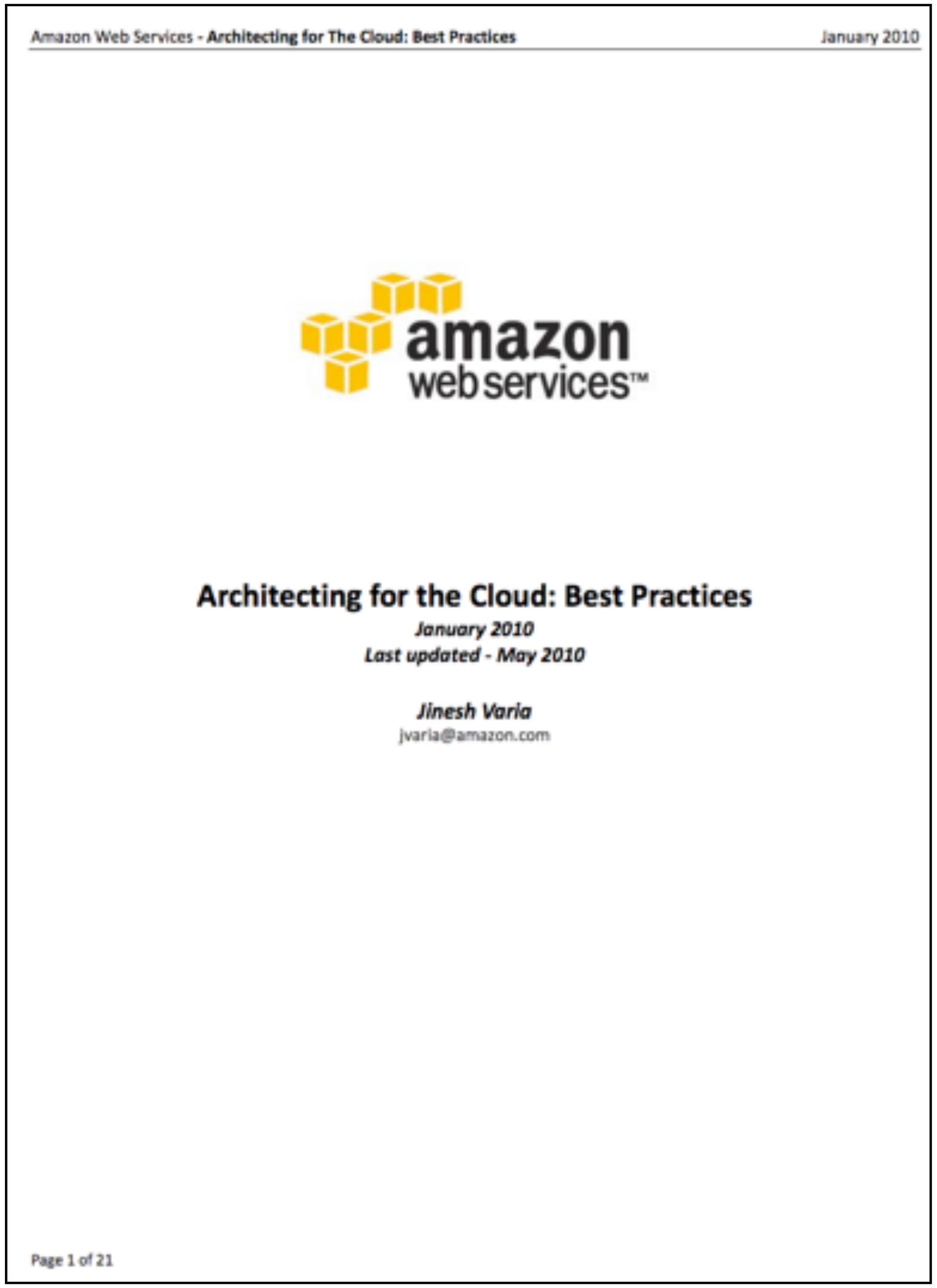
Design Patterns for the Cloud

based on

Amazon Web Services

Architecting for the Cloud: Best Practices

Jinesh Varia



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

Architecting for the Cloud: Best Practices

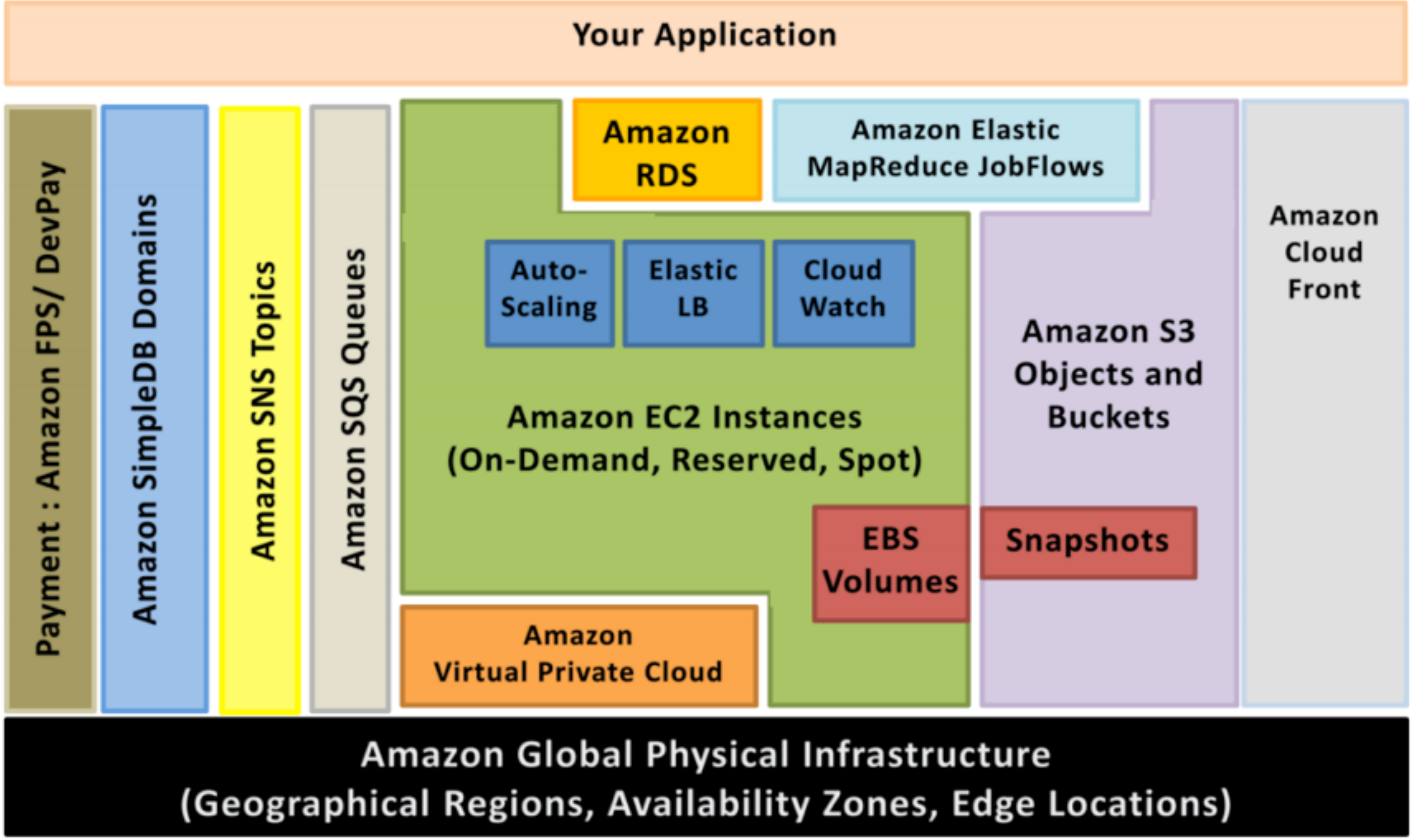
January 2010
Last updated - May 2010

Jinesh Varia
jvaria@amazon.com

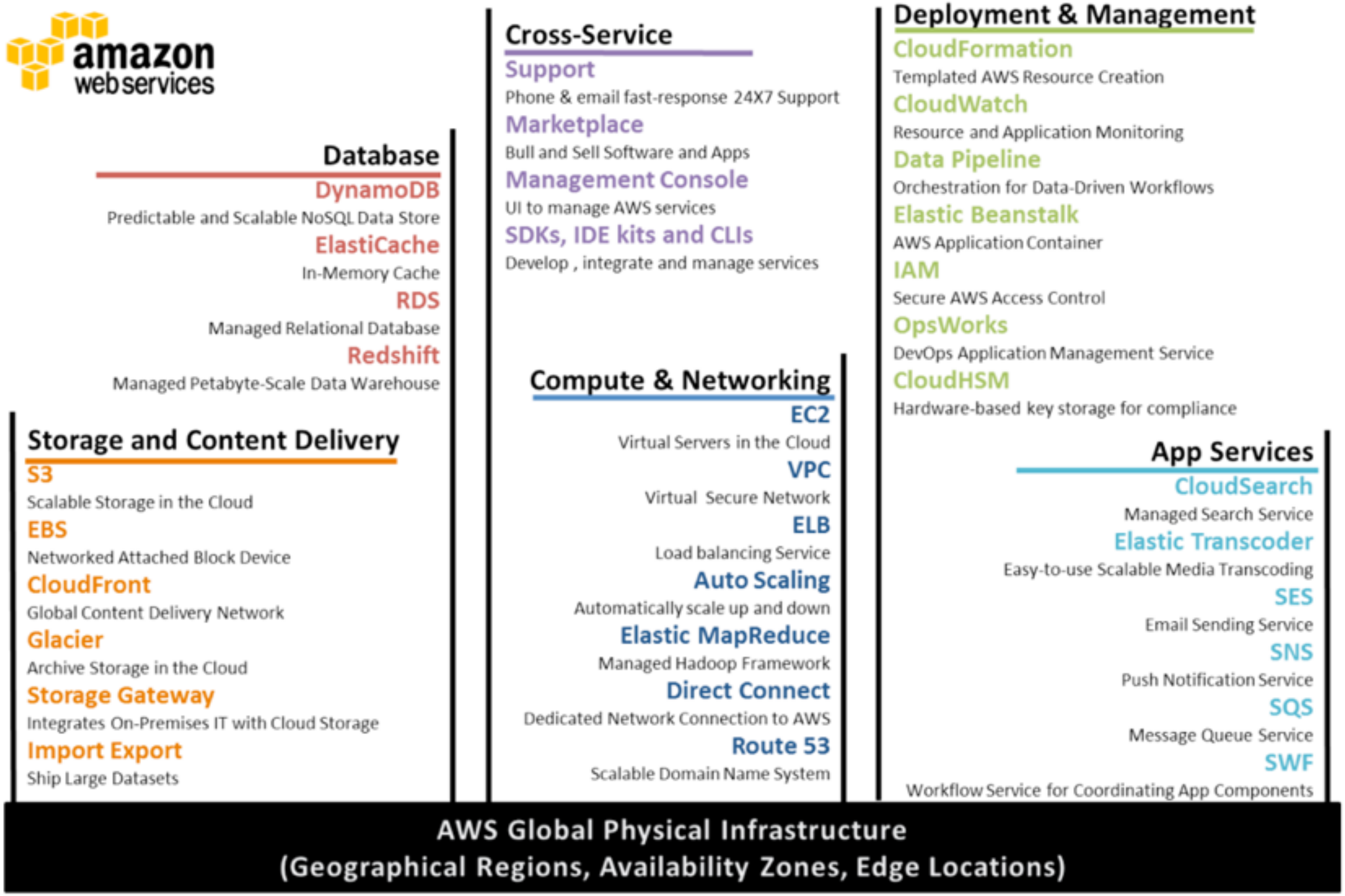
Page 1 of 21

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

Amazon Web Services


Your Application
Payment : Amazon FPS/ DevPay
Amazon SimpleDB Domains
Amazon SNS Topics
Amazon SQS Queues
Amazon RDS
Amazon Elastic MapReduce JobFlows
Auto-Scaling
Elastic LB
Cloud Watch
Amazon EC2 Instances (On-Demand, Reserved, Spot)
EBS Volumes
Amazon Virtual Private Cloud
Amazon S3 Objects and Buckets
Snapshots
Amazon Cloud Front
Amazon Global Physical Infrastructure (Geographical Regions, Availability Zones, Edge Locations)

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

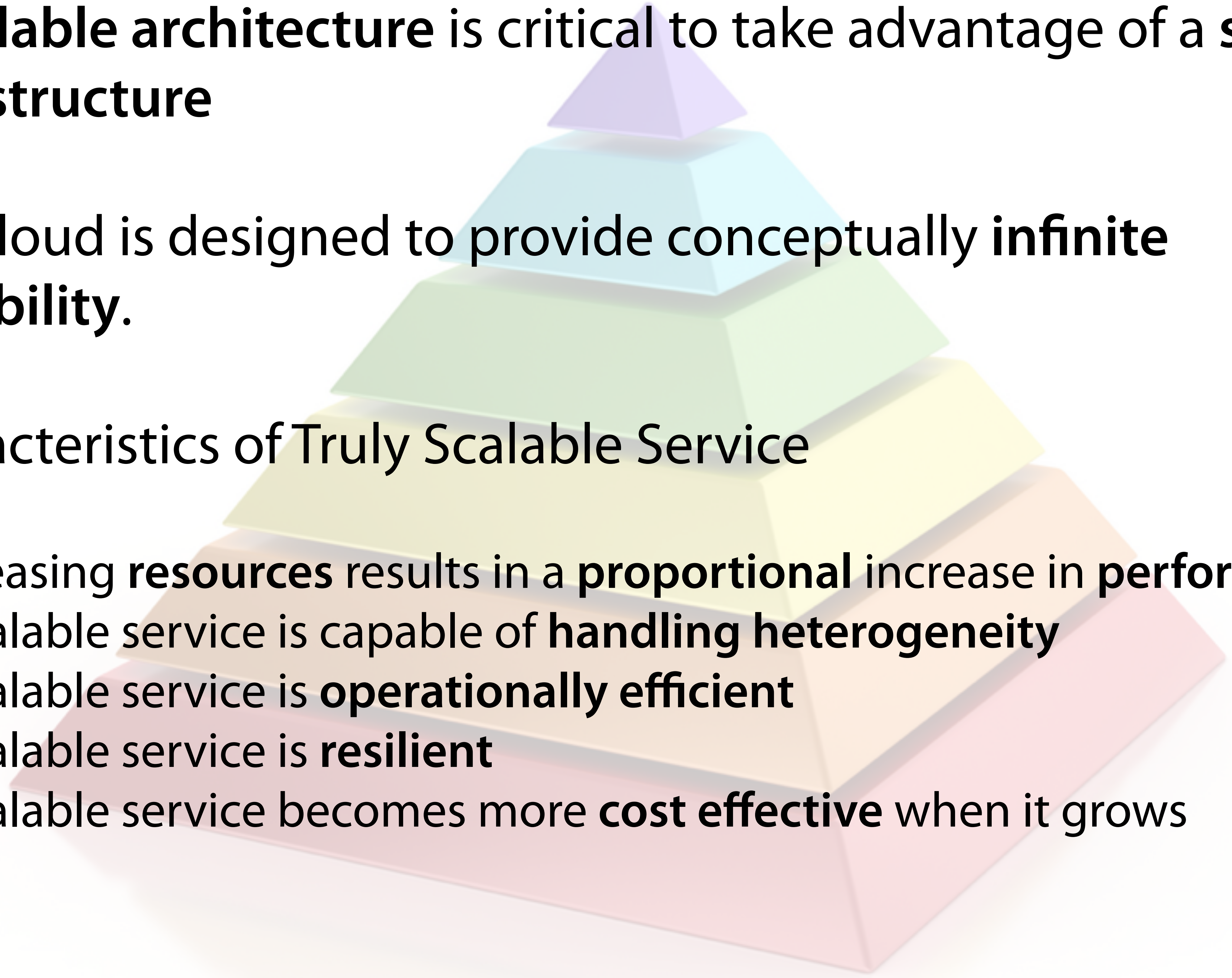
SNS
Push Notification Service

SQS
Message Queue Service

SWF
Workflow Service for Coordinating App Components

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

Scalable Architectures



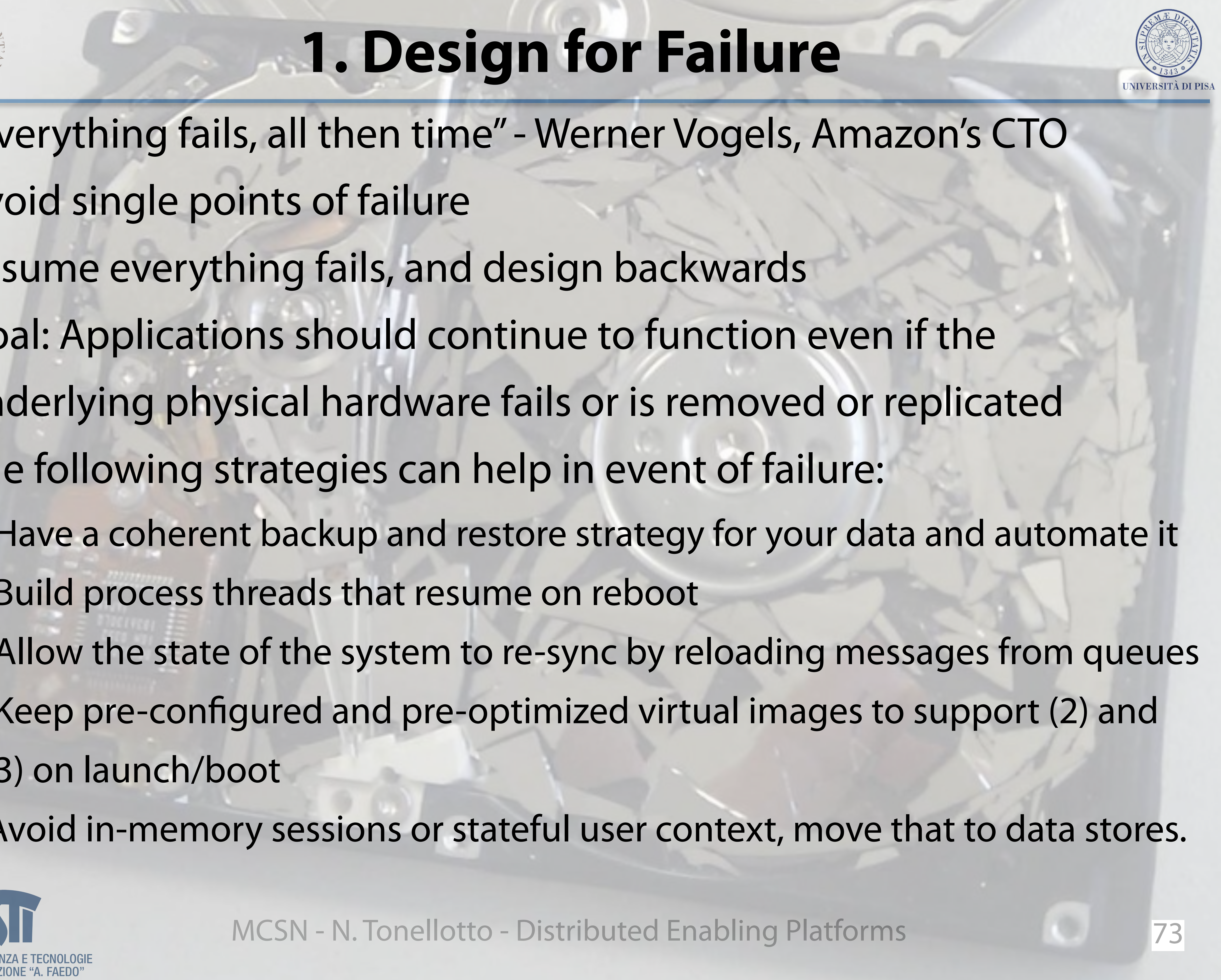
A **scalable architecture** is critical to take advantage of a **scalable infrastructure**

The cloud is designed to provide conceptually **infinite 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.

1. AWS Tactics

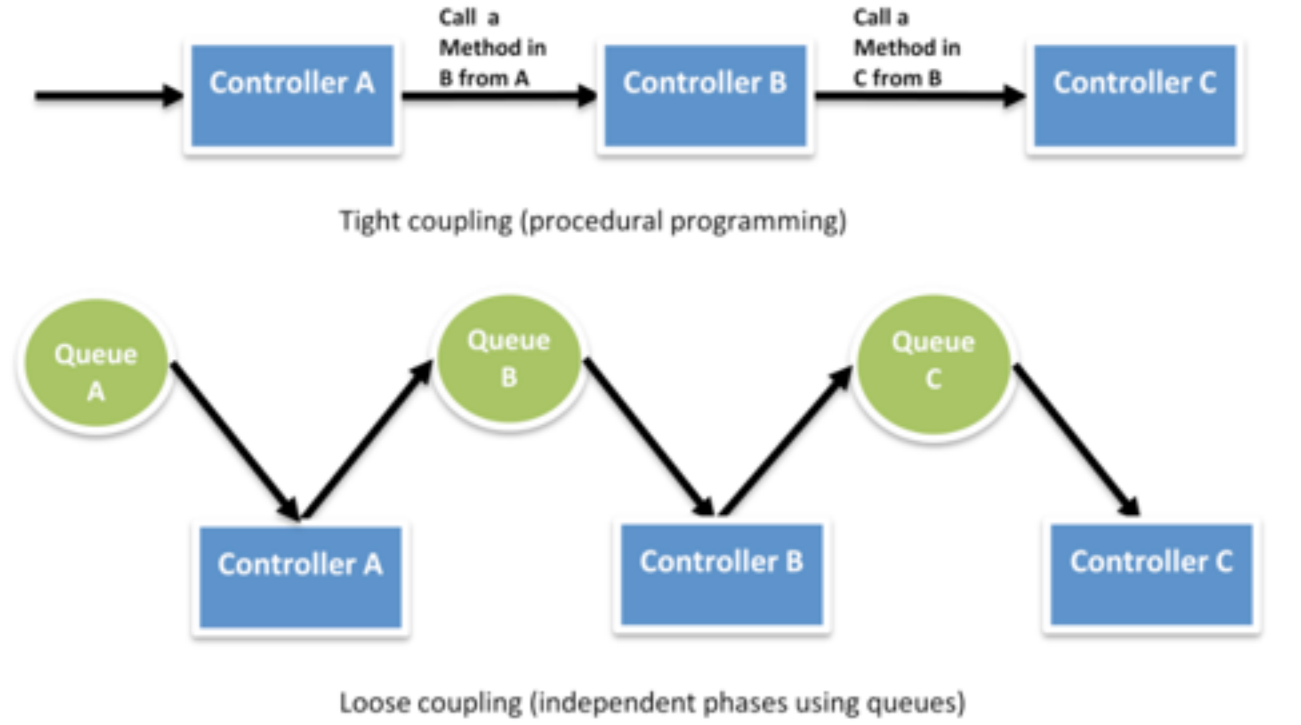
1. **Elastic IP** is a static IP that is dynamically re-mappable. You can quickly remap and failover to another set of servers so that your traffic is routed to the new servers.
2. **Availability Zones** are conceptually like logical datacenters. By deploying your architecture to multiple availability zones, you can ensure highly availability.
3. Maintain an **Amazon Machine Image** so that you can restore and clone environments very easily in a different Availability Zone.
4. Utilize **Amazon CloudWatch** (or various real-time open source monitoring tools) to get more visibility and take appropriate actions in case of hardware failure or performance degradation.
5. Setup an **Auto scaling group** to maintain a fixed fleet size so that it replaces unhealthy Amazon EC2 instances by new ones.
6. Utilize **Amazon EBS** and set up cron jobs so that incremental snapshots are automatically uploaded to **Amazon S3** and data is persisted independent of your instances.
7. Utilize **Amazon RDS** and set the retention period for backups, so that it can perform automated backups.

2. Design Loosely Coupled Systems

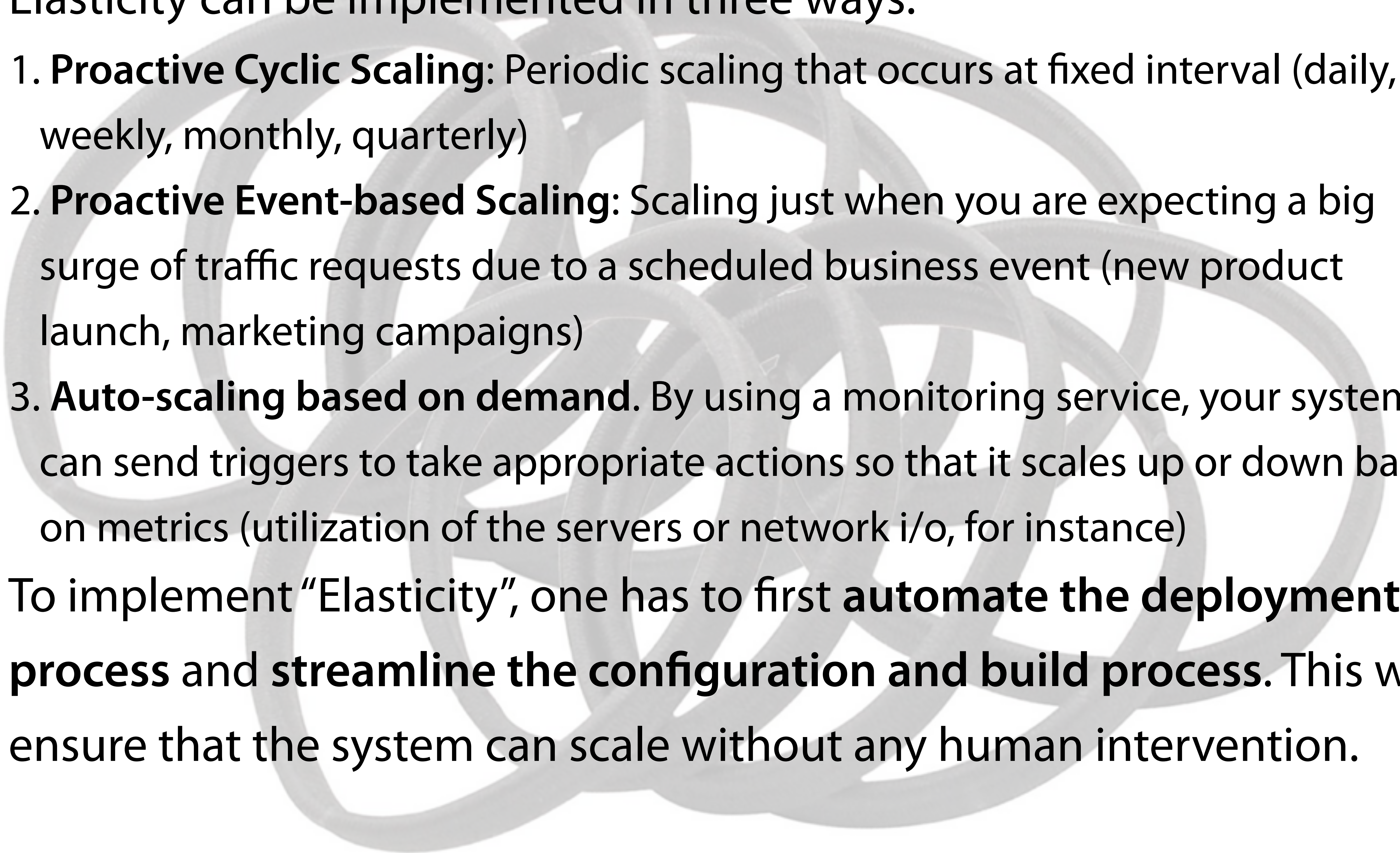
- The cloud reinforces the SOA design principle that **the more loosely coupled the components of the system, the bigger and better it scales.**
- Build components that **do not have tight dependencies** on each other.
- Build **asynchronous systems** and scaling horizontally become very important in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

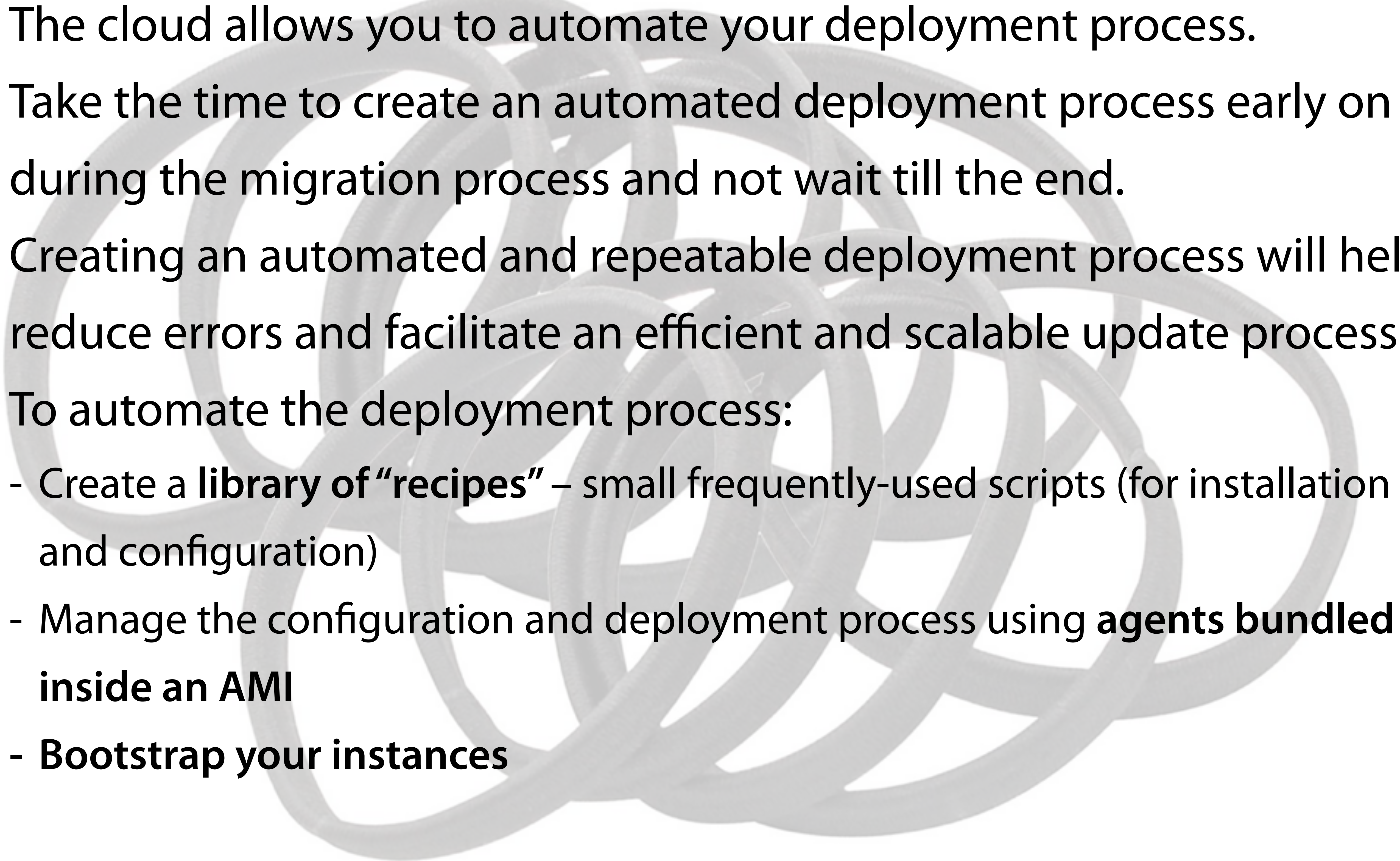

Controller A
Call a Method in B from A
Controller B
Call a Method in C from B
Controller C
Tight coupling (procedural programming)
Queue A
Queue B
Queue C
Controller A
Controller B
Controller C
Loose coupling (independent phases using queues)

3. Implement Elasticity



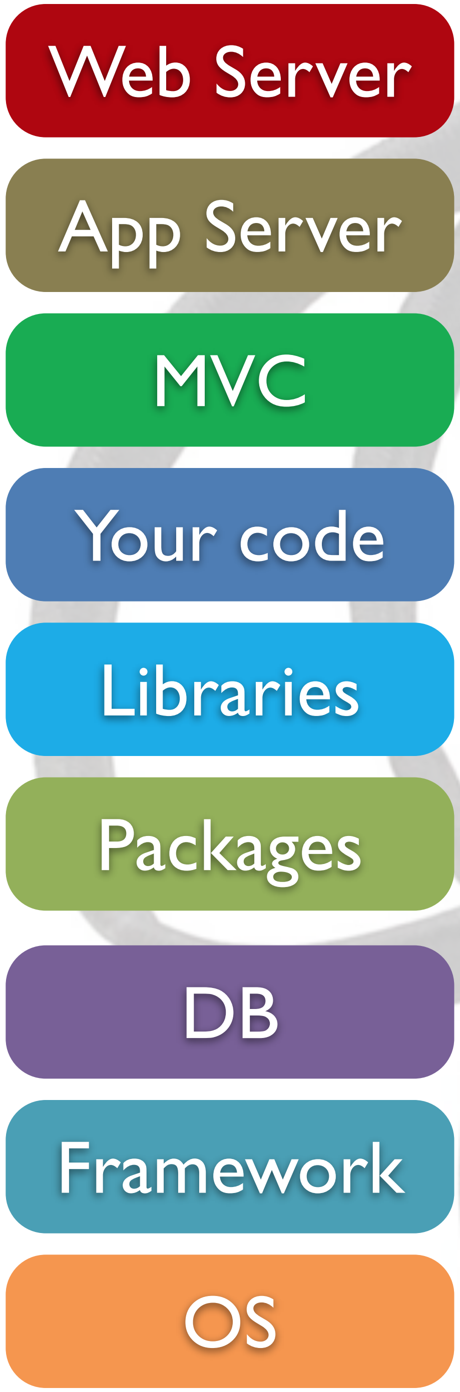
- Elasticity can be implemented in three ways:
  1. **Proactive Cyclic Scaling**: Periodic scaling that occurs at fixed interval (daily, weekly, monthly, quarterly)
  2. **Proactive Event-based Scaling**: Scaling just when you are expecting a big surge of traffic requests due to a scheduled business 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**
  - **Bootstrap your instances**

3. AMI Design Approaches



Web Server

App Server

MVC

Your code

Libraries

Packages

DB

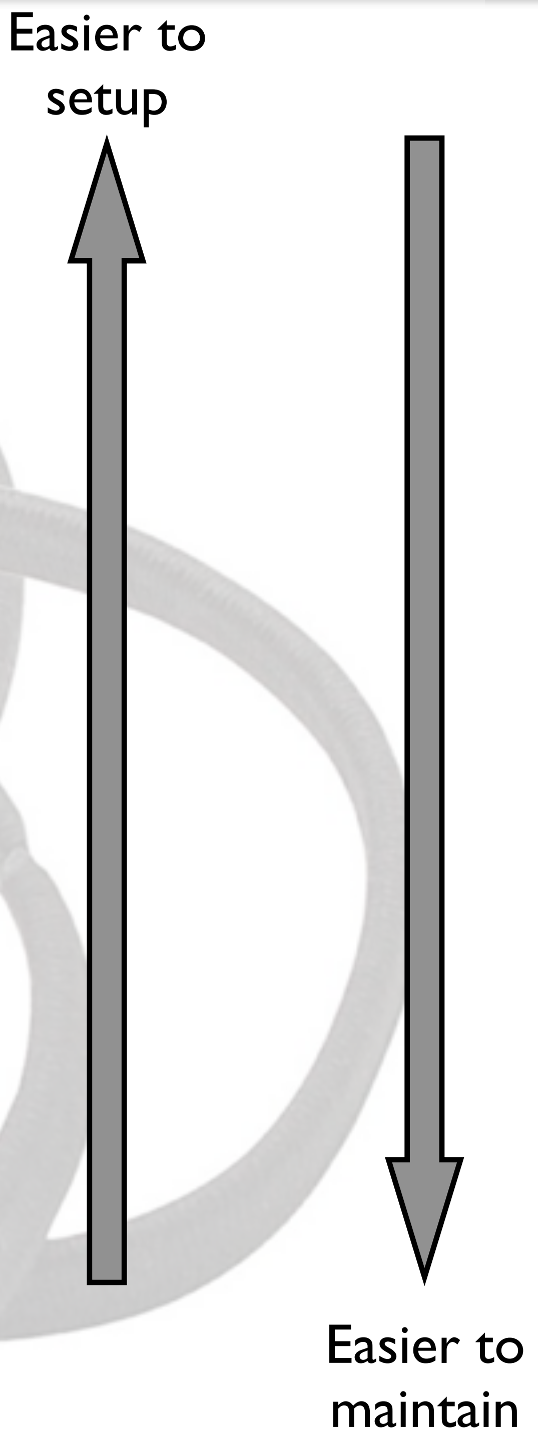
Framework

OS

1. Inventory of static AMIs

2. Golden AMIs with fetch on boot

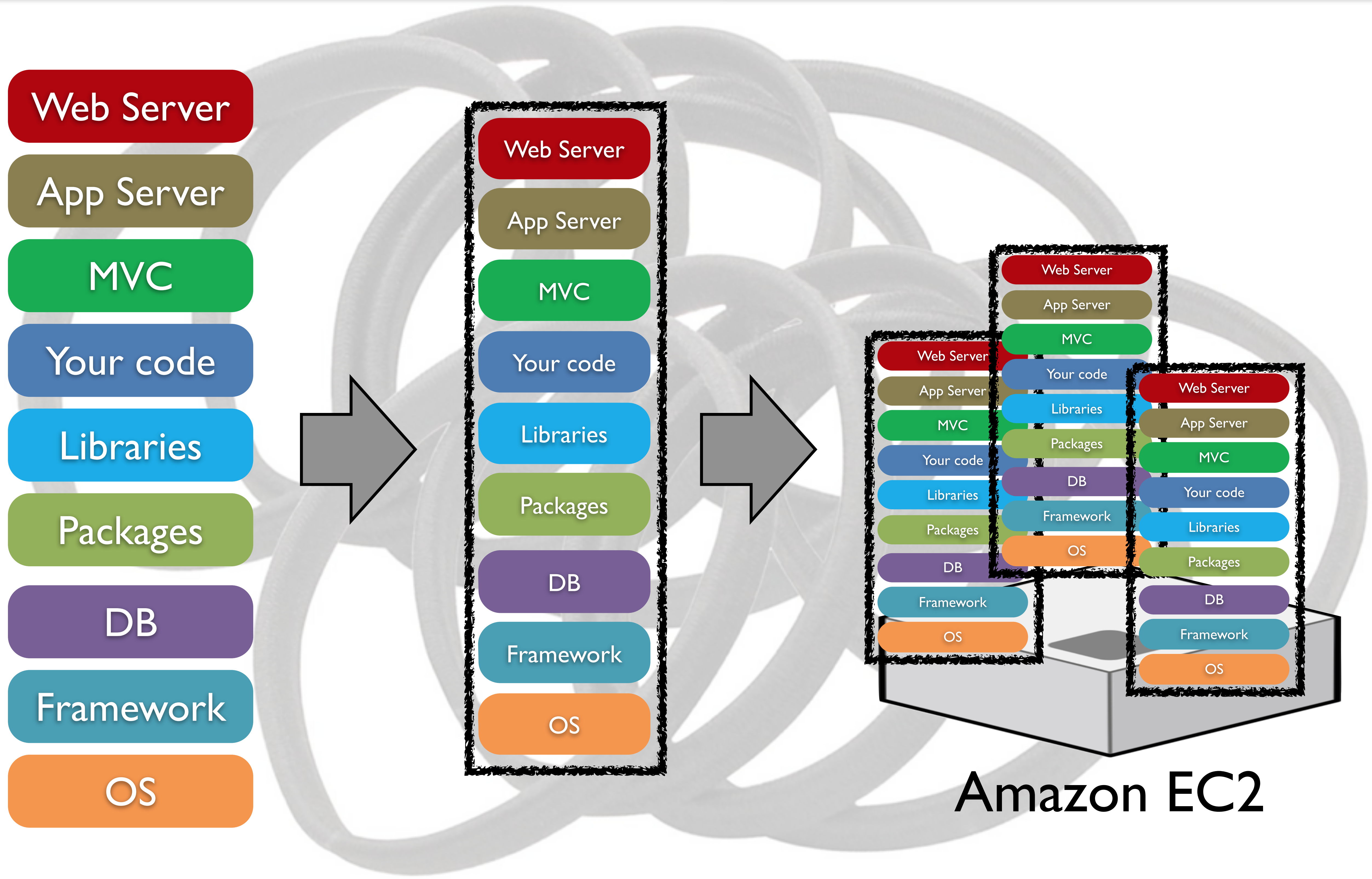
3. AMIs with Just Enough OS and agent



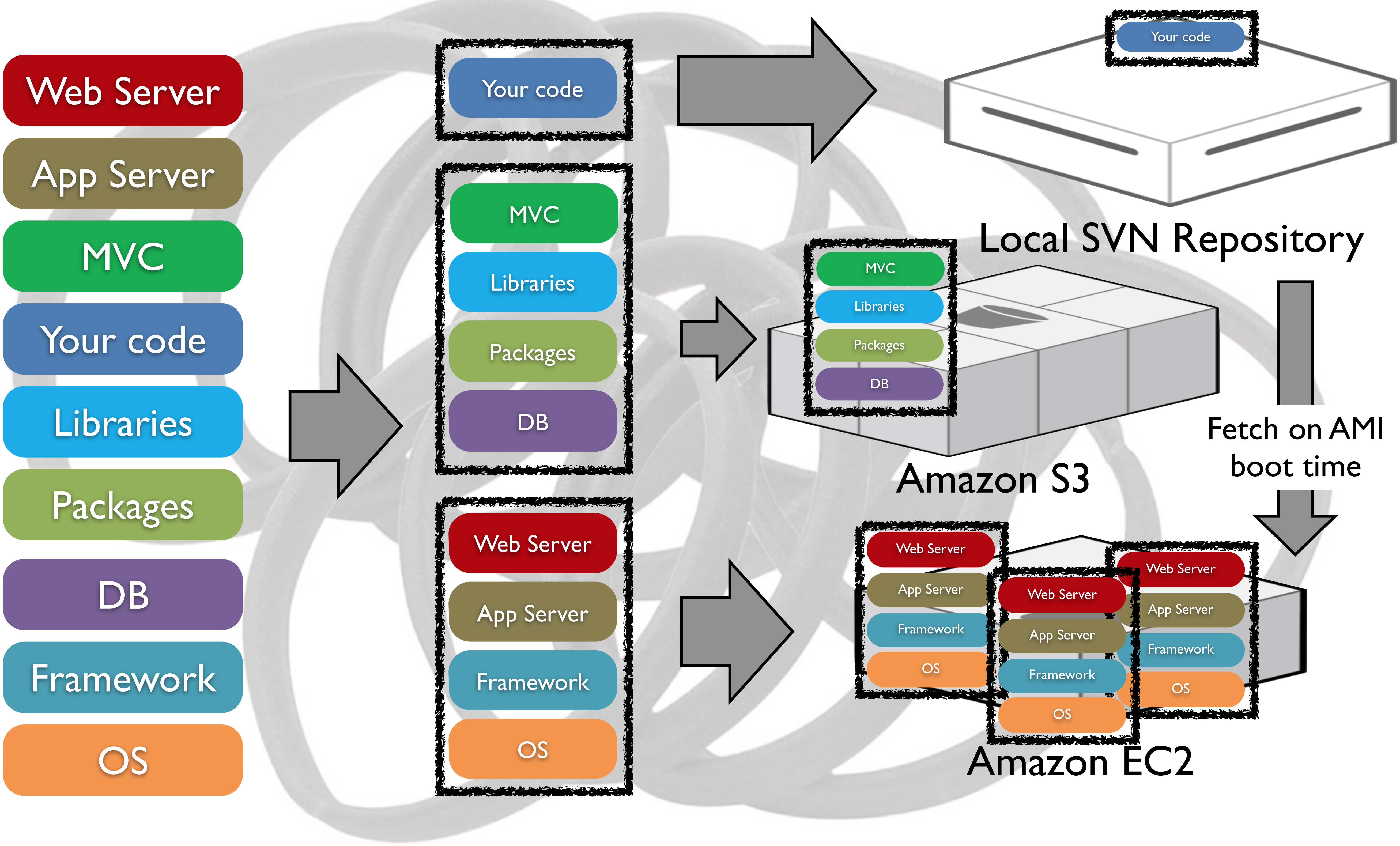
Easier to setup

Easier to maintain

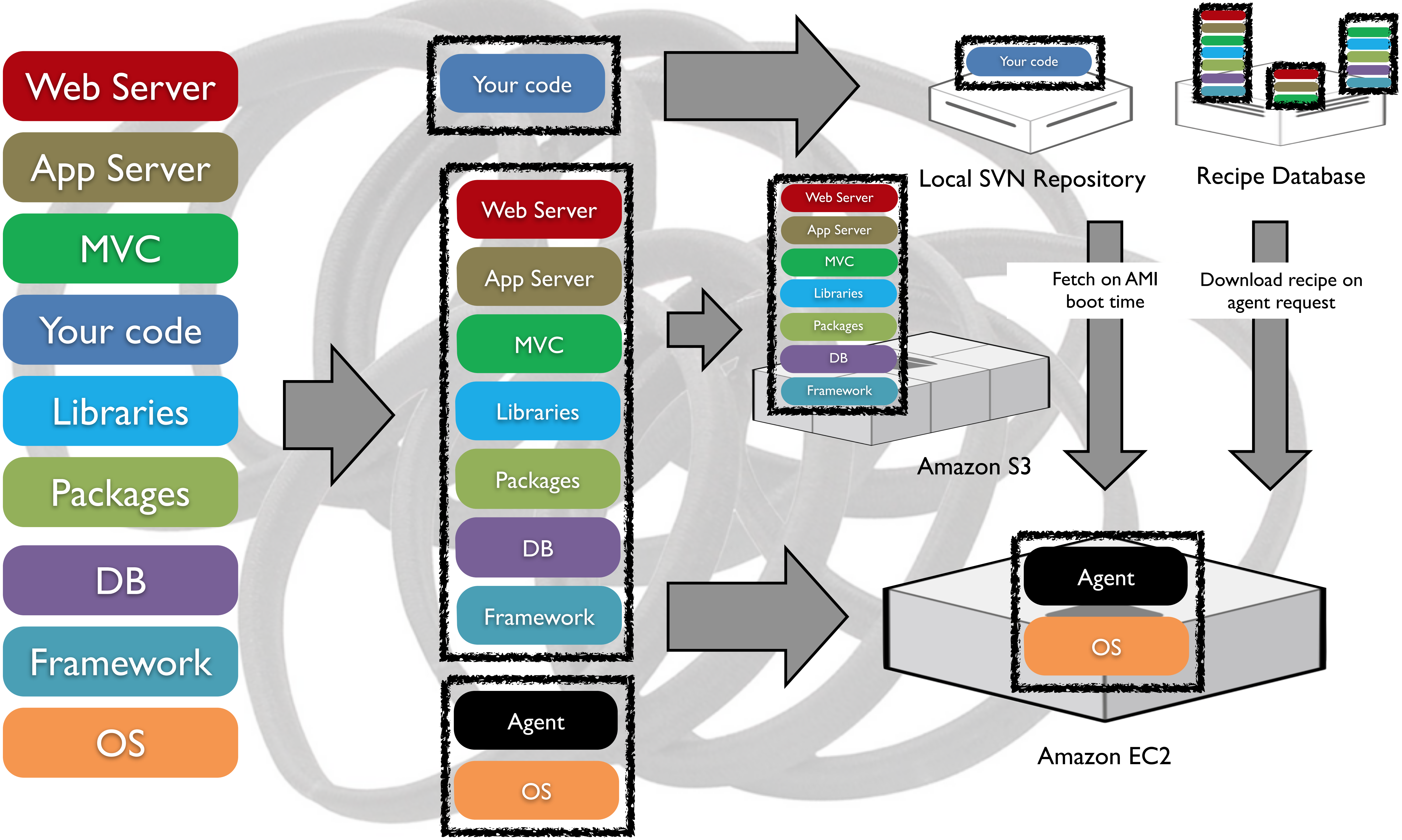
3. Inventory of static AMIs



3. Golden AMIs with fetch on boot



3. AMI with JeOS and agent



3. AWS Tactics

1. Define **Auto-scaling groups** for different clusters
2. **Monitor your system metrics** (CPU, Memory, Disk I/O, Network I/O) using Amazon CloudWatch and take appropriate actions (launching new AMIs dynamically using the Auto-scaling service) or send notifications.
3. **Store and retrieve machine configuration information dynamically**: Utilize Amazon SimpleDB to fetch config data during boot-time of an instance (eg. database connection strings). SimpleDB may also be used to store information about an instance such as its IP address, machine name and role.
4. Design a build process such that it **dumps the latest builds to a bucket** in Amazon S3; **download the latest version** of an application from during system startup.
5. Invest in **building resource management tools** (Automated scripts, pre-configured images) or Use smart open source configuration management tools.
6. Bundle **Just Enough Operating System** (JeOS) and your software dependencies into an Amazon Machine Image so that it is easier to manage and maintain. Pass configuration files or parameters at launch time and retrieve user data and instance metadata after launch.
7. Reduce bundling and launch time by **booting from Amazon EBS volumes** and attaching multiple Amazon EBS volumes to an instance. **Create snapshots** of common volumes and **share snapshots** among accounts wherever appropriate.
8. Application components should **not assume health or location of hardware** it is running on.