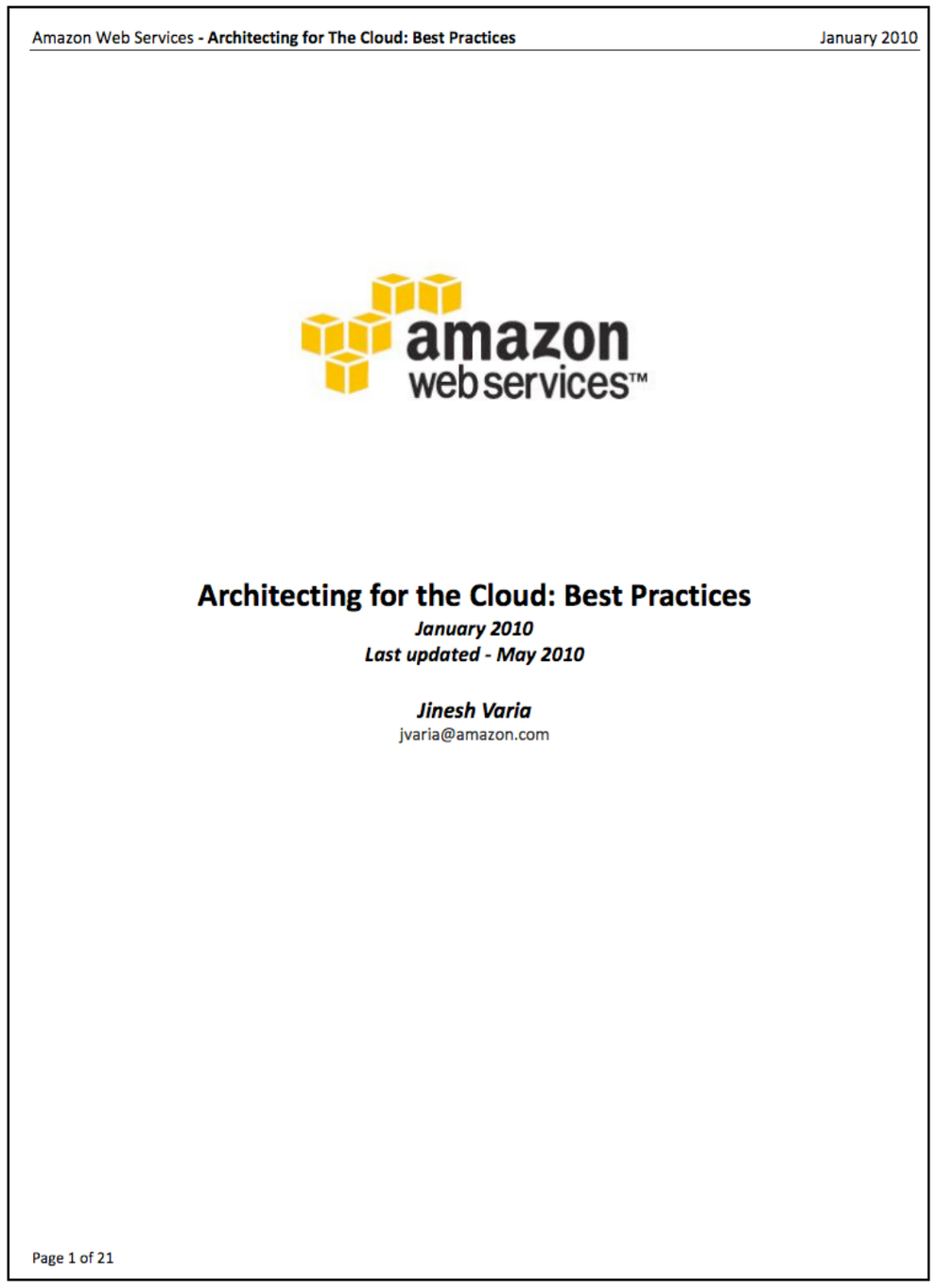
Design Patterns for the Cloud

based on

Amazon Web Services

Architecting for the Cloud: Best Practices

Jinesh Varia


Amazon Web Services - **Architecting for The Cloud: Best Practices** January 2010

amazon web services™

Architecting for the Cloud: Best Practices

January 2010

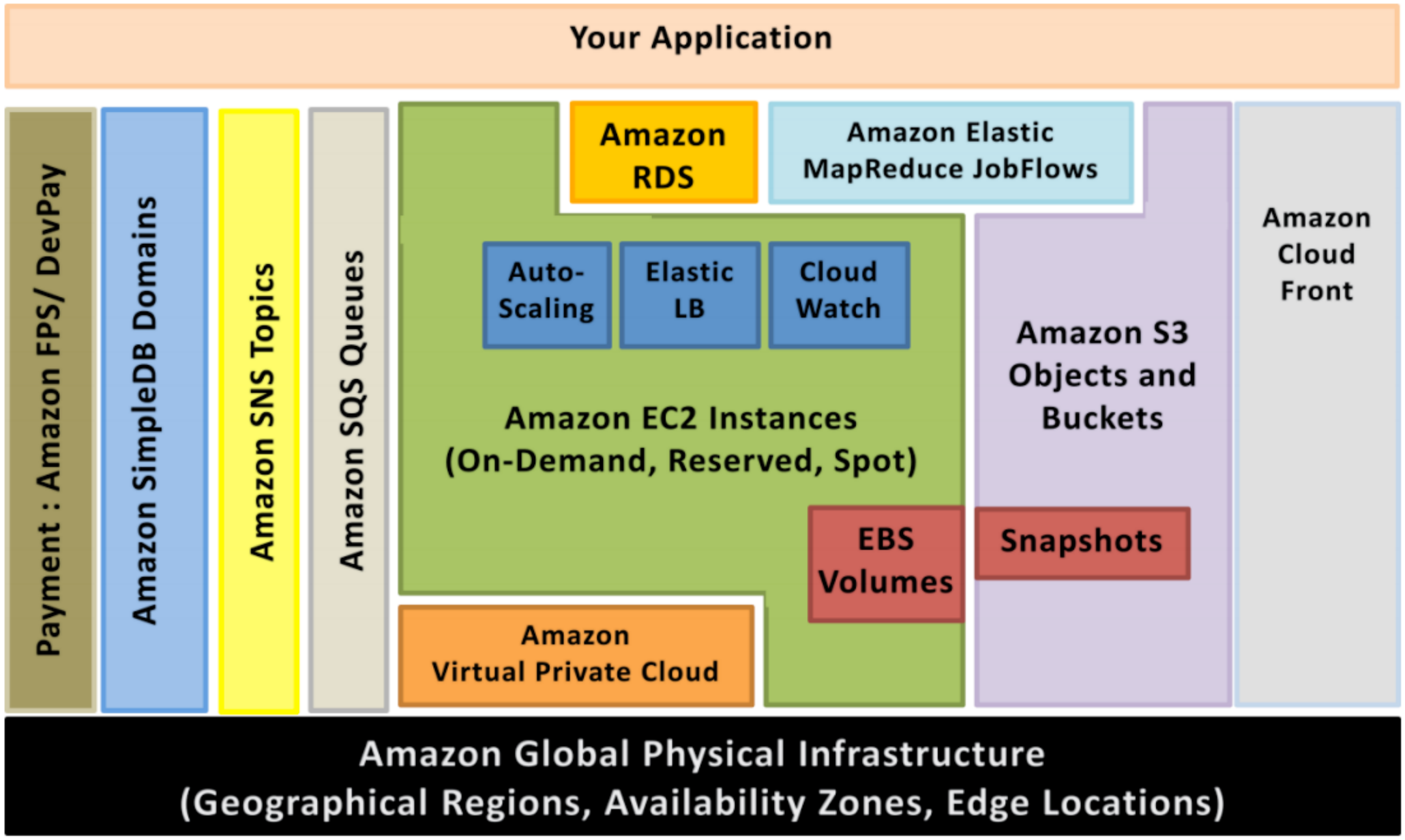
Last updated - May 2010

Jinesh Varia
jvaria@amazon.com

Page 1 of 21

http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

Amazon Web Services



Amazon Web Services

amazon web services

Database

DynamoDB
Predictable and Scalable NoSQL Data Store

ElastiCache
In-Memory Cache

RDS
Managed Relational Database

Redshift
Managed Petabyte-Scale Data Warehouse

Storage and Content Delivery

S3
Scalable Storage in the Cloud

EBS
Networked Attached Block Device

CloudFront
Global Content Delivery Network

Glacier
Archive Storage in the Cloud

Storage Gateway
Integrates On-Premises IT with Cloud Storage

Import Export
Ship Large Datasets

Cross-Service

Support
Phone & email fast-response 24X7 Support

Marketplace
Bull and Sell Software and Apps

Management Console
UI to manage AWS services

SDKs, IDE kits and CLIs
Develop , integrate and manage services

Compute & Networking

EC2
Virtual Servers in the Cloud

VPC
Virtual Secure Network

ELB
Load balancing Service

Auto Scaling
Automatically scale up and down

Elastic MapReduce
Managed Hadoop Framework

Direct Connect
Dedicated Network Connection to AWS

Route 53
Scalable Domain Name System

Deployment & Management

CloudFormation
Templated AWS Resource Creation

CloudWatch
Resource and Application Monitoring

Data Pipeline
Orchestration for Data-Driven Workflows

Elastic Beanstalk
AWS Application Container

IAM
Secure AWS Access Control

OpsWorks
DevOps Application Management Service

CloudHSM
Hardware-based key storage for compliance

App Services

CloudSearch
Managed Search Service

Elastic Transcoder
Easy-to-use Scalable Media Transcoding

SES
Email Sending Service

SNS
Push Notification Service

SQS
Message Queue Service

SWF
Workflow Service for Coordinating App Components

AWS Global Physical Infrastructure
(Geographical Regions, Availability Zones, Edge Locations)

Scalable Architectures

A **scalable architecture** is critical to take advantage of a **scalable infrastructure**

The cloud is designed to provide conceptually **unlimited scalability**.

Characteristics of Truly Scalable Service

- Increasing **resources** results in a **proportional** increase in **performance**
- A scalable service is capable of **handling heterogeneity**
- A scalable service is **operationally efficient**
- A scalable service is **resilient**
- A scalable service becomes more **cost effective** when it grows

1. Design for Failure

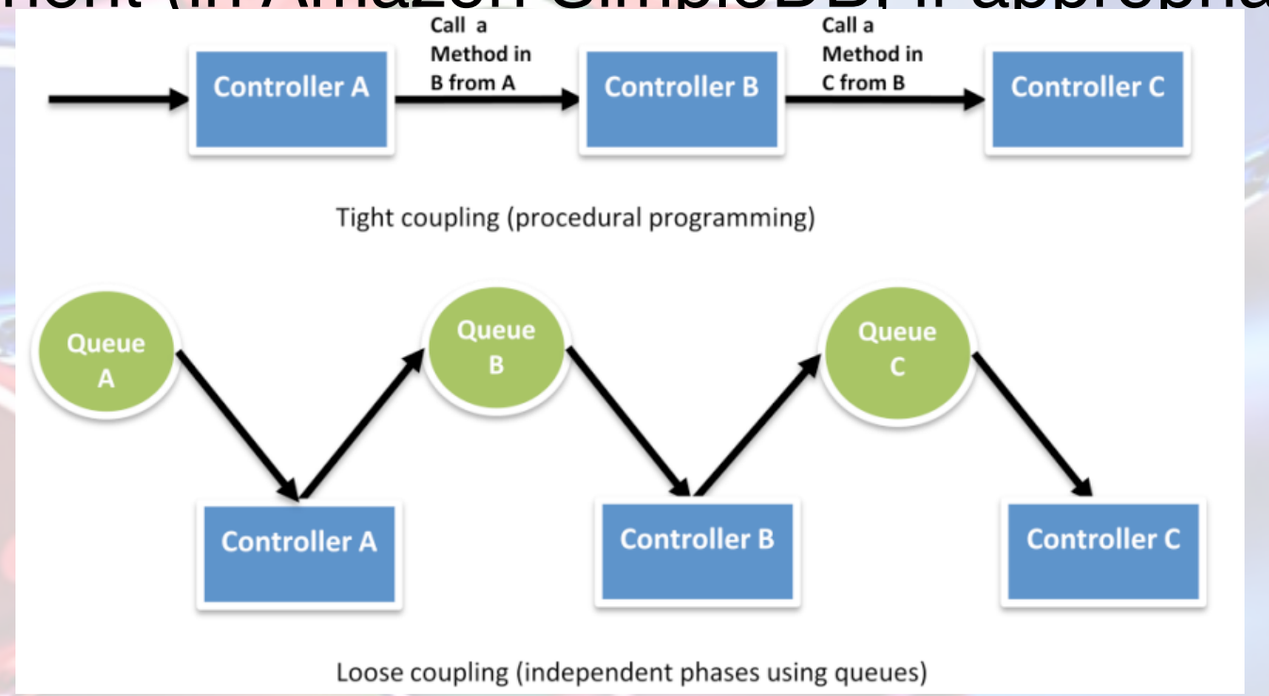
- “Everything fails, all then time” - Werner Vogels, Amazon’s CTO
- Avoid single points of failure
- Assume everything fails, and design backwards
- Goal: Applications should continue to function even if the underlying physical hardware fails or is removed or replicated
- The following strategies can help in event of failure:
  1. Have a coherent backup and restore strategy for your data and automate it
  2. Build process threads that resume on reboot
  3. Allow the state of the system to re-sync by reloading messages from queues
  4. Keep pre-configured and pre-optimized virtual images to support (2) and (3) on launch/boot
  5. Avoid in-memory sessions or stateful user context, move that to data stores.

2. Design Loosely Coupled Systems

- The cloud reinforces the SOA design principle that **the more loosely coupled the components of the system, the bigger and better it scales**.
- Build components that **do not have tight dependencies** on each other.
- Build **asynchronous systems** and scaling horizontally become very important in the context of the cloud.
- Build systems to **scale out** by adding more instances of same component

2. AWS Tactics

1. Use **Amazon SQS** as buffers between components
2. Design every component such that it expose a **service interface** and is **responsible for its own scalability** in all appropriate dimensions and **interacts** with other components **asynchronously**
3. Bundle the logical construct of a component into an **Amazon Machine Image** so that it can be deployed more often
4. Make your applications **as stateless as possible**. Store session state outside of component (in Amazon SimpleDB, if appropriate)



3. Implement Elasticity

- Elasticity can be implemented in three ways:
  1. **Proactive Cyclic Scaling**: Periodic scaling that occurs at fixed interval (daily, weekly, monthly, quarterly)
  2. **Proactive Event-based Scaling**: Scaling just when you are expecting a big surge of traffic requests due to a scheduled business event (new product launch, marketing campaigns)
  3. **Auto-scaling based on demand**. By using a monitoring service, your system can send triggers to take appropriate actions so that it scales up or down based on metrics (utilization of the servers or network i/o, for instance)
- To implement “Elasticity”, one has to first **automate the deployment process** and **streamline the configuration and build process**. This will ensure that the system can scale without any human intervention.

3. Design your AMI

- The cloud allows you to automate your deployment process.
- Take the time to create an automated deployment process early on during the migration process and not wait till the end.
- Creating an automated and repeatable deployment process will help reduce errors and facilitate an efficient and scalable update process.

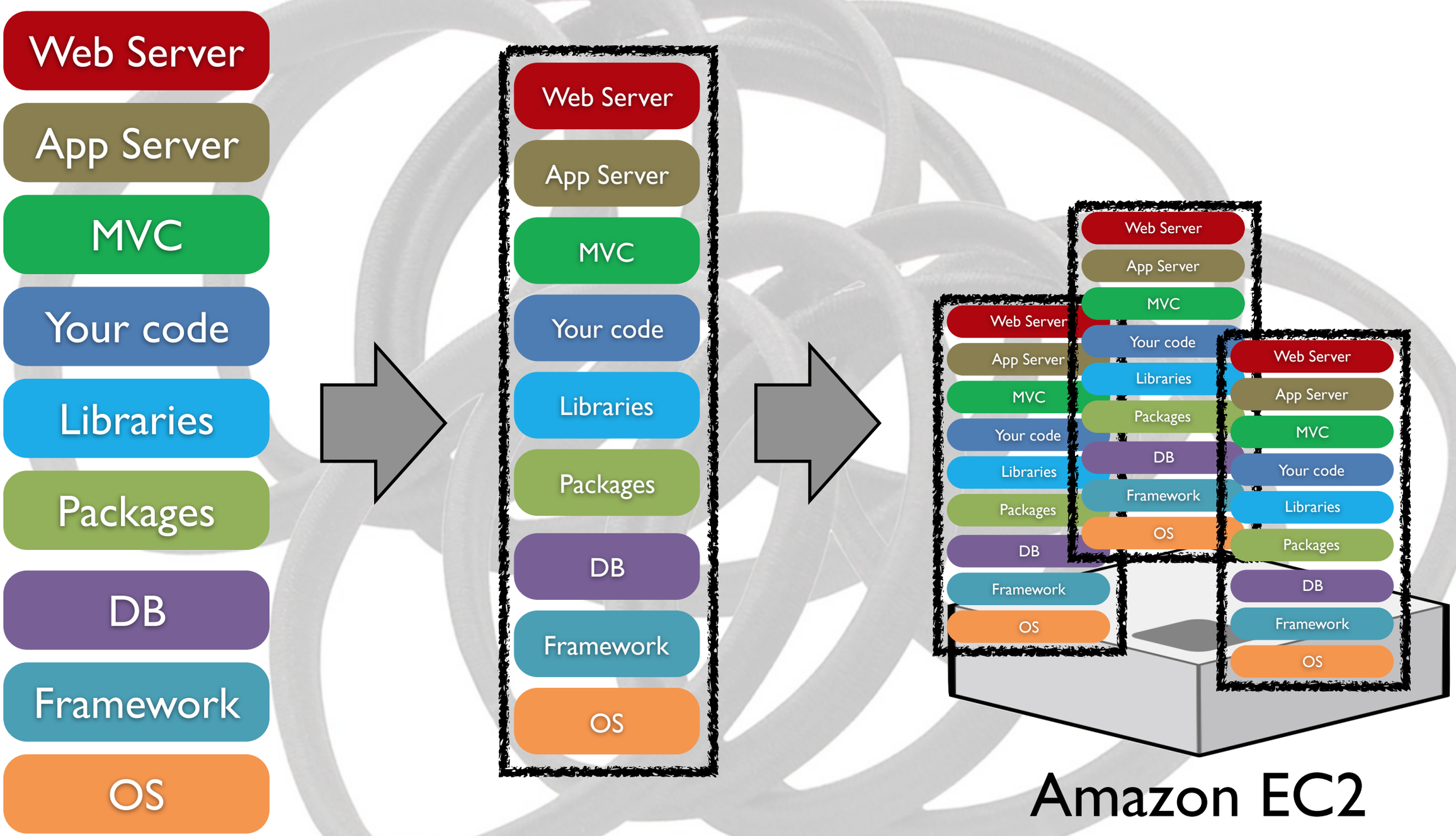
- To automate the deployment process:
  - Create a **library of "recipes"** – small frequently-used scripts (for installation and configuration)
  - Manage the configuration and deployment process using **agents bundled inside an AMI**

3. AMI Design Approaches

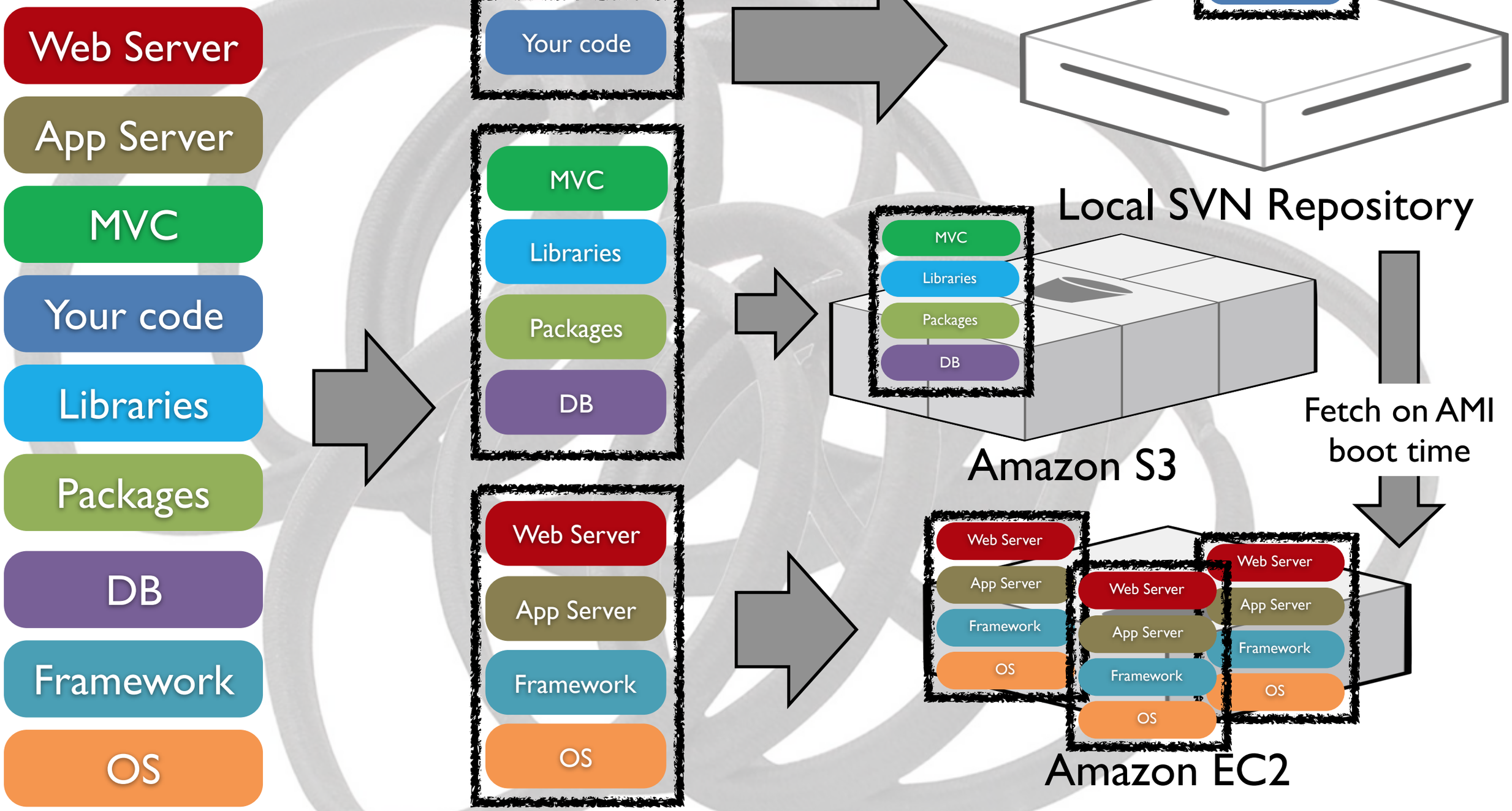
1. **Inventory of static AMIs**

2. **Golden AMIs with fetch on boot**

3. Inventory of static AMIs



3. Golden AMIs with fetch on boot



3. AMI with JeOS and agent

Web Server

App Server

MVC

Your code

Libraries

Packages

DB

Framework

OS

Your code

Web Server

App Server

MVC

Libraries

Packages

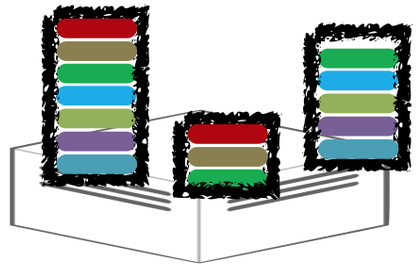
DB

Framework

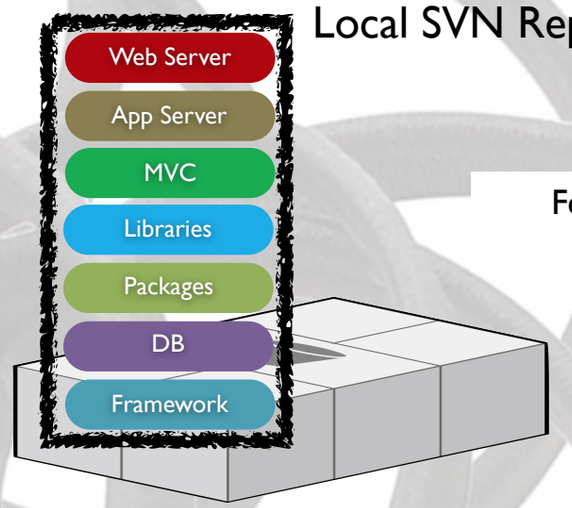
Agent

OS

Local SVN Repository



Recipe Database



Amazon S3

Fetch on AMI boot time

Download recipe on agent request

Amazon EC2

4. Think Parallel

- **Serial** and **Sequential** is now history
- The cloud is designed to handle massively parallel operations when it comes to accessing (retrieving and storing) data: **leverage request parallelization**
- **Multi-threading** your requests by using multiple concurrent threads
- The processes of a cloud application should be made **thread-safe** through a **share-nothing** philosophy
- **Distribute** the incoming requests across **multiple asynchronous** web servers using **load balancer**

5. Leverage Storage Options

- In the cloud, you are **paying for bandwidth** in and out of the cloud
- Transfer and the cost can add up very quickly.
- Keep **dynamic data** closer to the compute element
- Keep **static data** closer to the end-user
- If a large quantity of data that needs to be processed resides outside of the cloud, use **Sneakernet :-)**
- If the data is static and not going to change often (for example, images, video, audio, PDFs, JS, CSS files), it is advisable to take advantage of a **content delivery service** so that the static data is cached at an edge location closer to the end-user (requester) thereby lowering the access latency.

5. AWS Tactics

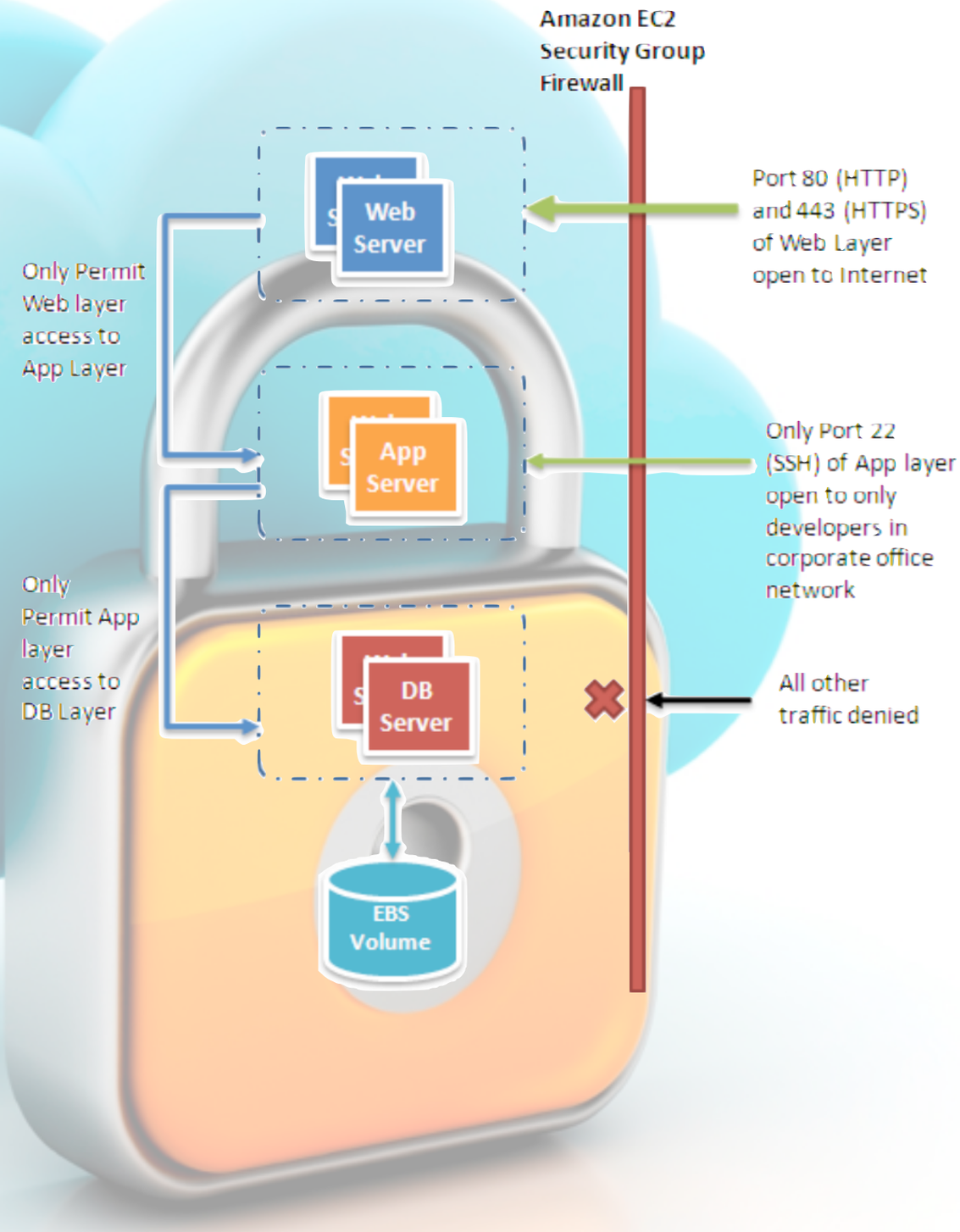
	Amazon S3 + CF	Amazon EC2 Ephemeral Store	Amazon EBS	Amazon SimpleDB	Amazon RDS
Ideal for	Storing Large write-once, read-many types of objects, Static Content Distribution	Storing non-persistent transient updates	Off-instance persistent storage for any kind of data,	Querying light-weight attribute data	Storing and querying structured Relational and referential Data
Ideal examples	Media files, audio, video, images, Backups, archives, versioning	Config Data, scratch files, TempDB	Clusters, boot data, Log or data of commercial RDBMS like Oracle, DB2	Querying, Mapping, tagging, click-stream logs, metadata, shared-state management, indexing	Complex transactional systems, inventory management and order fulfillment systems
Not recommended for	Querying, Searching	Storing Database logs or backups, customer data		Relational (joins) query	
Not recommended examples	Database, File Systems	Sensitive data	Content Distribution	OLTP, DW cube rollups	Simple lookups

6. Security

- In the cloud, **security** should be implemented **in every layer** of the cloud application architecture
- **Physical security** is typically handled by your service provider
- **Network** and **application-level security** is your responsibility
- Protect your data **in transit**
- Protect your data **at rest**
- Protect your **AWS credentials**
- Manage multiple Users and their permissions with **IAM**

6. AWS Tactics

- Every Amazon EC2 instance is protected by one or more **security groups**
- **Named sets of rules** that specify which ingress (i.e., incoming) network traffic should be delivered to your instance.
- You can specify TCP and UDP ports, ICMP types and codes, and source addresses.
- Security groups give you basic **firewall-like protection** for running instances.



Web Application Hosting Example

Traditional Architecture

Exterior Firewall
Hardware or Software Solution to open standard ports (80, 443)

Web Load Balancer
Hardware or Software solution to distribute traffic over web servers

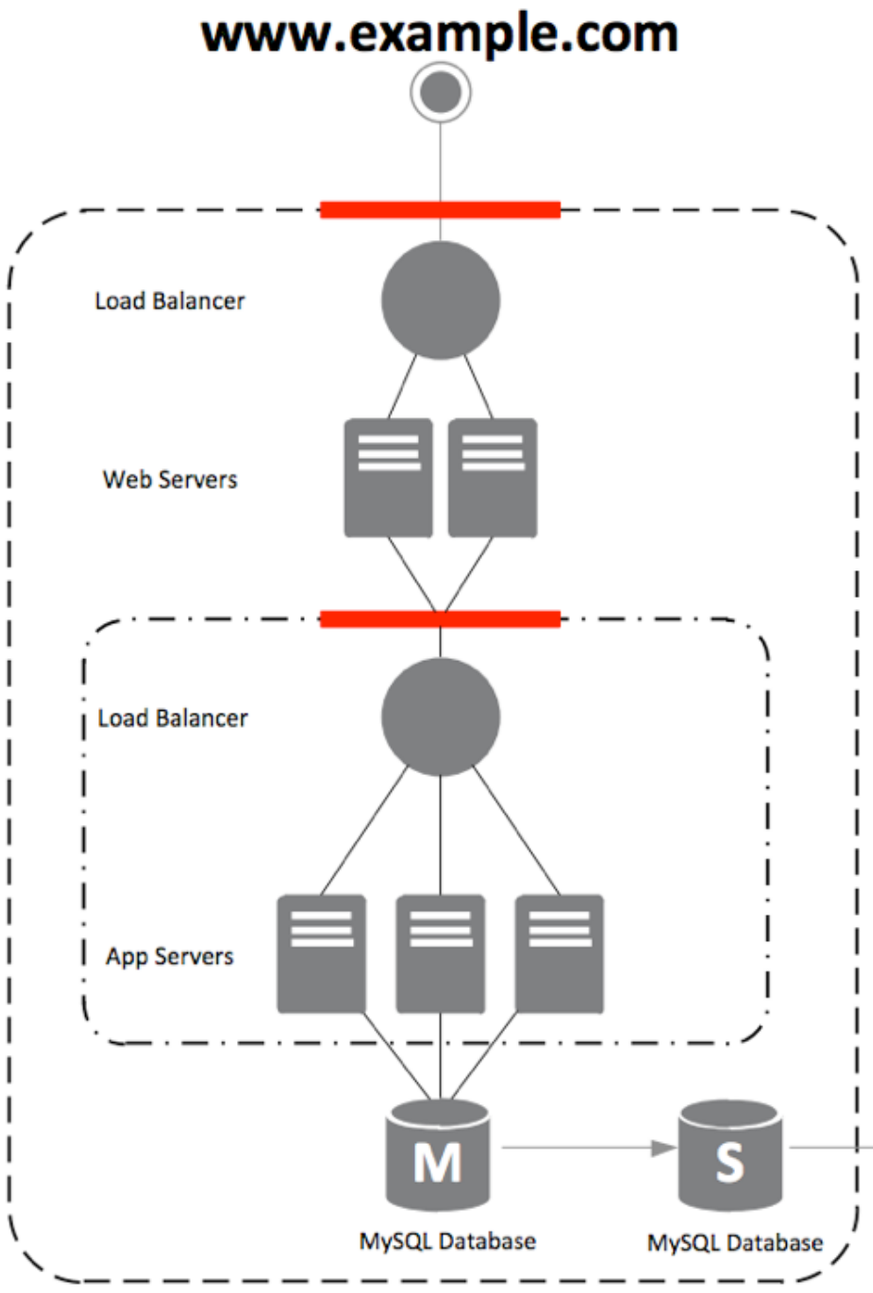
Web Tier
Fleet of machines handling HTTP requests

Backend Firewall limits access to application tied from web tier

App Load Balancer
Hardware or Software solution to spread traffic over app servers

App Server Tier
Fleet of machines handling Application specific workloads
Caching server machines can be implemented at this layer

Data Tier
Database Server machines with master and local running separately, Network storage for static objects



taken from: http://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf

Web Application Hosting Example

Amazon WS Architecture

Route 53
Provides DNS services to simplify domain management and zone APEX support (http://example.com)

Elastic Load Balancer
ELB to spread traffic to Web Server Autoscaling groups

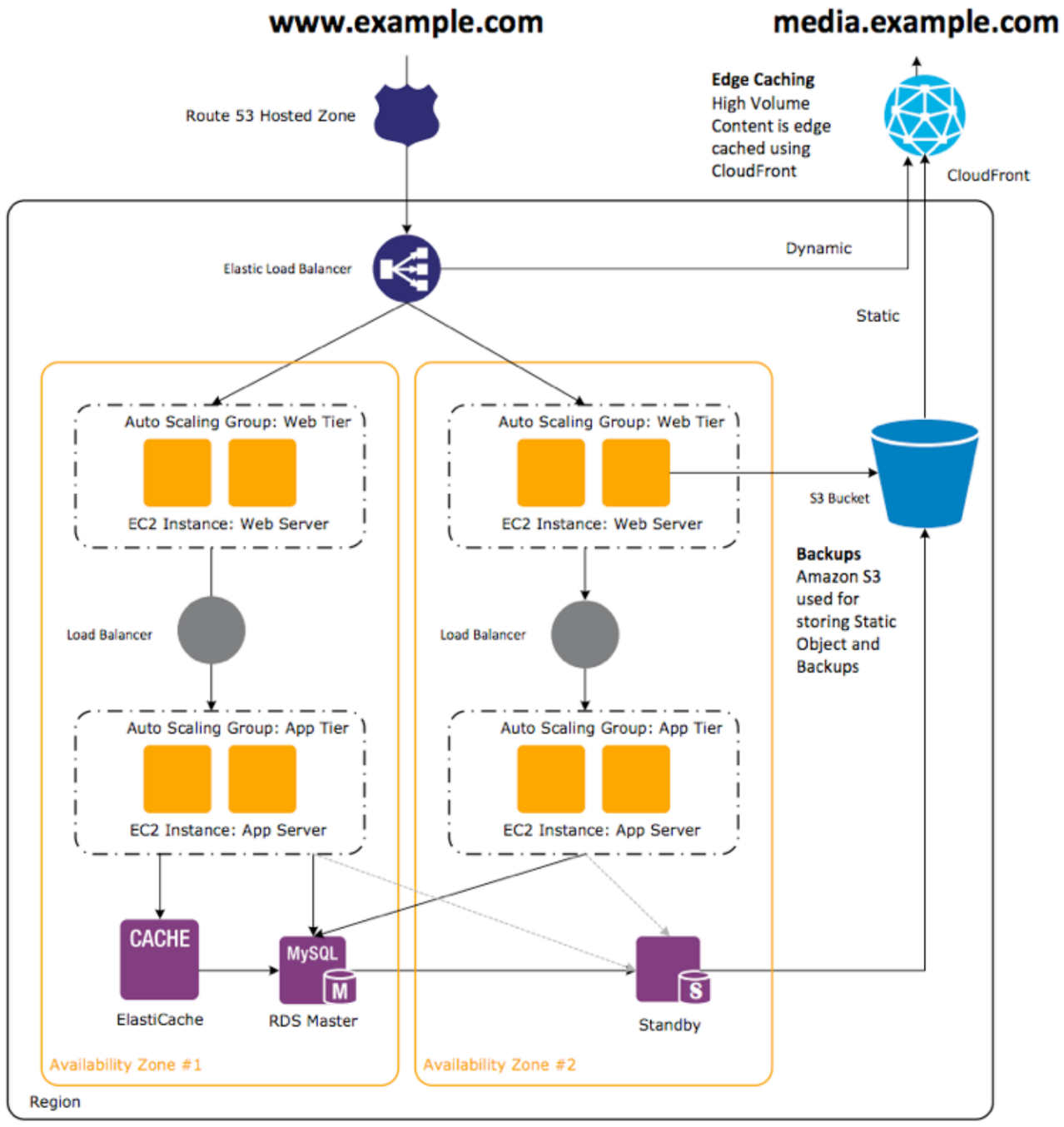
Exterior Firewall moved to every Web Server instance via Security Groups
Auto Scaling Web Tier
Group of EC2 instances handling HTTP requests.
Backend Firewall moved to every back-end instance

App Server Load Balancer
Software LB (e.g. HAProxy) on EC2 instances to spread traffic over app server cluster

Auto-scaling App Tier
Group of EC2 instances running the actual app. Instances belong to Auto Scaling group

ElastiCache
Provides caching services for app, removing load from database tier

DB Tier
MySQL RDS DB creates a highly available architecture with multi-AZ deployments. Read-only replicas can also be used to scale read intensive applications



taken from: http://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf

Top 10 Obstacles for Cloud Computing

- **Availability of a service:**
  - Organizations worry about whether Cloud services will have adequate availability
  - Very (very) high availability can be achieved by adopting multiple Cloud Computing providers
  - Even if the Cloud provider has multiple data centers in different geographic regions, it may have common software infrastructure and accounting systems, or the company may even go out of business
- **Data lock-in:**
  - Software stacks have improved interoperability among platforms, but
    the APIs for Cloud Computing itself are still essentially

Top 10 Obstacles for Cloud Computing

- **Data confidentiality and auditability:**
  - My sensitive corporate data will never be in the cloud
  - Current Cloud offerings are essentially public (rather than private) networks, exposing the system to more attacks
  - There are also requirements for auditability and privacy laws (many Nations have laws requiring SaaS providers to keep customer data and copyrighted material within national boundaries)
  - There are no fundamental obstacles to making a Cloud Computing environment as secure as the vast majority of in-house IT environments, well-understood technologies (e,g., encrypted storage, VPN, firewalls,...)
- **Data Transfer Bottlenecks:**
  - Applications continue to become more data-intensive, significant costs in the Cloud
  - Avoid Internet transfers by shipping disks
  - Data Storage for free, CPU cycles sustain the business
  - WAN and LAN bandwidth are still bottlenecks

Top 10 Obstacles for Cloud Computing

- **Performance unpredictability:**
  - Multiple Virtual Machines can share CPUs and main memory surprisingly well in Cloud Computing, but I/O sharing is more problematic
  - Improve architectures and operating systems to efficiently virtualize interrupts and I/O channels
  - Flash memory adoption will decrease I/O interference
- **Scalable storage:**
  - Short-term usage, no up-front cost, and infinite capacity on-demand is more difficult to achieve with persistent storage with respect to computation
- **Bugs in large-scale distributed systems:**
  - Removing errors in very large scale distributed systems is very challenging
  - A common occurrence is that bugs cannot be reproduced in smaller

Top 10 Obstacles for Cloud Computing

- **Scaling quickly:**
  - Pay-as-you-go certainly applies to storage and to network bandwidth, both of which count bytes used
  - Computation is slightly different, depending on the virtualization level:
    - Google AppEngine automatically scales in response to load increases and decreases, and users are charged by the cycles used
    - Amazon EC2 charges by the hour for the number of instances you occupy, even if your machine is idle
  - Automatically scale quickly up and down in response to load is important in order to save money, but without violating service level agreements
- **Reputation fate sharing:**
  - Reputations do not virtualize well. One customer's bad behavior can affect the reputation of the cloud as a whole
  - For instance, blacklisting of Amazon EC2 IP addresses by spam-prevention services may limit which applications can be effectively hosted
  - Legal issues
- **Software licensing:**
  - Current software licenses commonly restrict the computers on which the software can run