Cloud Computing / Enterprise Application Patterns

Class Administration

- Seminar Proposals Due Today.
- Software Project Proposal Due Next Week.
- Seminar Guidelines.
  - An hour long presentation.
  - Utilize diagrams, visualize aids.
  - Focus on the software components, not the business components. How can I benefit if I'm building a new system?
  - Provide examples, case studies.

Teams

- Team 1: MicroServices
- Team 2: Robotics
- Team 3: SDLC & MicroServices
- Team 4: Machine Learning
- Team 5: NLP & Computer Vision
- Team 6: SalesForce
- Team 7: Chaos Engineering

Definition

- Virtual servers available over the internet. Main goal is virtualization. Virtualization generalizes the physical infrastructure, which is the most rigid component, and makes it available as a soft component that is easy to use and manage.
- Any servers consumed outside the company's firewall is "in the cloud".
- The cloud aims to cut costs, and help the users focus on their core business instead of being impeded by IT obstacles.
- Cloud computing provides all of its resources as services.

Cloud Characteristics

- **Agility** improves with users' ability to re-provision technological infrastructure resources.
- Cloud computing systems typically use Representational State Transfer(REST) based APIs.
- **Cost**: cloud providers claim that computing costs reduce. Capital expenditure vs. operational.
- **Device and location independence:** enable users to access systems using a web browser regardless of their location or what device they use.
- **Virtualization:** technology allows sharing of servers and storage devices and increased utilization.

Cloud Characteristics

- **Multitenancy:** enables sharing of resources and costs across a large pool of users.
- **Reliability:** improves with the use of multiple redundant sites, which makes well-designed cloud computing suitable for business continuity and disaster recovery.
- **Scalability and elasticity:** via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis.
- **Performance:** is monitored, and consistent and loosely coupled architectures are constructed using web services.
- **Security** can improve due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data, and the lack of security for stored kernels.

Five essential characteristics

- ***On-demand self-service/provisioning.*** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- ***Broad network access.*** Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms.
- ***Resource pooling.*** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

Five essential characteristics

- ***Rapid elasticity.*** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
- ***Measured service. Pay per consumption.*** Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts).

Cloud Architecture Best Practices

1. Design for failure and nothing fails
2. Loose coupling sets you free
3. Implement elasticity
4. Build security in every layer
5. Don't fear constraints
6. Think parallel
7. Leverage different storage options

1. Design For Failure

- 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/replaced.
  - When, not if, an individual component fails, the application does not fail.

What is Chaos Monkey?

- **Chaos Monkey** is a software tool that was developed by Netflix engineers to test the resiliency and recoverability of their Amazon Web Services (AWS). The software simulates failures of instances of services running within Auto Scaling Groups (ASG) by shutting down one or more of the virtual machines.
- Github: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

2. Loose Coupling Architecture

- Design architectures with independent components The looser they're coupled, the bigger they scale
- Design every component as a black box
- De-coupling for hybrid models
- Load balance clusters

Software Architecture Trends

- Layered Architectures/Separate of Frontend development from backend.
- Best practices architecture:
  - Presentation layer/framework for the frontend.
  - RESTful layer/services for the backend
  - Backend database depending on the application design and requirements
- How would the application be packaged?
  - Either within a container
  - Traditional
- How would it be deployed?
  - Deploy the container
  - Provision a VM and deploy the code

FrontEnd Applications
FeathersJS
Node.js
Angular

FrontEnd Applications
Ember.js Node.js
Sails.js

Mobile Applications
Android
iPhone

Backend/Restful Layer
(Interacting with other backend services and database layer)

Vagrant/Docker Container
(Managing Virtualbox and Boot2Docker Machines)

Linux/Ubuntu 14.04

Portability/Containers Layer

- What's the benefit?
- **Immediate:** Improve developer's productivity
  - Imagine a new contractor is joining the team to implement enhancements for a new system.
  - With a container, you can just download the specific image, get latest from Github and you are set. No need to configure the environment or download source code. We will go through a demo of that.
- **Future Benefits:**
  - Changes also Continuous Integration (Phase III within the testing strategy).
  - Deploying containers using Jenkins.
  - Automation of Chef recipes. How?

What's the point?

- With the implementation of this layered architecture and this technology stack we reap multiple benefits and we enable the following:
  - Portability of the development environment among developers.
  - Ability to deploy containers in the future to DEV, QA and Prod environments.
  - Automation and improving the CI process(phase III).
  - Feeding scripts and infrastructure as code to the Infrastructure team.
- How do we accomplish this portability vision?

Through Containers

- Containers allow developers to package up an application and all of it's various components, package it all up in this box called container which is an isolated environment.
- What does it solve?
  - Solves the problem of dependencies. The underlying host OS and the environment that an application runs on is completely abstracted from the application.
- Few choices; the most dominant is Docker.
- What's the idea behind Docker?
  - Separates applications from Infrastructure using container technologies, similar to how virtual machines separate the operating system from the physical machine.

What are the components?

- Docker Engine: portable, lightweight runtime and a packaging tool.
- Docker Hub: cloud service for sharing containers. This is offered publicly. What does this mean internally?
- To use Docker you need two things:
  1. Install the boot2docker(Docker runtime) on your laptop/developer workstation to run containers.
  2. Ability to share containers through a secure internal Docker Hub.

Docker Benefits

- Improve developer's productivity:
  - Imagine a new contractor is joining the team to implement enhancements for a new system.
  - With a container, you can just download the specific image, get latest from Github and you are set. No need to configure the environment or download source code. We will go through a demo of that.
- Ability to deploy .NET solutions onto Linux environments killing pretty much windows server during Build 2015:

  http://thenewstack.io/docker-just-changed-windows-server-as-we-know-it/

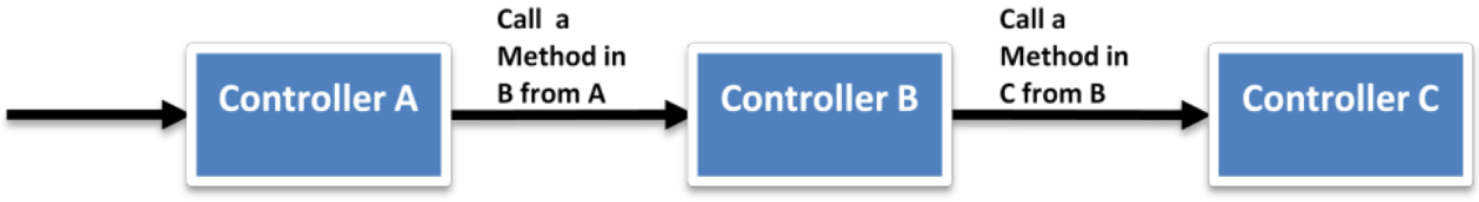
Tight Coupling

- Point to point example
  - Controller A
  - Controller B
  - Controller C

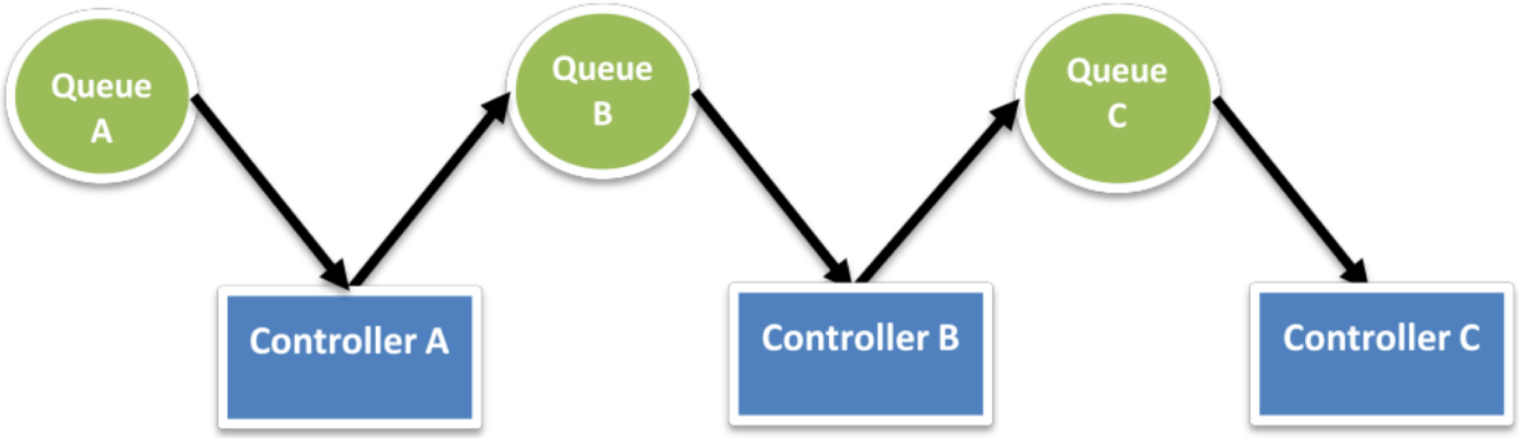
Loose Coupling using Queues

- Queue Example
  - Controller A
  - Controller B
  - Controller C

Tight vs Loose Coupling



Tight coupling (procedural programming)



Loose coupling (independent phases using queues)

3. What's the difference between Elasticity and Scalability?

- Elasticity: utilize resources in a dynamic and efficient way
- Scalability: ability of an application to grow without manual intervention or changing the design.
- Consider all infrastructure tiers: DNS, load-balancing, web, app, DB, network, storage.

On-Prem Elasticity

- It's accommodated by over-provisioning and under-utilizing resources.
- Costs include: capital, space, power, cooling and maintenance.
- An elastic environment is highly utilized all the time.

Blockers

- Manual: deployments, configurations, etc.
- Tight Coupling: chain reaction if a component fails
- Stateful: applications fail if you lose state
- Vertical: more RAM vs more nodes

How to implement elasticity and scale?

- Automated configurations and deployments.
- Loose coupling
- Stateless
- Horizontal scaling rather than vertical.

Implement Elasticity

- Don't assume the health, availability or fixed location of components
- Use designs that are resilient to reboot and re-launch
- Bootstrap your instances: When an instance launches it should ask "*Who am I and what is my role?*"
- Favor dynamic configuration

Implement Elasticity

- Auto Scaling
- Elastic Load Balancing
- NoSQL DB for configuration bootstrapping

4. Build Security in Every Layer

- 3 Tiers:
  - Web Tier: Ports 80 and 443 only open to the internet.
  - Application Tier: Engineering staff have SSH access to the App Tier.
  - Database Tier: All ports are blocked except ports from the App Tier.

5. Don't Fear Constraints

Re-think traditional architectural constraints

- Need more RAM
  - Instead, consider distributing load across machines or a shared cache
- Need better IOPS for database
  - Instead, consider multiple read replicas, sharding, or DB clustering !
- Hardware failed or config got corrupted
  - "Rip and replace" – Simply toss bad instance and instantiate replacement

6. Think Parallel

- Use multi-threading and concurrent requests to cloud services
- Run parallel **MapReduce** Jobs
- Use **Elastic Load Balancing** to distribute load

7. Leverage Many Storage Options

- Amazon S3: Large Objects
- Amazon Cloudfront: Content Distribution
- Amazon Simple DB: Simple non relational data
- Amazon EBS: Persistent block storage
- Amazon RDS: Automated, managed relational data
- Amazon EFS: NFS as a service
- Amazon Glacier: backup solution/deep archive

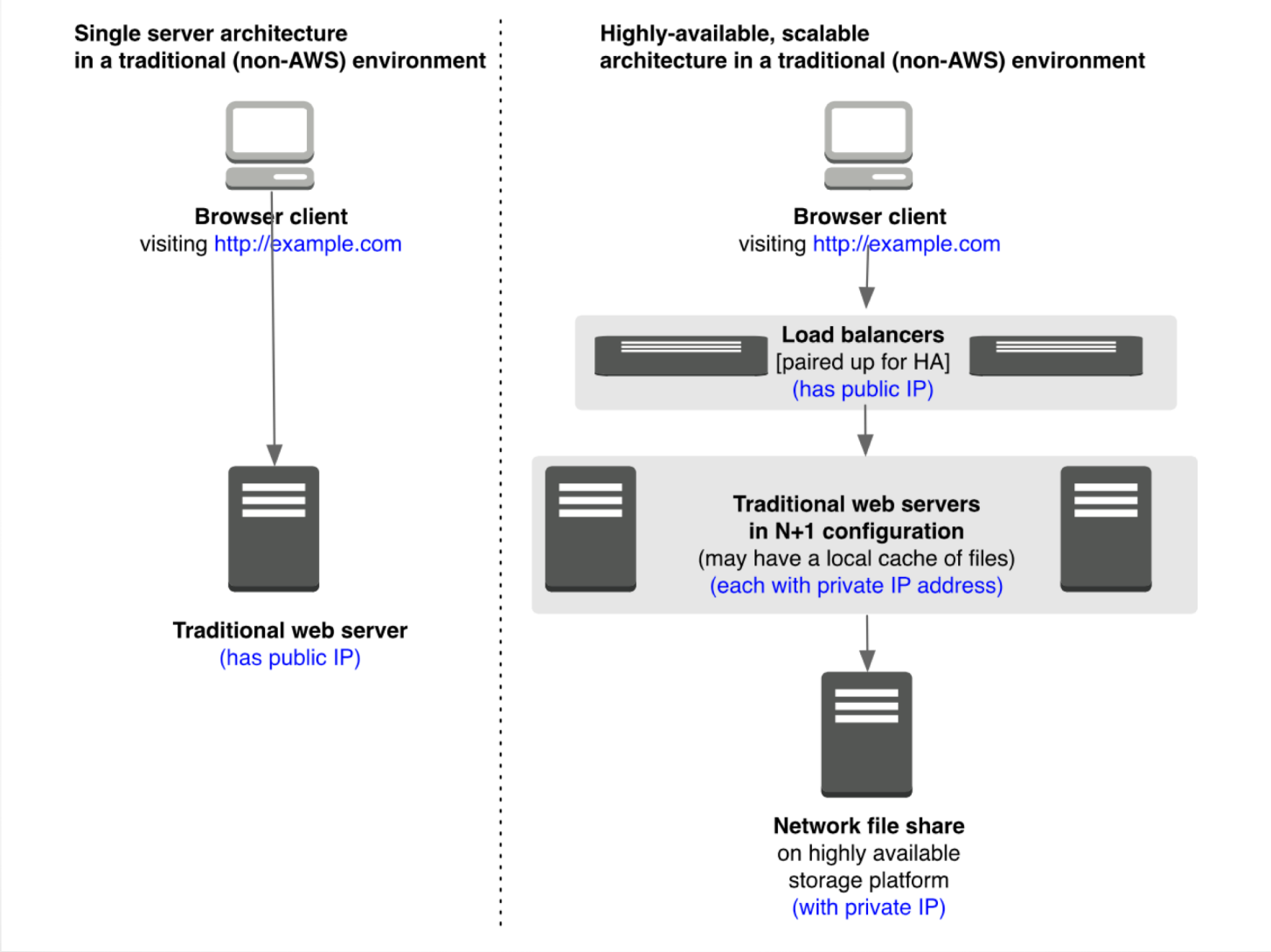
Enterprise Application Patterns

- Web Storage Pattern
- Event Driven Scaling
- Infrastructure As Code

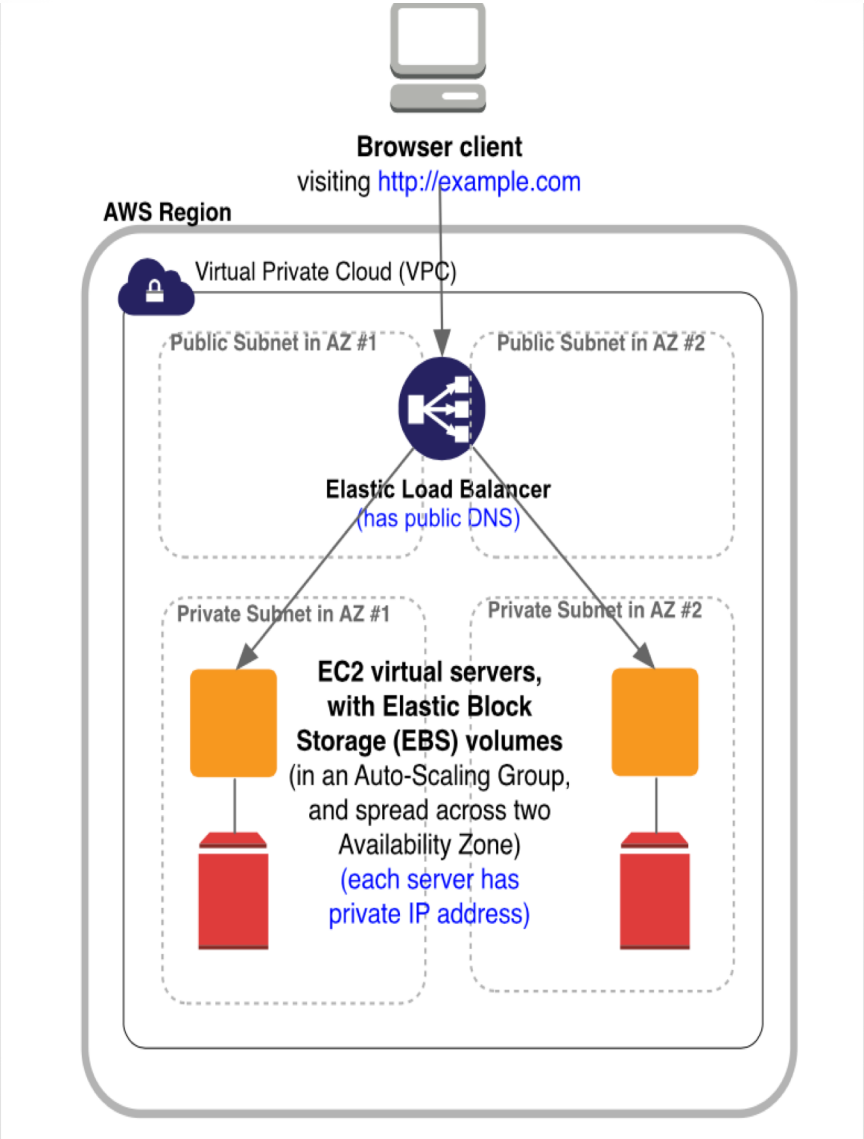
Web Storage Pattern

- Static vs Dynamic Websites.
- No server-side code execution is required.
- Traditional use of web servers challenged.

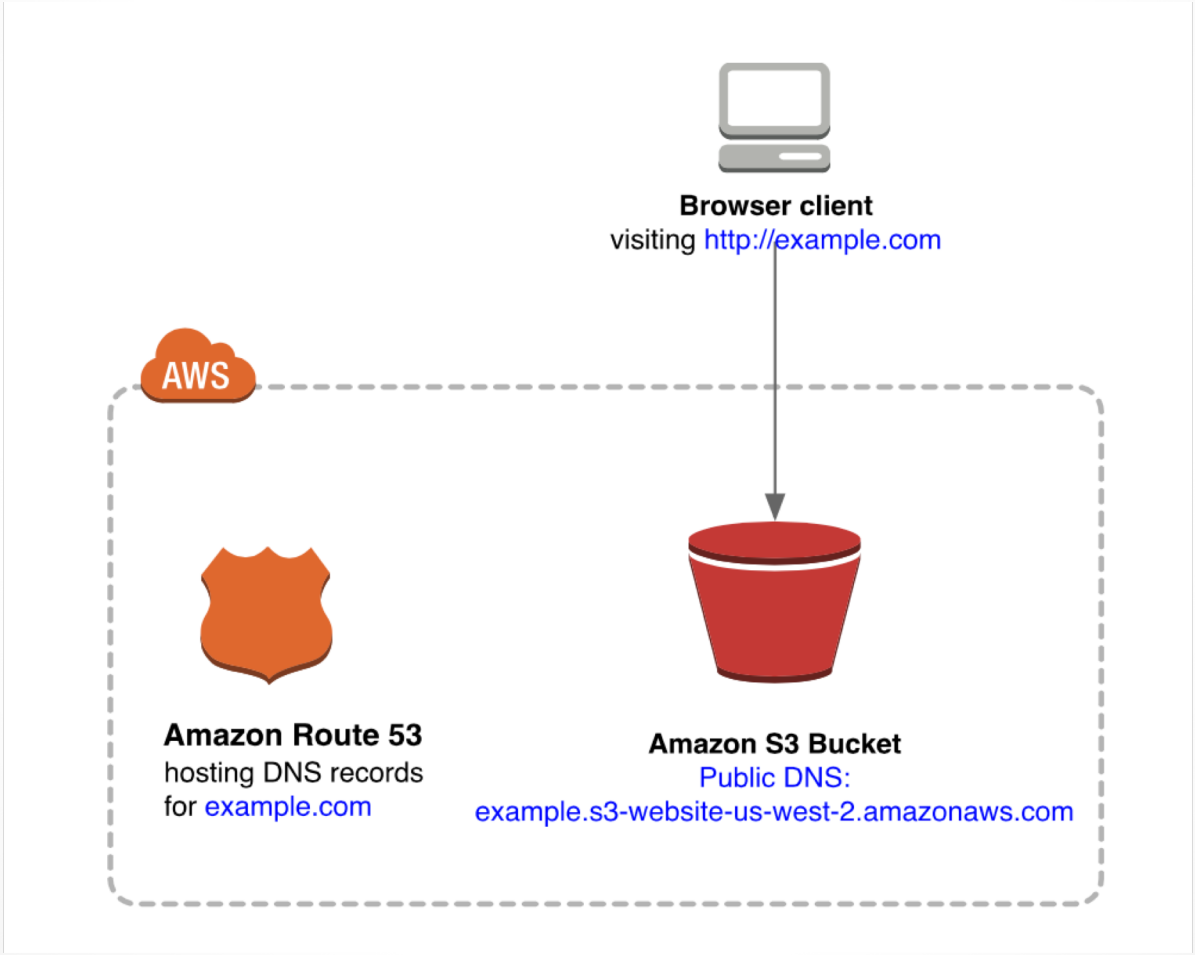
Traditional non-AWS Architecture



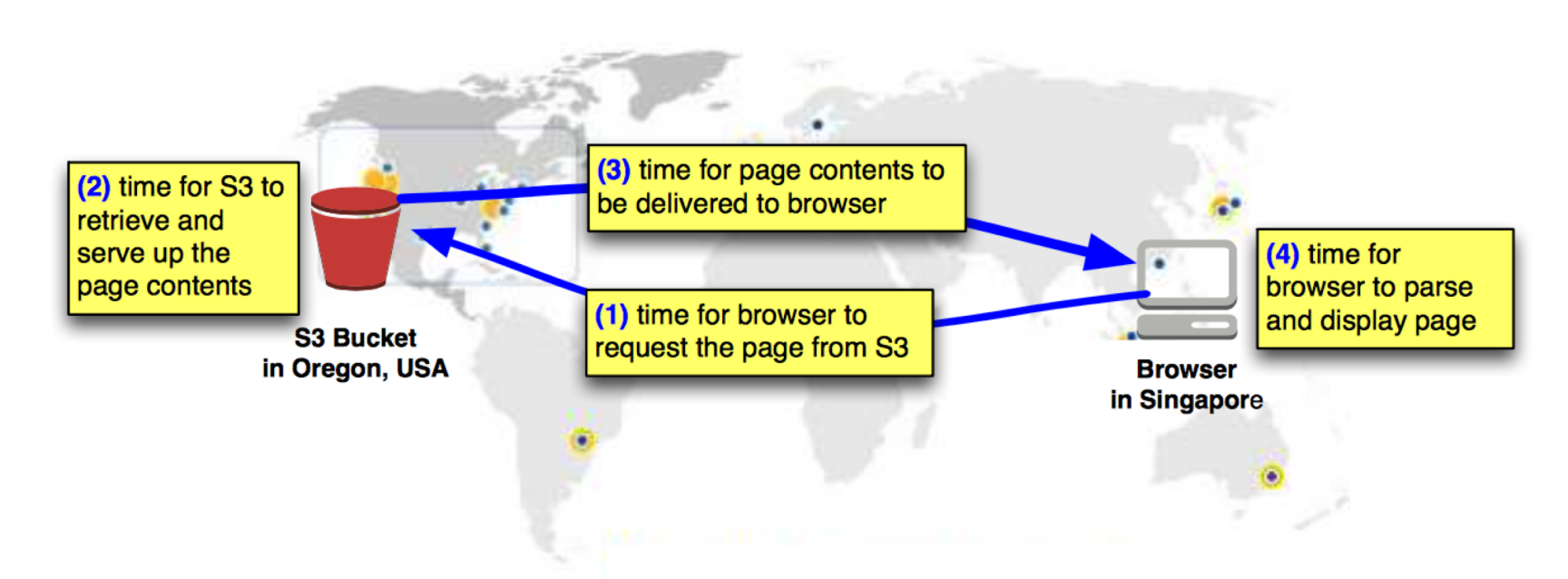
Lift and Shift AWS Architecture



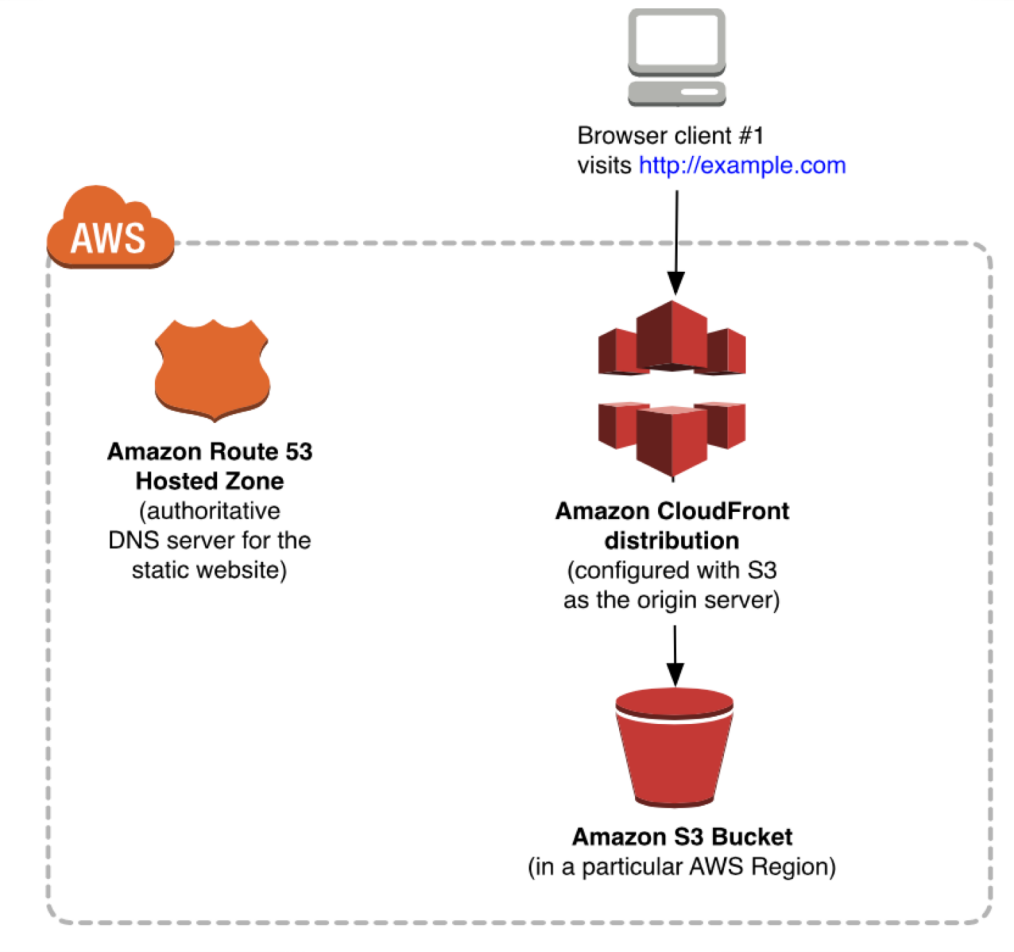
Can we do better?



User in Singapore trying to access the site?



AWS Architecture



Web Storage Pattern Summary

- Traditional (non-AWS) architectures for static websites.
- Evolved the architecture into a cloud-native architecture based on Amazon S3, CloudFront, and Route53.
- Resulting Architecture:
  - Highly available and scalable
  - Responsive user experience at low cost

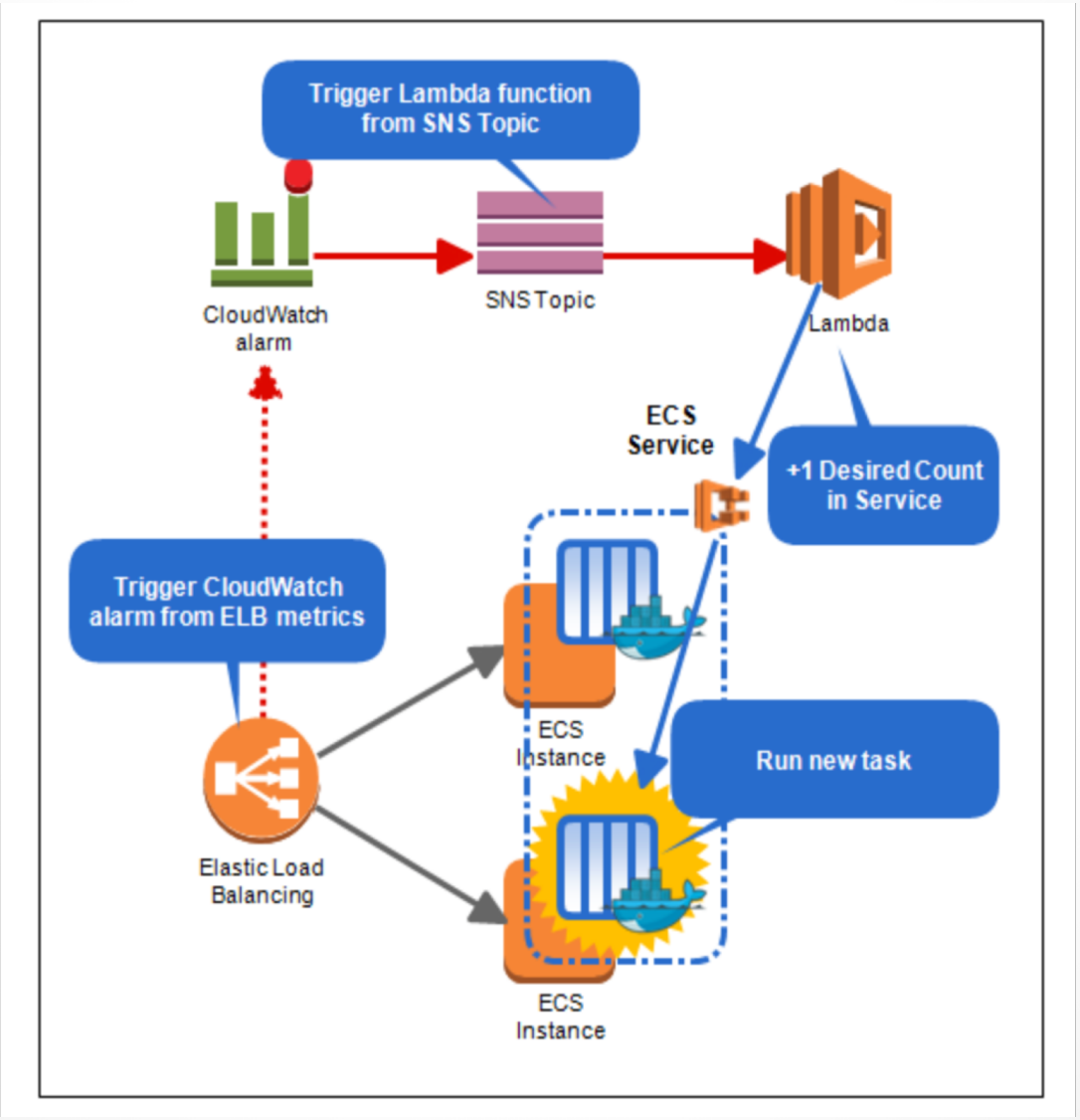
Event Driven Scaling

Used to scale Amazon ECS Services automatically using Amazon CloudWatch and AWS Lambda.

How does it work?

1. Amazon ECS cluster and service: Amazon EC2 compute instances for Docker containers and an Amazon ECS service to run your application
2. Amazon CloudWatch alarm: Identify when a CloudWatch metric goes above/below a threshold to increase/decrease the tasks (containers) running in the Amazon ECS service
3. Amazon SNS topic: Send a notification when the Amazon CloudWatch alarm is triggered
4. AWS Lambda function: Increase/decrease the desiredCount of the Amazon ECS service in response to the CloudWatch alarm

Event Driven Scaling



Amazon Web Services

Overview of Services

- Storage
- Compute/Containers
- Database
- Monitoring
- Messaging
- Networking
- Content Delivery
- Deployment & Management

Storage

- **Amazon Simple Storage Service(S3)**
  - Store and retrieve any amount of data, at any time, from
  - Anywhere on the web
  - Highly scalable, reliable, fast and inexpensive
- **Amazon Elastic Block Store (EBS)**
  - Block level storage volumes for use with Amazon EC2 instances
  - Off-instance storage that persists independently
  - Can be attached to a running Amazon EC2 instance and exposed as a device
- **Amazon Glacier**
  - Secure, durable, and extremely low-cost storage service for data archiving and long-term backup
  - Amazon Glacier is optimized for infrequently accessed data where a retrieval time of several hours is suitable.

Compute/Containers

- **Amazon Elastic Compute Cloud (Amazon EC2)**
  - Resizable compute capacity
  - Complete control of your computing resources
  - Reduces the time required to obtain and boot new
  - Server instances to minutes
  - Scale capacity as your computing requirements change
  - Pay only for capacity that you actually use
- **Amazon Elastic Container Service(ECS)**
  - Eliminates the need for you to install, operate, and scale your own cluster management infrastructure.
  - Highly scalable, high performance container management service that supports Docker containers.

Database

- **Amazon SimpleDB**
  - No schema, automatic indexing
  - Eliminates the administrative burden of data modeling, index maintenance, and performance tuning
  - Real-time lookup and simple querying of structured data
- **Amazon Relational Database Service (RDS)**
  - Automatically patches the database software and backs up your database
  - Cost-efficient and resizable capacity
  - Manages time-consuming database administration tasks
  - Access to the full capabilities of a familiar MySQL database
- **Amazon DynamoDB**
  - NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale.
  - Fully managed cloud database and supports both document and key-value store models

Monitoring

- **Amazon CloudWatch**
  - Visibility into resource utilization, operational performance, and overall demand patterns
  - Metrics such as CPU utilization, disk reads and writes, and network traffic
  - Accessible via the AWS Management Console, web service APIs or Command Line Tools

Networking

- **Amazon Elastic Load Balancing**
  - Supports the routing and load balancing of HTTP, HTTPS and TCP traffic to EC2 instances
  - Supports health checks to ensure detect and remove failing instances
  - Dynamically grows and shrinks required resources based on traffic
  - Seamlessly integrates with Auto-scaling to add and remove instances based on scaling activities

Messaging

- **Amazon Simple Notification Service (SNS)**
  - Set up, operate, and send notifications
  - Publish messages from an application and immediately deliver them to subscribers or other applications
- **Amazon Simple Email Service (Amazon SES)**
  - Bulk and transactional email-sending service
  - Eliminates the hassle of email server management, network configuration, and meeting rigorous Internet Service Provider (ISP) standards
- **Amazon Simple Queue Service (SQS)**
  - Hosted queue for storing messages as they travel between computers
  - Move data between distributed components of their applications

Content Delivery

- **Amazon CloudFront**
  - Web service for content delivery
  - Distribute content to end users with low latency, high data transfer speeds, and no commitments
  - Delivers your content using a global network of edge locations

Deployment and Management

- **AWS Elastic Beanstalk**
  - Simply upload your application
  - Automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring
  - Retain full control over the AWS resources powering your application
- **AWS CloudDeploy**
  - Service that automates code deployments to any instance, including Amazon EC2 instances and instances running on-premises.
  - Automate software deployments, eliminating the need for error-prone manual operations, and the service scales with your infrastructure so you can easily deploy to one instance or thousands.

References

- AWS Documentationhttps://aws.amazon.com/architecture/
- http://jineshvaria.s3.amazonaws.com/public/cloudbestpractices-jvaria.pdf
- Static Sites: https://d0.awsstatic.com/whitepapers/Building%20Static%20Websites%20on%20AWS.pdf