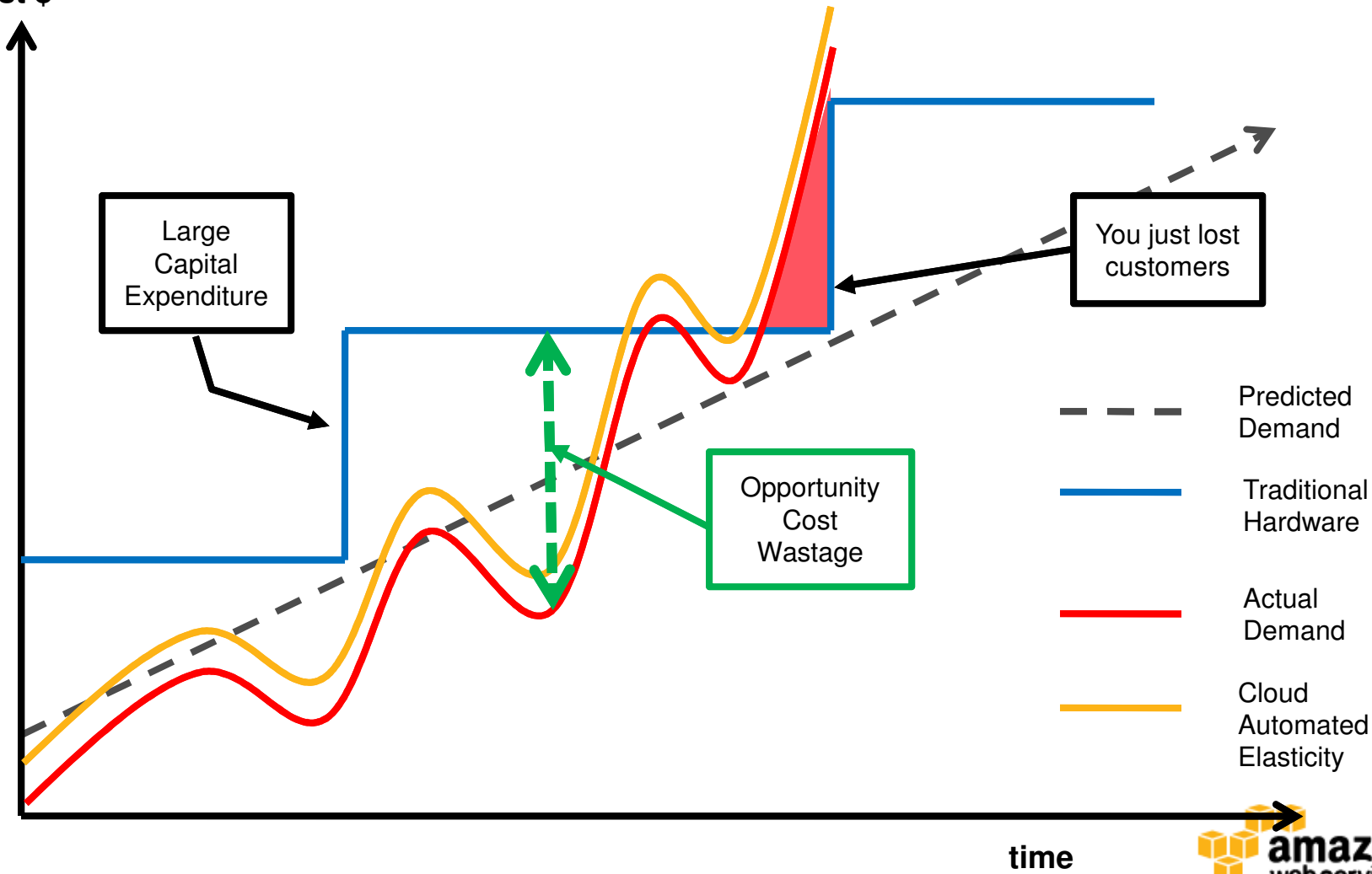


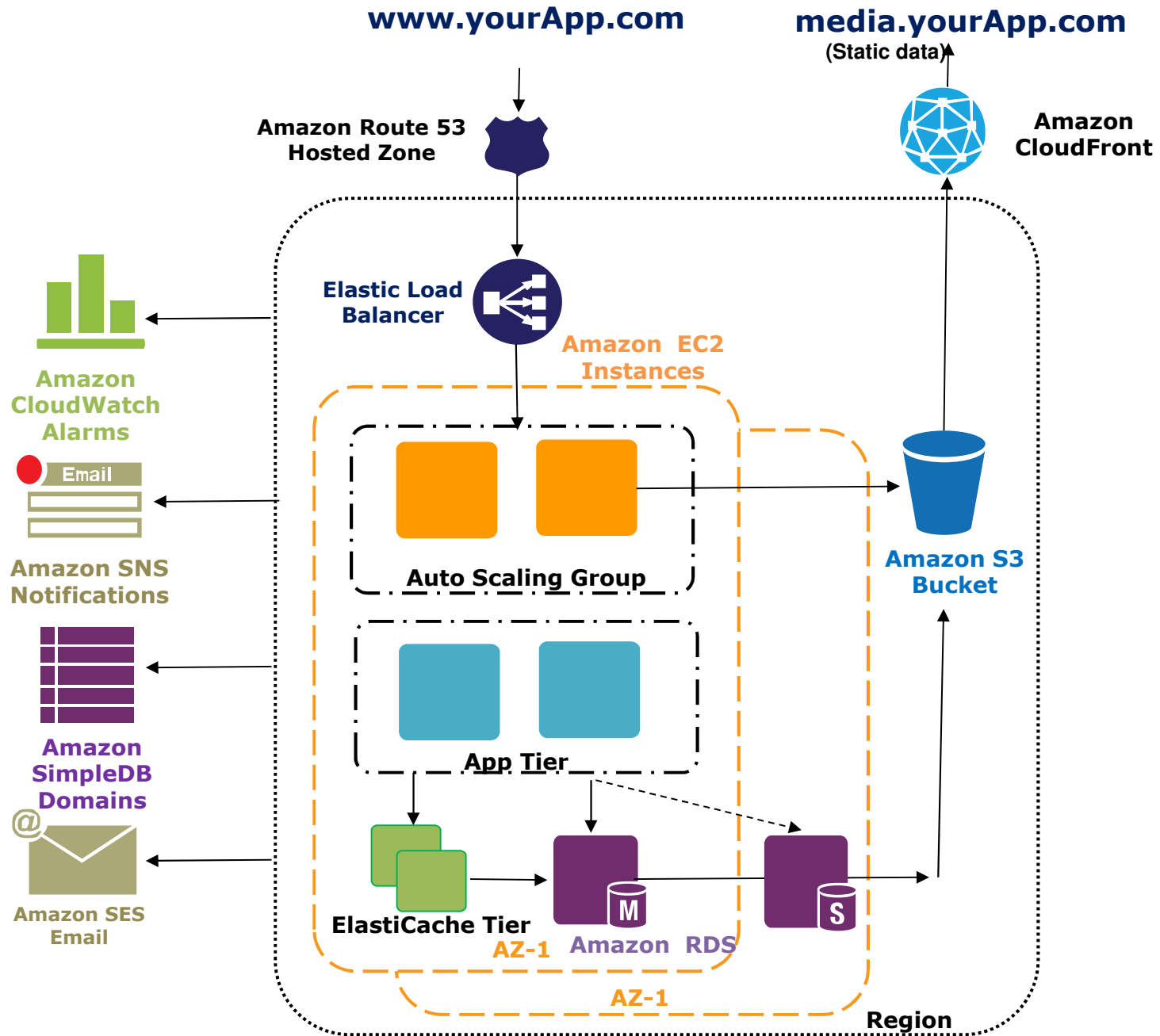
**amazon**  
**web services™**

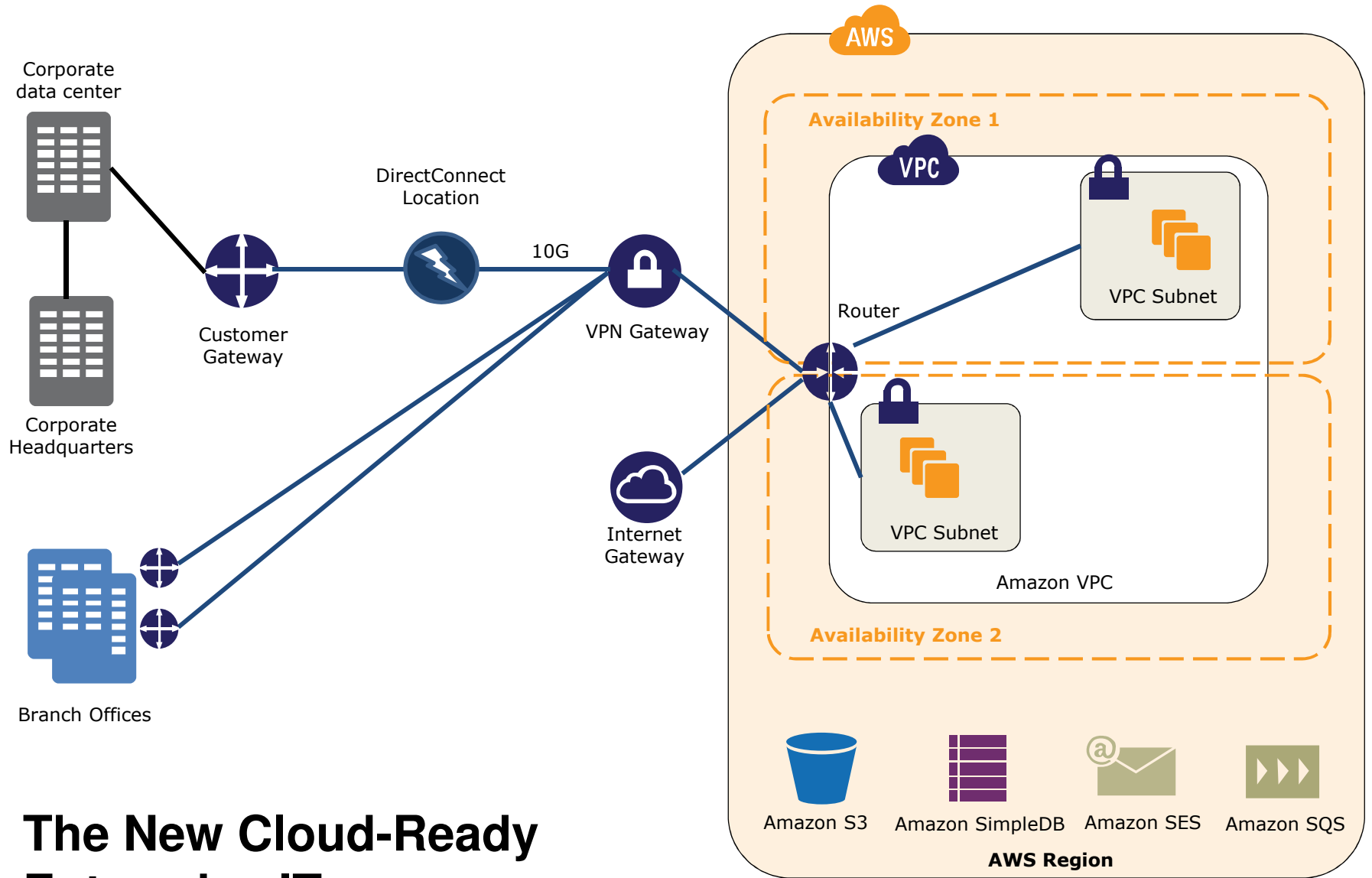
<http://aws.amazon.com>

# Optimize by Implementing Elasticity

Infrastructure  
Cost \$







# The New Cloud-Ready Enterprise IT