Carnegie Mellon University
Software Engineering Institute

Cloud Computing: An Architecture-centric View

John Klein
jklein@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Objectives for This Course

Explain how cloud computing is different from traditional data center deployment

Identify how the controllability and observability of cloud-based systems impacts test and evaluation approaches

Explain how cloud computing promotes and inhibits system quality attributes (including cybersecurity), and how this impacts test and evaluation approaches

Identify potential areas of risk in cloud-based systems

Introduction

Who am I?

Who are you?

Why are you here?

Instructor Introductions

John Klein

Senior Member of the Technical Staff, CMU SEI

jklein@sei.cmu.edu

https://www.sei.cmu.edu/about/people/profile.cfm?id=klein_14435

Tim Morrow

Security Solutions Engineer, CMU SEI

tbm@cert.org

https://www.sei.cmu.edu/about/people/profile.cfm?id=morrow_16360

Agenda – 1

Definitions and fundamental concepts

- essential characteristics of cloud computing, cloud delivery service models, deployment approaches (private, community, hybrid), government-specific cloud offerings

Enabling technologies

- virtualization, containerization, infrastructure as code

Cloud native services

- out-of-the-box services from cloud providers for storage and databases, application integration, monitoring, scaling and load balancing, identity and access management, analytics

Agenda – 2

Introduction to Security

Quality attributes in the cloud

- how cloud computing promotes or inhibits qualities such as availability, performance, scalability, testability, modifiability/ extensibility, and cybersecurity

Distributed systems concepts

- communication/coordination limits in distributed systems, consistency/availability/partition tolerance tradeoffs for distributed state/data, time synchronization

Using the cloud to support test and evaluation

- how to leverage the elasticity and scalability of the cloud to test and evaluate systems

Rules of Engagement

We will be very busy today.

To complete everything and get the most from the course, we will need to follow some rules of engagement:

- Your participation is essential.
- Feel free to ask questions at any time.
- Discussion is good, but we might need to cut some discussions short in the interest of time.
- Please try to limit side discussions during the lectures.
- Please turn off your cell phone ringers and computers.
- Let's try to start on time.

Any Questions So Far?

Cloud Computing: An Architecture-centric View

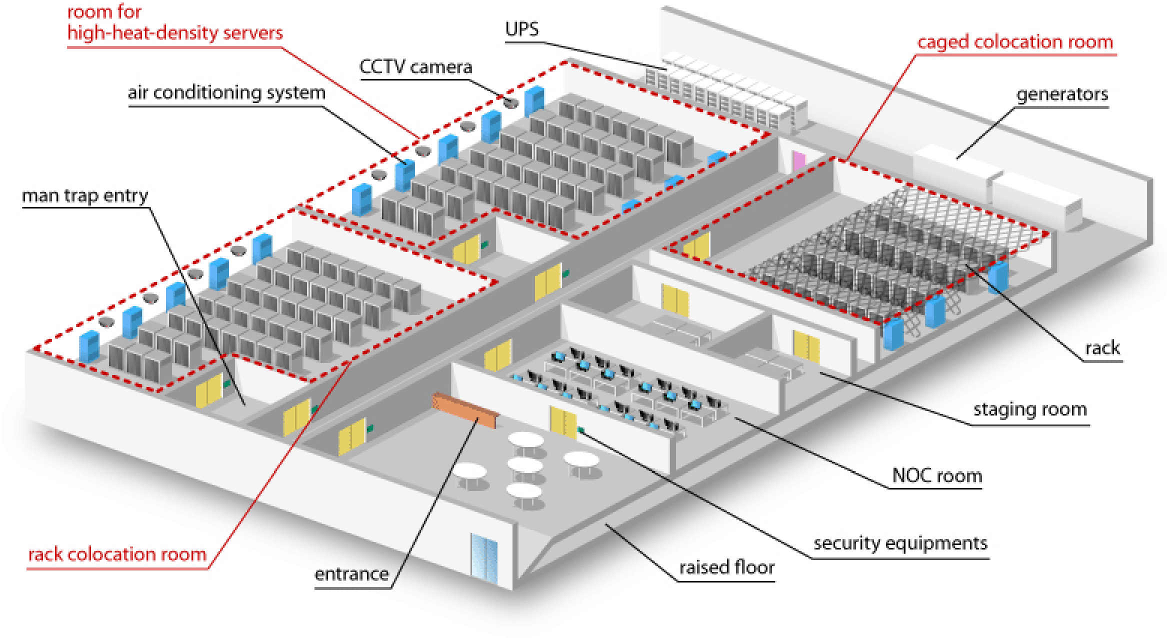
Definitions and Fundamental Concepts

Definitions and Fundamental Concepts

In this module, we will discuss

- What makes cloud computing different from a typical data center
- Cloud service models
- Cloud delivery models
- Cloud options available for US government systems
- Security controls
- Service level agreements

Data Center Deployment

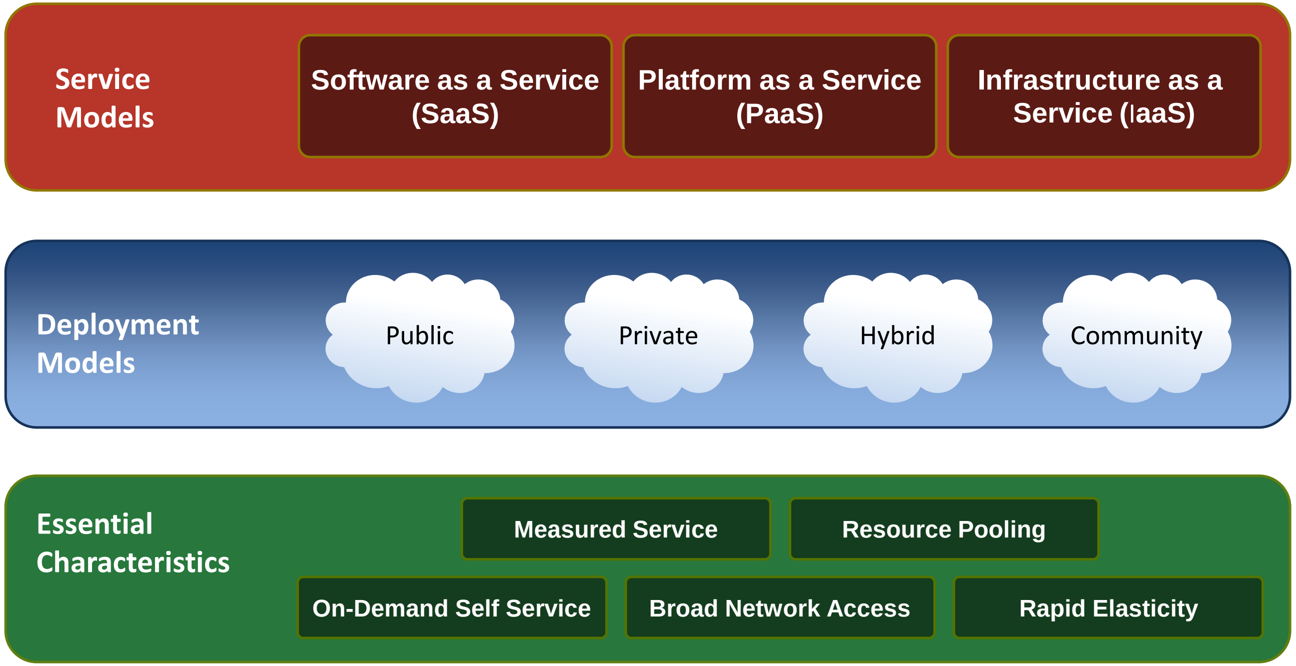


Cloud Computing

*“A model for enabling **convenient**, **on-demand** network access to a **shared** pool of **configurable** computing resources (e.g., networks, servers, storage, applications, and services) that can be **rapidly provisioned and released** with minimal management effort or service provider interaction.”*

Source: National Institute of Standards and Technology (NIST), 2011

Cloud Computing Models and Essential Characteristics



Source: National Institute of Standards and Technology (NIST), 2011

NIST Cloud Model's Five Essential Characteristics

On-demand self-service – 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 (e.g., mobile phones, tablets, laptops, and workstations)

Resource pooling – the service 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.

NIST Cloud Model’s 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 – 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).

Deployment Models

Public

- Offered as a service, usually over an Internet connection
- Typically charge a pay-per-use fee
- Users can scale on-demand and do not need to purchase hardware
- Cloud providers manage the infrastructure and pool resources into capacity required by consumers

Private

- Deployed inside the firewall and managed by the user organization
- User organization owns the software and hardware running in the cloud
- User organization manages the cloud and provides cloud resources
- Resources typically not shared outside the organization and full control is retained by the organization

Hybrid

- Combination of public and private cloud and/or community

Community

- Cloud that contains functionality tailored for the industry that it serves

Service Delivery Models

Infrastructure as a Service (IaaS)

- CPUs
- Disk drives
- Networks
- Data centers

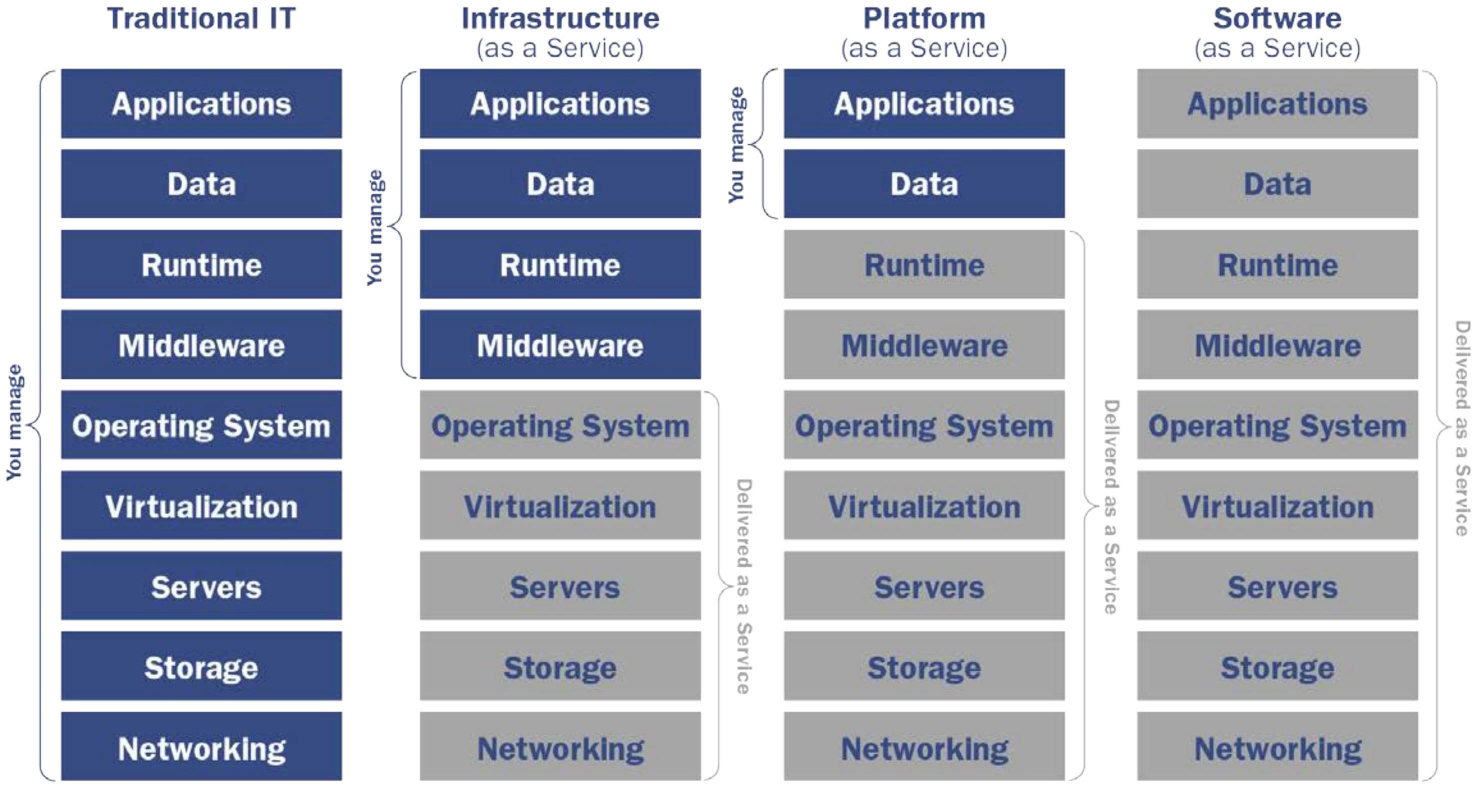
Platform as a Service (PaaS)

- Development and runtime tools and environment

Software as a Service (SaaS)

- Enterprise apps
- Desktop apps
- Mobile apps

Shared Responsibilities Model



Drivers for Cloud Computing Adoption

Availability	24x7 access to data and applications from anywhere
Big Data	Public clouds have significantly reduced the cost of entry into big data, machine learning, and artificial intelligence systems
Elasticity and Scalability	Organizations can request, use, and release as many resources as needed based on changing needs and user demand
Lower Infrastructure Costs	The pay-per-use model allows an organization to only pay for the resources they need with basically no investment in the physical resources available in the cloud — there are no infrastructure maintenance or upgrade costs
Reduced Development Times	• Available tools and platforms, in addition to DevOps procedures, can reduce amount of code to write and deployment times • Multi-organizational projects can work simultaneously on common data and information
Reliability	In order to support SLAs (service-level agreements), cloud providers have reliability mechanisms that are much more robust than those that could be cost-effectively provided by a single organization
Risk Reduction	Organizations can use the cloud to test ideas and concepts before making major investments in technology

Challenges for Cloud Computing Adoption

Interoperability	A universal set of standards and/or interfaces has not yet been defined, resulting in a significant risk of vendor lock-in
Latency	All access to the cloud is done via the internet, introducing latency into every communication between the user and the environment
Legal Issues	There are concerns in the cloud computing community over jurisdiction, data protection, data location, fair information practices, international data transfer, and legal access to data
Platform or Language Constraints	Some cloud environments provide support for specific platforms and languages only
Security	The key concern is data privacy: organizations typically do not have control of or know where their data is being stored
Skills/Knowledge	Different skills are needed to make use of clouds at the different services than a traditional IT center
Compliance	Satisfying NIST Special Publication 800-53 security controls and assessment procedures for the program's appropriate security control level
Portability	Cloud service providers provide similar functionality but implement their services differently

FedRAMP

Government-wide program for unclassified cloud computing that standardizes:

- Security assessment
- Authorization
- Continuous monitoring for cloud products and services
- https://www.fedramp.gov/about-us/about/

There are three main players in the FedRAMP process:

- Agencies
- Cloud service providers (CSPs)
- Third party assessment organizations (3PAOs)

FedRamp Authorization Playbook is the starting point

- https://www.fedramp.gov/introducing-the-new-agency-authorization-playbook/

Relevant Security Documentation for FedRAMP

FIPS Publication 199 Standards for Security Categorization of Federal Information and Information Systems

FIPS Publication 200 Minimum Security Requirements for Federal Information and Information Systems

NIST 800-53 Security Controls Catalog, revision 4

FIPS Publication 199

Defines three levels of potential impact on organizations or individuals should there be a breach of security (i.e. a loss of confidentiality, integrity, or availability).

LOW impact if the loss of confidentiality, integrity, or availability could be expected to have a limited adverse effect on organizational operations, organizational assets, or individuals.

MODERATE impact if the loss of confidentiality, integrity, or availability could be expected to have a serious adverse effect on organizational operations, organizational assets, or individuals.

HIGH impact if the loss of confidentiality, integrity, or availability could be expected to have a severe or catastrophic adverse effect on organizational operations, organizational assets, or individuals.

Security Categorization:

SC(system)={(**confidentiality**, impact), (**integrity**, impact), (**availability**, impact)}

FIPS Publication 200

Identifies seventeen security-related areas with regard to protecting the confidentiality, integrity, and availability of federal information systems and the information processed, stored, and transmitted by those systems.

1. access control
2. awareness and training
3. audit and accountability
4. certification, accreditation, and security assessments
5. configuration management
6. contingency planning
7. identification and authentication
8. incident response
9. maintenance
10. media protection
11. physical and environmental protection
12. planning
13. personnel security
14. risk assessment
15. systems and services acquisition
16. system and communications protection
17. system and information integrity

Examples of FedRAMP Cloud Service Providers (CSPs)

Provider	Service Model Supported	Impact Level	Authorizations
AWS US East/West	IaaS	Moderate	83
AWS GovCloud	IaaS	Moderate	39
AWS GovCloud High	IaaS, PaaS	High	8
Google G Suite	PaaS, SaaS	Moderate	10
Google Services (Google Cloud Platform Products)	IaaS, PaaS, SaaS	Moderate	0
Microsoft Commercial Cloud	IaaS, PaaS	Moderate	56
Microsoft Azure Government	IaaS, PaaS	High	15
Microsoft 365 Multi-Tenant & Supporting Services	SaaS	Moderate	33

Service-Level Agreements

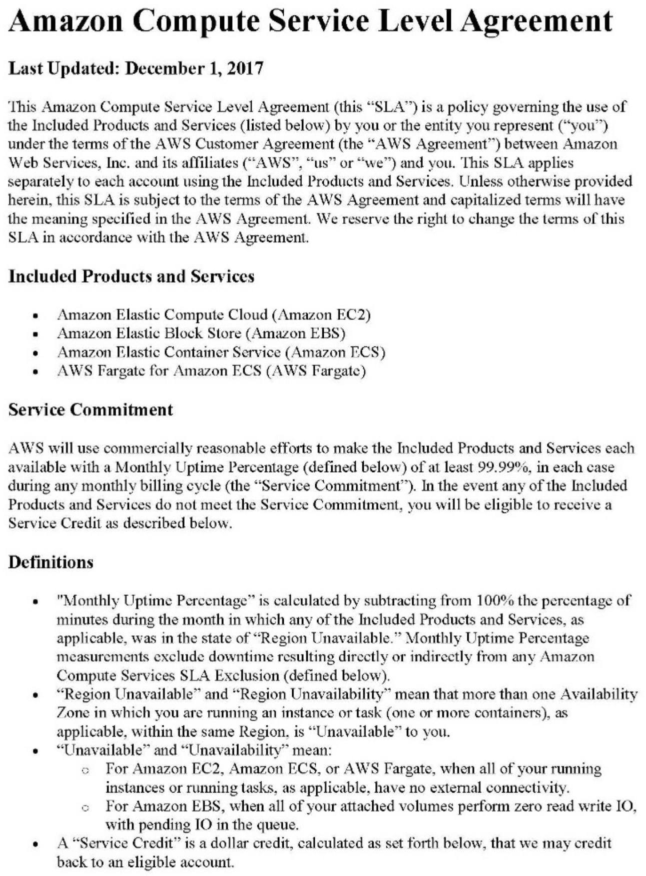
A service level agreement (SLA) is a formal negotiated agreement (contract) between service consumers and providers.

Minimal SLA outline

- Parties in the agreement
- Services provided that are covered by the SLA
- Service performance metrics
- Incident handling — procedures, response times, consequences when response times are not met
- Records/logs to keep
- Performance review and problem management
- Termination arrangements

Each CSP has their own SLA.

Example: Amazon Compute SLA



Amazon Compute Service Level Agreement

Last Updated: December 1, 2017

This Amazon Compute Service Level Agreement (this "SLA") is a policy governing the use of the Included Products and Services (listed below) by you or the entity you represent ("you") under the terms of the AWS Customer Agreement (the "AWS Agreement") between Amazon Web Services, Inc. and its affiliates ("AWS", "us" or "we") and you. This SLA applies separately to each account using the Included Products and Services. Unless otherwise provided herein, this SLA is subject to the terms of the AWS Agreement and capitalized terms will have the meaning specified in the AWS Agreement. We reserve the right to change the terms of this SLA in accordance with the AWS Agreement.

Included Products and Services

- Amazon Elastic Compute Cloud (Amazon EC2)
- Amazon Elastic Block Store (Amazon EBS)
- Amazon Elastic Container Service (Amazon ECS)
- AWS Fargate for Amazon ECS (AWS Fargate)

Service Commitment

AWS will use commercially reasonable efforts to make the Included Products and Services each available with a Monthly Uptime Percentage (defined below) of at least 99.99%, in each case during any monthly billing cycle (the "Service Commitment"). In the event any of the Included Products and Services do not meet the Service Commitment, you will be eligible to receive a Service Credit as described below.

Definitions

- "Monthly Uptime Percentage" is calculated by subtracting from 100% the percentage of minutes during the month in which any of the Included Products and Services, as applicable, was in the state of "Region Unavailable." Monthly Uptime Percentage measurements exclude downtime resulting directly or indirectly from any Amazon Compute Services SLA Exclusion (defined below).
- "Region Unavailable" and "Region Unavailability" mean that more than one Availability Zone in which you are running an instance or task (one or more containers), as applicable, within the same Region, is "Unavailable" to you.
- "Unavailable" and "Unavailability" mean:
  - For Amazon EC2, Amazon ECS, or AWS Fargate, when all of your running instances or running tasks, as applicable, have no external connectivity.
  - For Amazon EBS, when all of your attached volumes perform zero read write IO, with pending IO in the queue.
- A "Service Credit" is a dollar credit, calculated as set forth below, that we may credit back to an eligible account.

https://aws.amazon.com/ec2/sla/

Definitions and Fundamental Concepts

In this module, we discussed

- What makes cloud computing different from a typical data center
- Cloud service models
- Cloud delivery models
- Cloud options available for US government systems
- Security controls
- Service level agreements

Cloud Computing: An Architecture-centric View

Enabling Technologies

Enabling Technologies

In this module, we will discuss

- What is virtualization and how it enables cloud computing
- How virtual servers are different from physical servers
- What are containers and how they support cloud computing
- How virtual machines are managed using scripts

Focusing our discussion

For much of the rest of this course, we are going to focus on Amazon’s IaaS technology – Amazon Web Services or AWS

Why IaaS?

- Our experience is that IaaS is the starting point for many system migrations to the cloud
- Understanding IaaS provides the necessary foundation to understand other cloud services - PaaS and SaaS are built on top of IaaS
- Amazon’s IaaS is starting to bleed into PaaS and SaaS

Why Amazon?

- Market leader in commercial and government sectors
- Broad offering, covers diverse capabilities
- Other vendors map their offerings to Amazon’s

How do they do it?

How does a cloud service provider deliver Infrastructure as a Service?

How do they achieve elasticity and on-demand capacity?

How much do you need to care about it?

Virtualization

NIST definition (800-125)

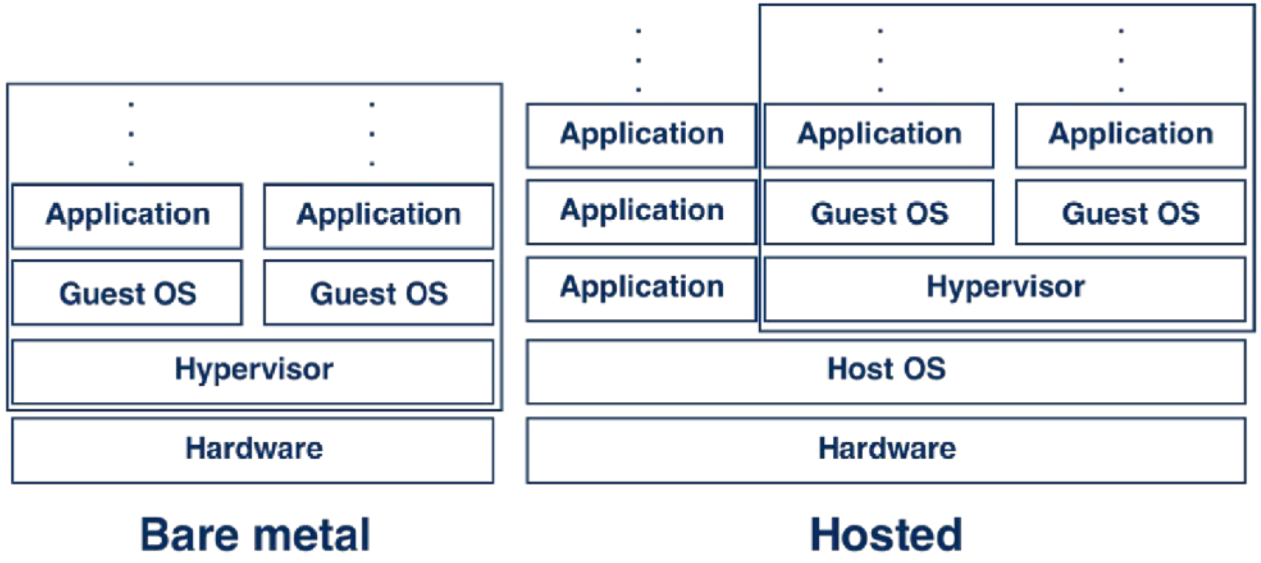
- *Virtualization* is the simulation of the software and/or hardware upon which other software runs.

Types of virtualization:

- Application – e.g., Java Virtual Machine
- Operating system – e.g., containers like Docker
- Full – One or more operating systems (and their applications) running on top of virtual hardware

We’ll talk about Full Virtualization first, and then come back to Containers

Types of Full Virtualization



Type 1

Type 2
(mostly used on Desktop)

Source: NIST 800-125

Virtualization Influences Deployment Partitioning

With physical servers:

- Deploy multiple applications on a physical server – introduces dependency management concerns
- Efficiency → Fill the server's capacity (while maintaining some reserve headroom)
- Physical failure may be a concern, i.e. don't deploy the primary and backup to the same physical server

With virtualized servers:

- Simplify dependencies – deploy one application per VM instance
- Efficiency of physical hardware utilization is the cloud provider's concern
- Physical hardware failure is (mostly) handled by the cloud service provider – we'll talk later about deployment patterns to improve availability

Virtualization and the Cloud

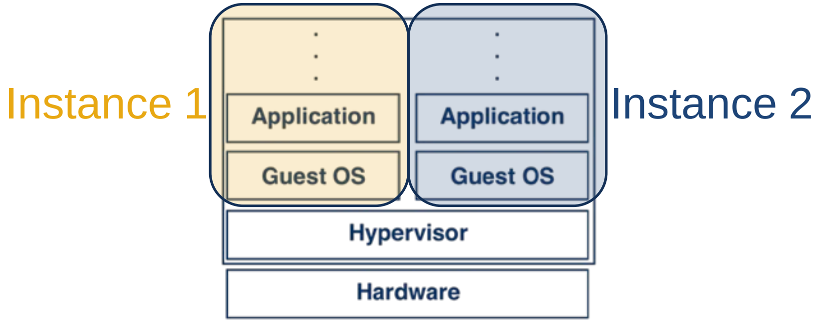
Cloud Service Providers use Type 1 virtualization

AWS used the Xen hypervisor, now moving to KVM-based implementation*

Physical reboots are a very rare event

Instance = executing guest OS + application (and middleware)

Multi-tenant – Instances on same physical server may belong to different users



* https://www.theregister.co.uk/2017/11/07/aws_writes_new_kvm_based_hypervisor_to_make_its_cloud_go_faster/

Images and Instances

An *instance* is a deployed and executing *image.*

- An image can be used to create multiple instances.

How are images created?

- Start with a *base image* – this is a minimal bootable guest OS image
- Deploy and start the base image
- Install more software (middleware, application, etc.) on the running instance
- Configure and tune the running instance (users, firewall, application settings)
- Take a *snapshot* of the instance to create a new image

We'll talk more this later – Infrastructure as Code

A VM instance is not like a traditional physical server – Storage

The disk on a physical server retains state from one boot to the next boot

- Not necessarily the case in the cloud

Boot volume (AWS EC2):

- Instance Store-backed
  - Ephemeral, data is not saved on shutdown, next boot is from clean image
  - Slower to start (in EC2)
- EBS Store-backed
  - Persistent, behaves like physical server boot disk
  - Faster to start (in EC2)
  - Incurs storage charges even when instance is not running

We don’t back up virtual servers – the image is the backup

A VM instance is not like a traditional physical server – Networking, Configuration, Access

Networking

- VM instances are assigned dynamic hostnames and IP addresses – there are no static IP addresses in the cloud
- Architectures must use discovery instead of static configuration

We can pass configuration variables to an instance when we start it.

- E.g., role=master or role=slave

Your only access is via ssh - you get the instance’s key when you launch it. **Don’t Lose That Key!**

Instances and Physical Hardware

The cloud service provider manages allocation of instance to physical nodes

Most cloud service providers offer several types of instance profiles

- CPU and memory capabilities
- Hypervisor tuning
- Network and storage

Each profile has a different pay-per-use cost

Profiles change over time as technology evolves

Example – Survey of Instance Types* in Amazon Elastic Compute Cloud (EC2)

General Purpose

- 4 subtypes, various sizes (23 total)
- Balance CPU, Memory, I/O

Compute Optimized

- 3 subtypes, 1 w/ SSD (16 total)
- High-end CPUs, variable memory sizes

Memory Optimized

- 4 subtypes, 3 w/ SSD (19 total)
- Up to 3,905GiB memory

Accelerated Computing

- 4 subtypes (11 total)
- GPU and FPGA

Storage Optimized

- 3 subtypes, HDD and SSD (15 total)
- High instance storage for replicated databases

*As of 1 Dec 2017

Containers

A VM image contains a full guest operating system

- May take 30-45 seconds to start, possibly longer depending on the time to copy the image from storage

What if my application doesn't need all of the services that the OS provides? E.g., Microservices or a Function-as-a-Service

An *Application Container** is a construct designed to package and run an application or its components running on a shared Operating System.

Containers are "lightweight" - <50 msec startup time, small enough to cache locally

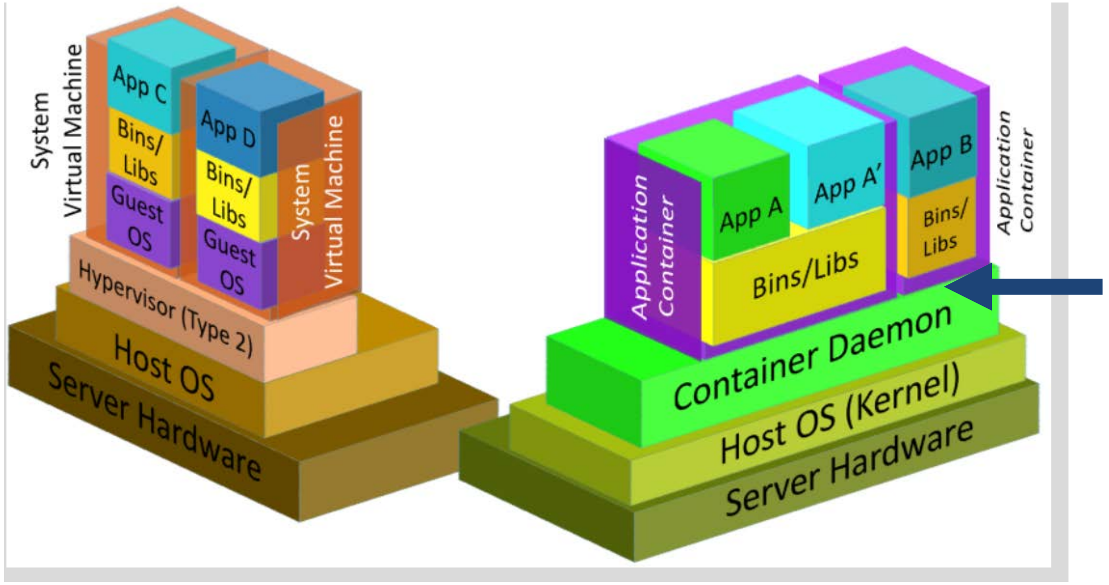
- Based on Linux kernel namespaces and cgroups
- Less robust isolation than VM provides, but enough for most use cases

Some similarities to VMs - boot from image, storage is ephemeral

Some differences – Images can be composed, networking is bridged through host's IP address

*From NIST 800-180 Draft

Container Compared to Full Virtualization



E.g., Docker

And, of course, you can run your container daemon on a guest OS in a VM

Source: NIST 800-180 Draft

Container Technology

This technology space can be confusing, because containers are being applied for both desktop and server use cases

Docker was emerging as the leading container engine (docker.org) for both cases, although recent business decisions have created some concerns

Desktop Use Case

- Don’t install applications or runtimes, instead run software in a container
- Especially useful if you need multiple versions of a runtime

Server Use Case

- Small, fast deployable units
- Fine-grained scalability

Containers on the Desktop
(This is not directly related to cloud computing)

Install Docker Engine

```
$ sudo apt-get install docker.io
```

Cache base images*

```
$ docker pull python:2.7
$ docker pull python:3.3
$ docker pull python:3.4
```

Execute for each Python version

```
$ docker run -i -t --rm python:2.7 python -m timeit "[i for i in range(1000)]"
10000 loops, best of 3: 82.2 usec per loop

$ docker run -i -t --rm python:3.3 python -m timeit "[i for i in range(1000)]"
10000 loops, best of 3: 83 usec per loop

$ docker run -i -t --rm python:3.4 python -m timeit "[i for i in range(1000)]"
10000 loops, best of 3: 87.7 usec per loop
```

*** Optimization – Docker will automatically pull on first use of an image if it is not cached locally**

Example from http://tiborsimko.org/docker-for-python-applications.html

Server-side Containers

Driven by microservices (a small, cohesive, independently deployable distributed service developed by a single team)

Applications have many (i.e. 10s) of microservices, with some executing multiple instances

Concerns

- Packaging dependencies
- Deployment efficiency (100s of instances)

Enter containers and container orchestration technology

- Docker container engine
- Kubernetes (“K8s”*) container management

Containers enable the *function as a service*, AKA *serverless* architecture style

* But only if you are a rock star full stack ninja developer

Creating VM Instances

Amazon Web Service (AWS) homepage has 10-minute Tutorial: Launch a Linux Virtual Machine using Amazon EC2

- Uses the AWS Management Console
- Wizard-driven VM instance creation – step through a few screens to configure and launch the instance
- Console shows the status of your running instances
- Great way to get started with AWS!

But this approach is not viable for more than a few instances

- Manual and error-prone
- Slow

Automate all the things – treat your infrastructure as code

Automate all the Things – Infrastructure as Code

Infrastructure as code is the process and technology to manage and provision computers and networks (physical and/or virtual) through scripts.

Scripts/code provide:

- Scale
- Automation
- Version control

Technology support for infrastructure as code

AWS Command Line Interface and language-specific libraries

- Wraps the AWS API – use your favorite scripting tools (shell script, Python, Ruby, ...)
- Fine-grained and detailed control
- Can do more than just manage VM instances
  - Manage images, manage storage and snapshots, ad hoc operations on services like DynamoDB and Identity and Access Management (IAM)

DevOps tools like Chef and Puppet use higher-level abstractions, make things easier and more efficient

Chef Script Examples (Chef scripts use Ruby)

```
httpd_service 'an websites' do
       instance_name 'bob'
       servername 'www.computers.biz'
       version '2.4'
       mpm 'event'
       threadlimit '4096'
       listen_ports ['1234']
       action :create
       action :start
end

mysql_service 'foo' do
       port '3306'
       version '5.5'
       initial_root_password 'change me'
       action [:create, :start]
end
```

Examples from https://github.com/chef-cookbooks/httpd and https://github.com/chef-cookbooks/mysql

Immutable/Versioned Infrastructure

Infrastructure as code promotes an IT operations approach called *immutable infrastructure*

- Immutable – “write once”
- Don’t update, recreate (or replace)

Don’t patch a running system, instead

- Rework the infrastructure as code scripts that generated the image
- Create a new image
- Test instances of the new image
- Deploy the new image to production

Allows us to version our infrastructure

- Rollback – some large-scale systems can’t be tested outside of the production environment – Infrastructure as Code and versioned infrastructure provide a safety net for testing in these situations
- Parity – test and production environments are identical

Infrastructure as Code - Takeaways

You need to be familiar with both approaches:

- Chef/Puppet/etc. – Fast, easy, default development tools
- AWS Command Line Interface – finer-grained control and visibility for T&E activities

Contractors should deliver their infrastructure as code artifacts

- Treat these like any other software deliverable
- It is code – some up-front design is usually needed to define approach and overall structure
- It is code – some documentation is needed to describe the artifacts

Key to agility

- Versioned infrastructure provides a safety net for rapid exploration and experimentation

Repeatability reduces implementation diversity

One more thing – Network Virtualization and Virtual Private Clouds

VMs provide isolation when sharing physical computer hardware

What about sharing the network?

A *virtual private cloud* or VPC uses private subnet addresses and VLAN technology to isolate network traffic between VMs

- When a VM is launched, it is assigned to a VPC
- Some CSPs (e.g., AWS) allow you to also purchase physical hardware isolation – VMs deployed to a VPC will not share physical hardware with VMs outside that VPC

Amazon also allows you to pay to place a VPN endpoint in the VPC

- Allows extending the enterprise network directly into the cloud for hybrid service delivery

Enabling Technologies

In this module, we discussed

- What is virtualization and how it enables cloud computing
- How virtual servers are different from physical servers
- What are containers and how they support cloud computing
- How virtual machines are managed using scripts

Cloud Computing: An Architecture-centric View

Cloud Native Services

Cloud Native Services

In this module, we will discuss

- Cloud platforms include many out-of-the-box services
- Architectures can trade off cloud native vs. portable implementations
- Impact on testing/assurance approach

Lift and shift to the cloud?

```
<lift-and-shift>
 1. Package each of your servers into a virtual machine image
 2. Choose a cloud service provider
 3. Select appropriate instance types
 4. Deploy your VM images
</lift-and-shift>
```

Done? Not quite!

```
<remediation>
 1. Persistent storage
 2. Static IP addresses
 3. …
</remediation>
```

The case for cloud native services

Scalable, secure, and highly-available distributed services are hard

- PostgreSQL has 270 configuration parameters
- Kafka message queue has 140 “top-level” configuration parameters
- How many impact security? performance? availability?

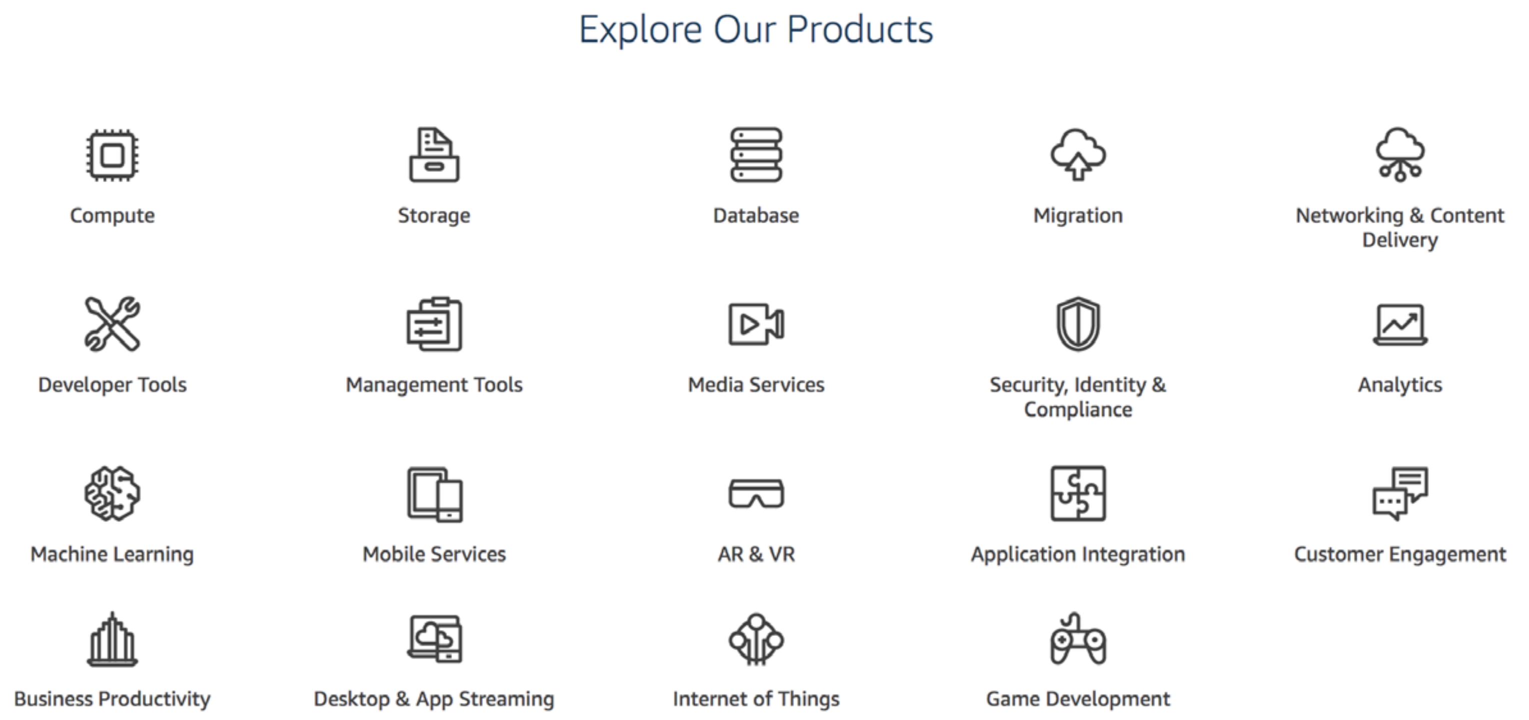
Managing distributed services is hard

- Patching and updating is harder in distributed system
- Monitoring
- Adding capacity to a running system
- …

Wouldn’t it be nice if this was somebody else’s problem?

Cloud Native Services to the rescue!

AWS Cloud Native Services


Explore Our Products

Compute
Storage
Database
Migration
Networking & Content Delivery
Developer Tools
Management Tools
Media Services
Security, Identity & Compliance
Analytics
Machine Learning
Mobile Services
AR & VR
Application Integration
Customer Engagement
Business Productivity
Desktop & App Streaming
Internet of Things
Game Development

Cloud Native Services – Annotation Key

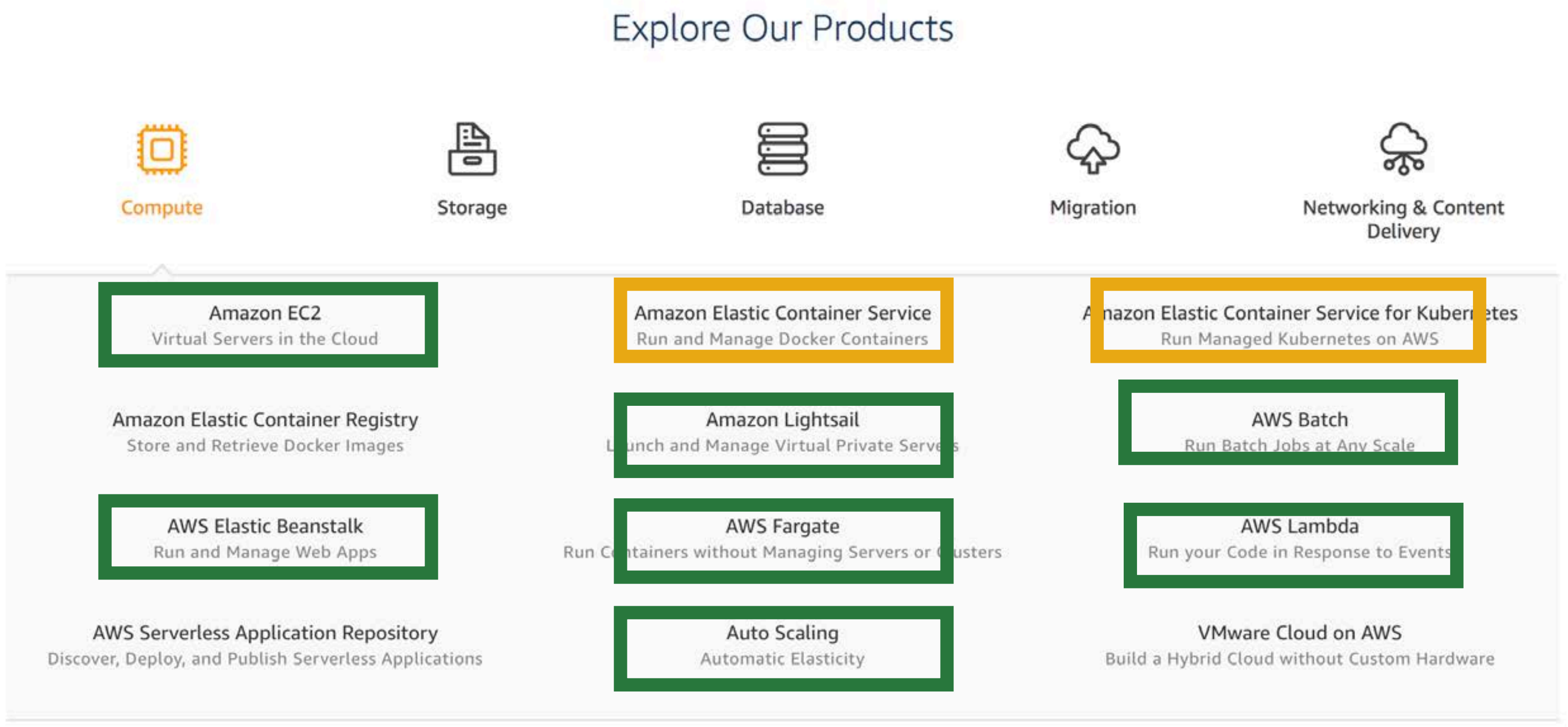
Replaces a traditional, portable component
(You could build this yourself in the cloud)

Only cloud service provider can feasibly deliver this service

There are some judgement calls here.

Note that we don't categorize every service offering.

AWS Cloud Native Services – Compute



Explore Our Products

Amazon EC2
Virtual Servers in the Cloud

Amazon Elastic Container Service
Run and Manage Docker Containers

Amazon Elastic Container Service for Kubernetes
Run Managed Kubernetes on AWS

Amazon Elastic Container Registry
Store and Retrieve Docker Images

Amazon Lightsail
Launch and Manage Virtual Private Servers

AWS Batch
Run Batch Jobs at Any Scale

AWS Elastic Beanstalk
Run and Manage Web Apps

AWS Fargate
Run Containers without Managing Servers or Clusters

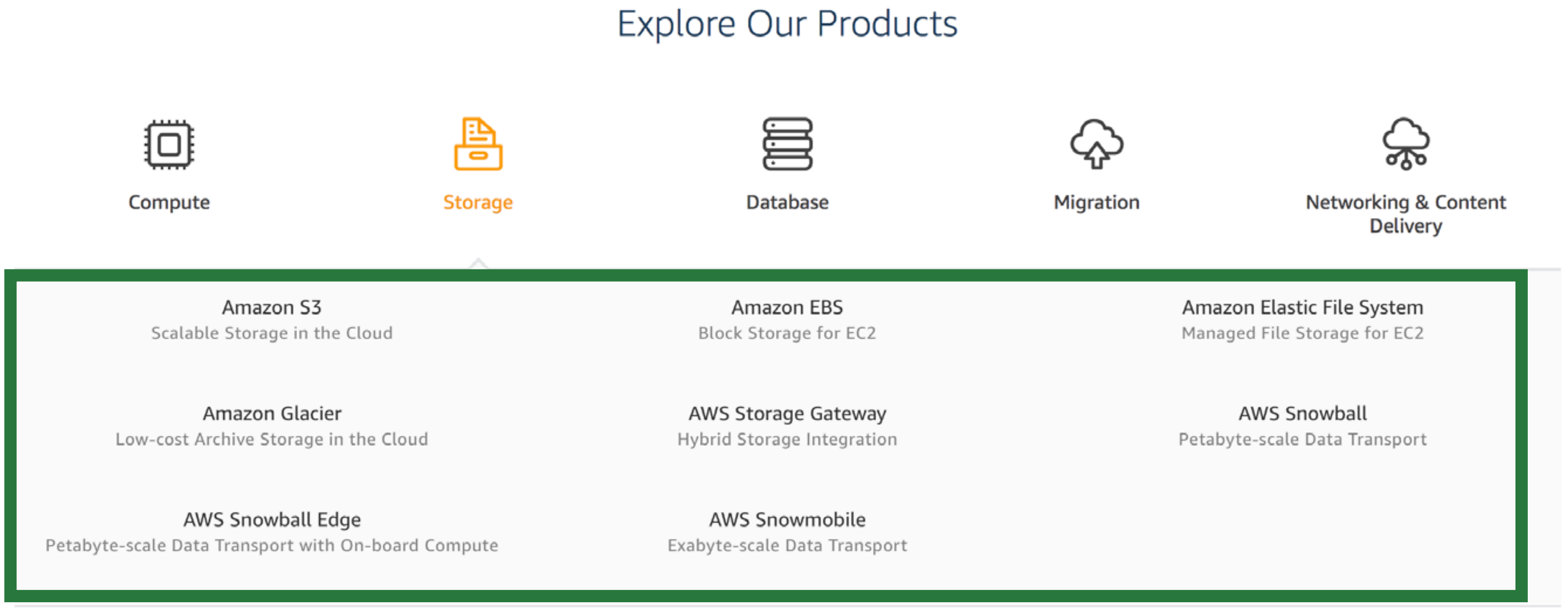
AWS Lambda
Run your Code in Response to Events

AWS Serverless Application Repository
Discover, Deploy, and Publish Serverless Applications

Auto Scaling
Automatic Elasticity

VMware Cloud on AWS
Build a Hybrid Cloud without Custom Hardware

AWS Cloud Native Services – Storage



Explore Our Products

Compute | Storage | Database | Migration | Networking & Content Delivery

Amazon S3
Scalable Storage in the Cloud

Amazon EBS
Block Storage for EC2

Amazon Elastic File System
Managed File Storage for EC2

Amazon Glacier
Low-cost Archive Storage in the Cloud

AWS Storage Gateway
Hybrid Storage Integration

AWS Snowball
Petabyte-scale Data Transport

AWS Snowball Edge
Petabyte-scale Data Transport with On-board Compute

AWS Snowmobile
Exabyte-scale Data Transport

Storage – Seems like a lot of options!

Basics:

- EBS – Elastic Block Store – the virtual hard disks for your VM
  - An EBS volume can be mounted by only one VM instance at a time
  - Size limited to 16TB per volume
  - Can be backed up/snapshot'ed in case of application crash
- EFS – Elastic File System – NFS in the cloud
  - Distributed file system, can be mounted by many VMs at a time
  - No size limits
  - Managed by AWS
- S3 – Simple Storage Service – object (blob) storage
  - Access via API or via http (can use to host static web content)
  - Virtually unlimited scale (both objects and buckets/namespaces)
  - Managed by AWS

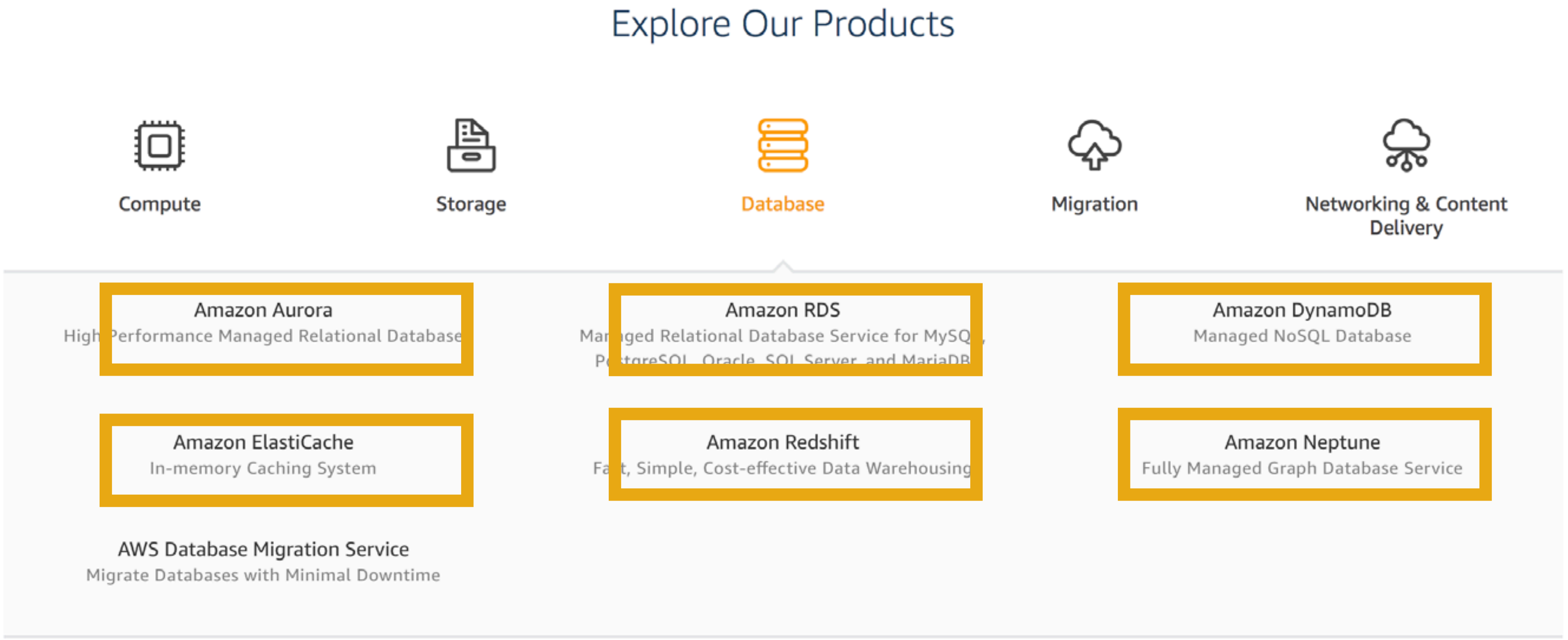
Storage

"Advanced":

- Glacier – low cost cold storage
- Storage Gateway – hybrid cloud storage solution
- Snowball and Snowmobile – peta-/exa-scale transport and storage (i.e. sneakernet)

- Snowball Edge – Onboard ingest processing

AWS Cloud Native Services – Database



Explore Our Products

Amazon Aurora
High Performance Managed Relational Database

Amazon RDS
Managed Relational Database Service for MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB

Amazon DynamoDB
Managed NoSQL Database

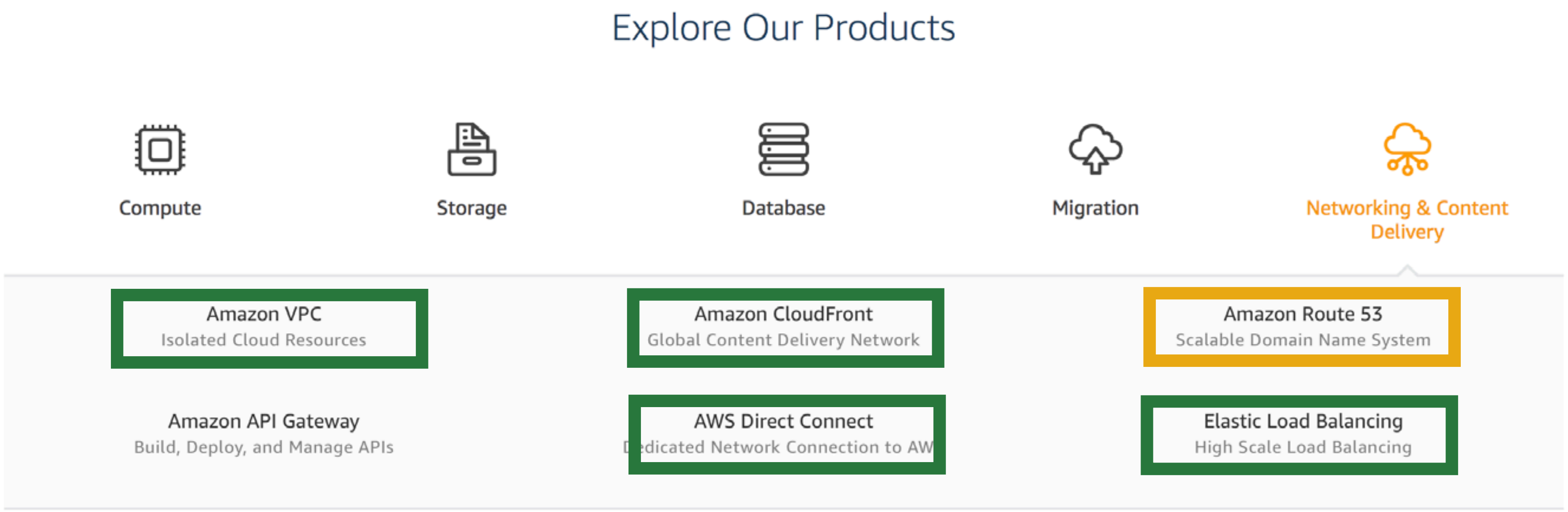
Amazon ElastiCache
In-memory Caching System

Amazon Redshift
Fast, Simple, Cost-effective Data Warehousing

Amazon Neptune
Fully Managed Graph Database Service

AWS Database Migration Service
Migrate Databases with Minimal Downtime

AWS Cloud Native Services – Networking



Explore Our Products

Amazon VPC
Isolated Cloud Resources

Amazon CloudFront
Global Content Delivery Network

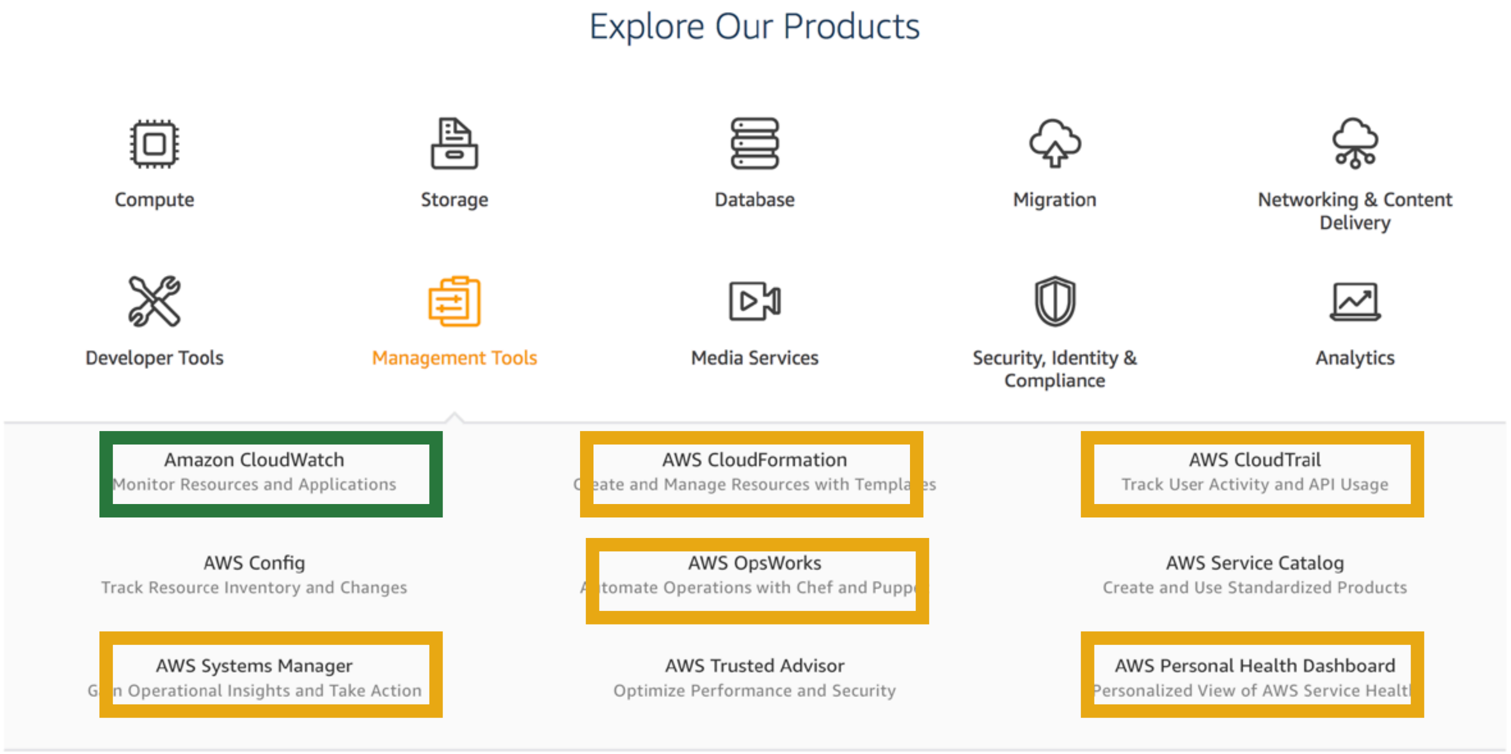
Amazon Route 53
Scalable Domain Name System

Amazon API Gateway
Build, Deploy, and Manage APIs

AWS Direct Connect
Dedicated Network Connection to AW

Elastic Load Balancing
High Scale Load Balancing

AWS Cloud Native Services – Management



Explore Our Products

Amazon CloudWatch
Monitor Resources and Applications

AWS CloudFormation
C eate and Manage Resources with Templa es

AWS CloudTrail
Track User Activity and API Usage

AWS Config
Track Resource Inventory and Changes

AWS OpsWorks
A tomate Operations with Chef and Pupp

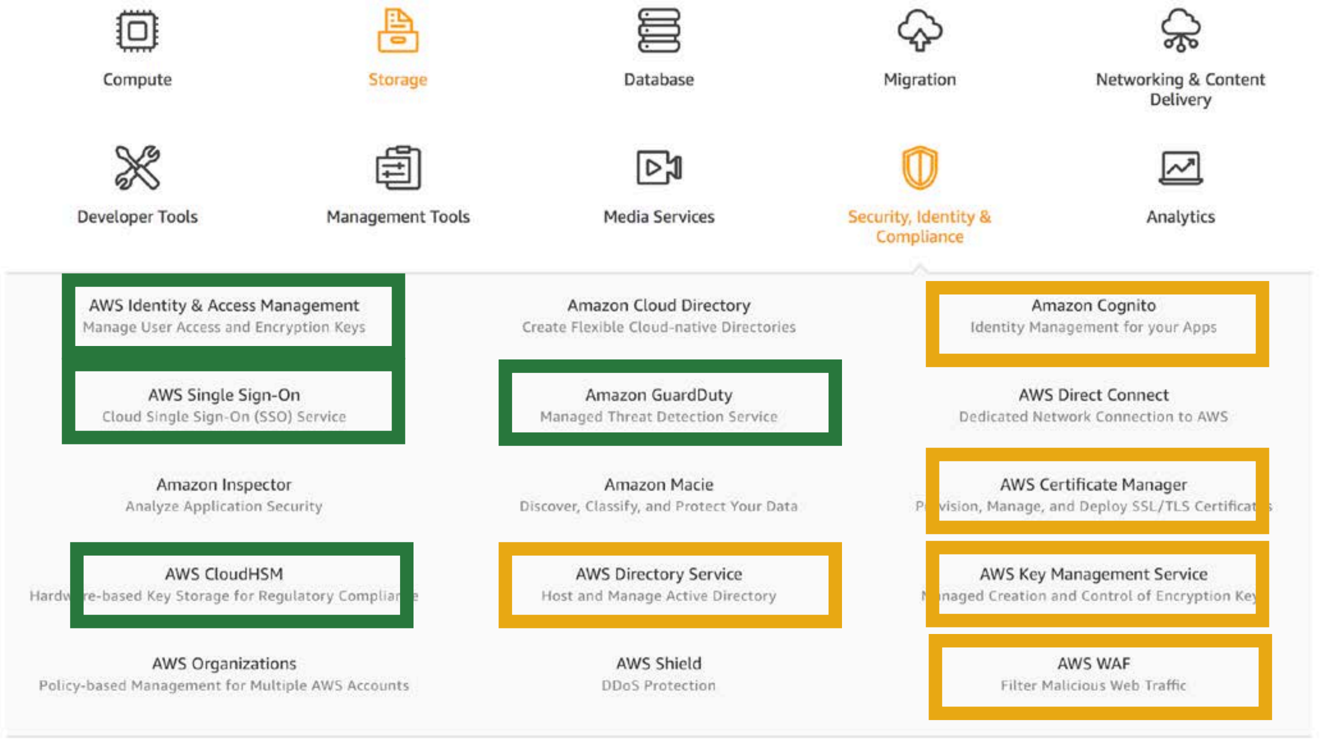
AWS Service Catalog
Create and Use Standardized Products

AWS Systems Manager
G n Operational Insights and Take Action

AWS Trusted Advisor
Optimize Performance and Security

AWS Personal Health Dashboard
Personalized View of AWS Service Healt

AWS Cloud Native Services – Security



AWS Identity & Access Management
Manage User Access and Encryption Keys

Amazon Cloud Directory
Create Flexible Cloud-native Directories

Amazon Cognito
Identity Management for your Apps

AWS Single Sign-On
Cloud Single Sign-On (SSO) Service

Amazon GuardDuty
Managed Threat Detection Service

AWS Direct Connect
Dedicated Network Connection to AWS

Amazon Inspector
Analyze Application Security

Amazon Macie
Discover, Classify, and Protect Your Data

AWS Certificate Manager
Pr vision, Manage, and Deploy SSL/TLS Certificat s

AWS CloudHSM
Hardw re-based Key Storage for Regulatory Complian e

AWS Directory Service
Host and Manage Active Directory

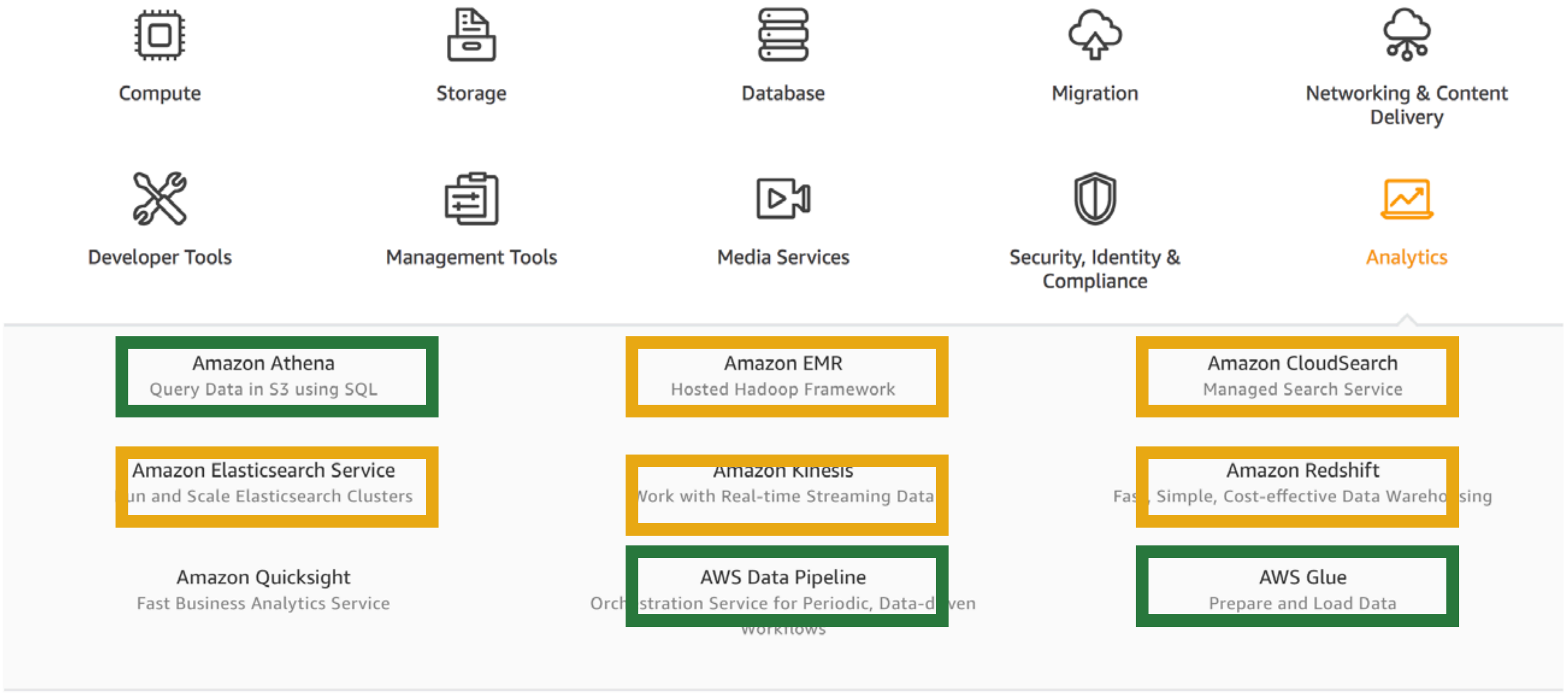
AWS Key Management Service
M naged Creation and Control of Encryption Key

AWS Organizations
Policy-based Management for Multiple AWS Accounts

AWS Shield
DDoS Protection

AWS WAF
Filter Malicious Web Traffic

AWS Cloud Native Services – Analytics



Compute

Storage

Database

Migration

Networking & Content Delivery

Developer Tools

Management Tools

Media Services

Security, Identity & Compliance

Analytics

Amazon Athena
Query Data in S3 using SQL

Amazon EMR
Hosted Hadoop Framework

Amazon CloudSearch
Managed Search Service

Amazon Elasticsearch Service
un and Scale Elasticsearch Clusters

Amazon Kinesis
Vork with Real-time Streaming Data

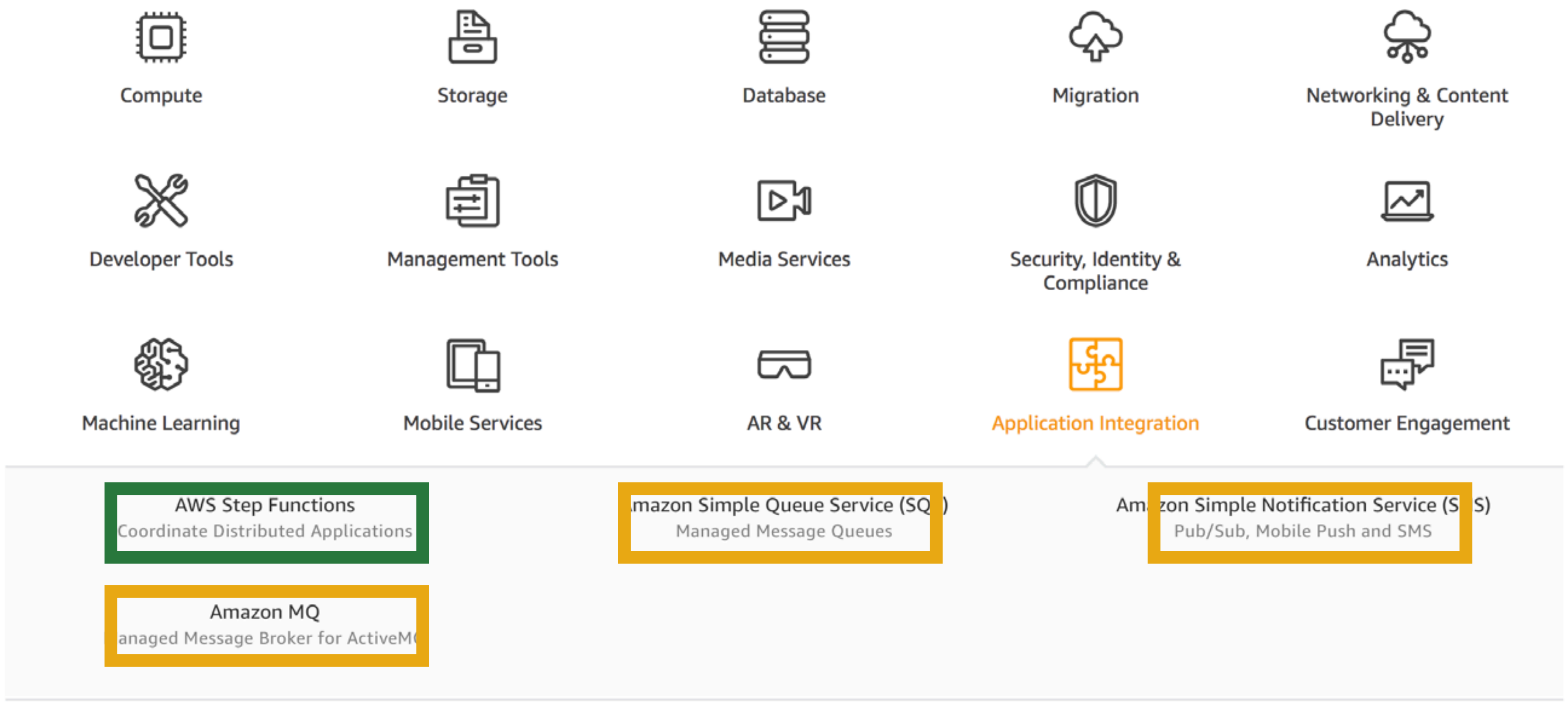
Amazon Redshift
Fas Simple, Cost-effective Data Wareho sing

Amazon Quicksight
Fast Business Analytics Service

AWS Data Pipeline
Orch stration Service for Periodic, Data-d ven Workflows

AWS Glue
Prepare and Load Data

AWS Cloud Native Services – Integration



Compute

Storage

Database

Migration

Networking & Content Delivery

Developer Tools

Management Tools

Media Services

Security, Identity & Compliance

Analytics

Machine Learning

Mobile Services

AR & VR

Application Integration

Customer Engagement

AWS Step Functions
Coordinate Distributed Applications

mazon Simple Queue Service (SQ)
Managed Message Queues

Am zon Simple Notification Service (S S)
Pub/Sub, Mobile Push and SMS

Amazon MQ
anaged Message Broker for ActiveM

Hey, what about the other CSPs?

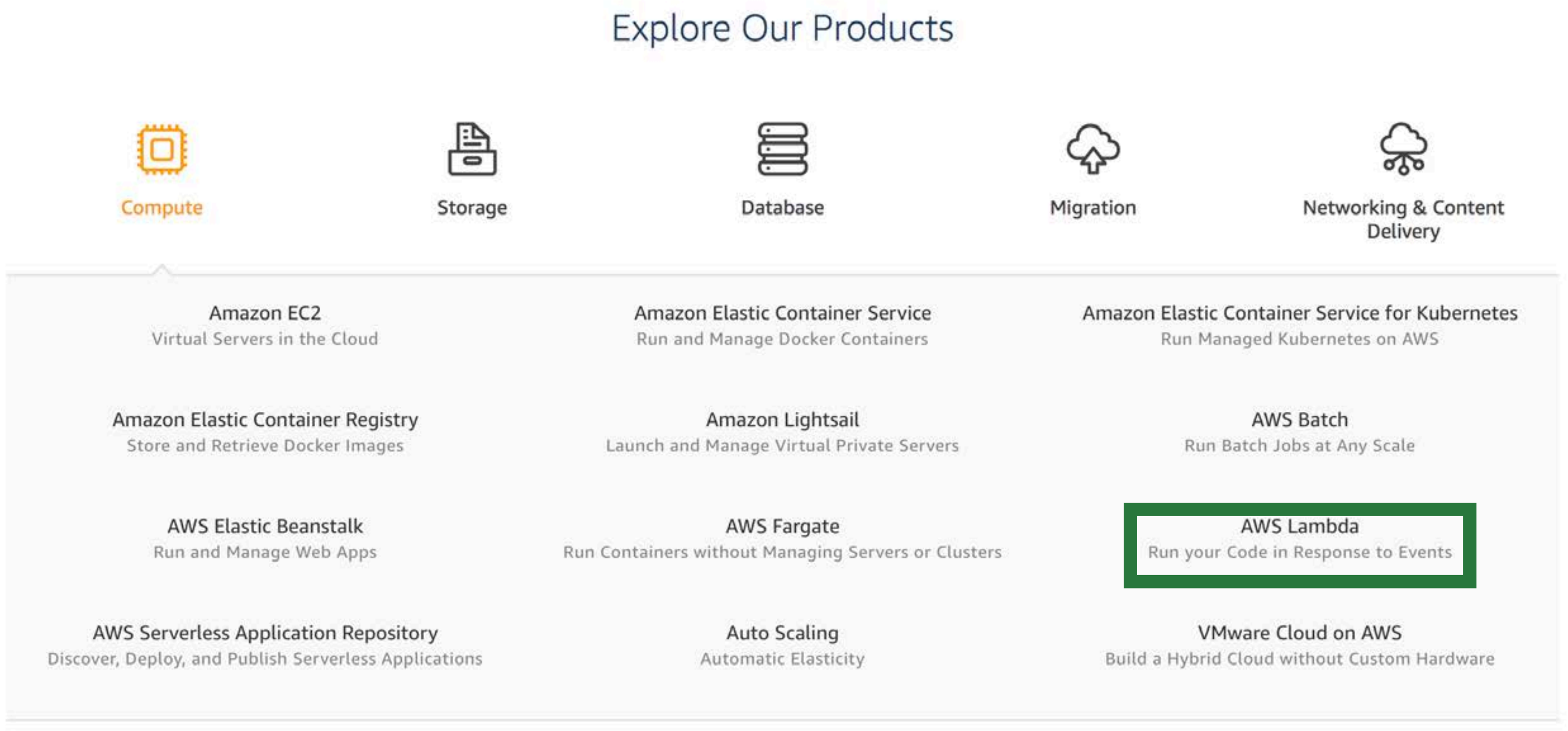
Microsoft Azure:

http://aka.ms/awsazureguide maps from AWS services to Microsoft Azure services

Google Compute Platform (GCP):

https://cloud.google.com/free/docs/map-aws-google-cloud-platform maps from AWS services to GCP services

Function as a Service – FaaS, a.k.a. “Serverless”


Explore Our Products

Amazon EC2 Virtual Servers in the Cloud	Amazon Elastic Container Service Run and Manage Docker Containers	Amazon Elastic Container Service for Kubernetes Run Managed Kubernetes on AWS
Amazon Elastic Container Registry Store and Retrieve Docker Images	Amazon Lightsail Launch and Manage Virtual Private Servers	AWS Batch Run Batch Jobs at Any Scale
AWS Elastic Beanstalk Run and Manage Web Apps	AWS Fargate Run Containers without Managing Servers or Clusters	AWS Lambda Run your Code in Response to Events
AWS Serverless Application Repository Discover, Deploy, and Publish Serverless Applications	Auto Scaling Automatic Elasticity	VMware Cloud on AWS Build a Hybrid Cloud without Custom Hardware

Fine-grained virtualization

VM – run OS on virtualized hardware

Container – run process on virtualized OS

What if I want to just run a function?

Function as a Service, or “Serverless”

- Pack up up your function code and dependencies (i.e. libraries)
- Upload the zip file to the CSP and bind it to a REST endpoint
- When the endpoint is invoked, the CSP creates a container and runs your function, passing the in the parameters from the REST invocation
- Pay per use, based on execution duration and memory utilization

FaaS Limitations and Options

Concurrency – Autoscaling

- Specify the number of concurrent instances when you bind to the REST endpoint
- CSP sets upper limit

Startup latency – “cold start”

- Delay in launching the container on the initial *concurrent* invocation
- Container is not unloaded immediately on function exit – remains for a few seconds
- Keep-alive: Send dummy invocations to keep the function “warm” – trades off cost against lower latency

FaaS Architecture Style

Extends "stateless service" style, e.g., request context is passed in by client

Return object to client in response

Any service state and data must be stored using a cloud native service (e.g., DynamoDB or S3)

Composition

- Client orchestrates invocations to multiple functions
- Nesting – a function synchronously invokes other services
  - Need to complete within the execution time limits for the initially invoked service
  - Return response to client
- Chaining – a function asynchronously invokes another service
  - Avoids execution time limits
  - Can't return a response to the client

Cutting through some of the Serverless hype

Benefits:

- For some workloads, FaaS pay-per-use cost will be less than other approaches
- For some applications, FaaS will be simpler to develop
- Some consider this to be “DevOps as a Service” – it pushes many Site Reliability Engineering concerns to the CSP, and may reduce full-stream development costs
- Enforces stateless architecture style, which improves scalability

Challenges:

- Can be difficult to debug
- Discontinuity in evolution if you reach the complexity or execution time limits of FaaS
- Use of cloud native services will inhibit portability (may not be a concern)

How do you choose whether to implement your own or use a cloud native service?

Development cost

- Probably lower if you start design to use cloud native service
- Obviously higher if you have to rework to use cloud native service

Pay per use cost

- <u>For a given scale</u>, cloud native services are usually more expensive
- Most cloud native services offer autoscaling or easy manual scaling

Service management cost

- Cloud services need no tuning, patching, updating, …
- Harder to quantify – what does it cost to manage your own service?

Security posture

- Cloud native services may be more secure than a self-implemented solution hosted in the cloud
- Cloud native services may already be accredited
- Again, hard to quantify

Test/Assurance Implications

1. Understand where cloud native services are being used
   - You need to look at the architecture/design to see this
2. Research the weaknesses, common misuse patterns, and limitations of each native service
   - Netflix engineering blog is one source for AWS
   - Lots of stories in the blogosphere
3. Test autoscaling, failover, access control configuration, …
   - You are more likely to find problems with application’s use of the service than the service implementation
   - We’ll talk more later about testing
4. Test carefully to avoid unintended side-effects
   - See the case studies that follow here

But before the case studies, a note on terminology

“Partition” has multiple meanings in the context of cloud computing

Verb, e.g., *network partition*

- Cause the network to split into two or more subnetworks that cannot communicate with each other
- This is the **P** in CAP

Noun, e.g., *database partition*

- In a distributed database, the complete data set is divided and each division may be copied. Each of these subsets is called a *partition.*
- Partitions are assigned to physical nodes, where they are stored.

Case study* – accidentally triggering performance throttling

System used AWS DynamoDB, a key-value distributed database service

DynamoDB hashes the key to select a partition to store the value

- Hashing function balances data across storage partitions

Service pricing is based on peak I/O for a partition

- Service throttles all accesses when you hit your I/O limit in *any* partition

Test script:

```
for value = 1 to 1000000
  store(”key”, value)
end
```

What’s wrong with that?

*A. Roussel and R. Branson. *The Million Dollar Engineering Problem* [Online]. https://segment.com/blog/the-million-dollar-eng-problem

Case study – accidentally triggering performance throttling

Note that the key never varies

- Every write operation is hashed to the same storage partition
- Tight loop in the script quickly saturates I/O for that partition and triggers rate throttling for *all* partitions

All I/O is throttled and <u>everything</u> slows down

Lessons learned:

1. Design your test cases to be compatible with the service’s architecture
2. If you can’t control the access pattern, then add protection against misuse (in this case, they pre-filtered requests and discarded requests where key=“key”|”ID”|”id”|”key_id”|…)

Case study*–initializing database triggers (nearly) endless partition re-balancing

MongoDB is a document database – each record is a JSON object

Database configuration defines maximum partition size

- When a partition hits that limit, it is split
- A new partition is created, half of the data is moved to the new partition
- This does not interrupt database access

Scenario – loading a database prior to testing

- Empty database has one partition
- Write test data records until the partition size limit is hit, triggers split and re-balance
- Writing continues during re-balance, quickly hits size limit for one of the new partitions, triggers another rebalance before the first one finished…

*J. Klein, I. Gorton, N. Ernst, et al., “Application-Specific Evaluation of NoSQL Databases,” in Proc. IEEE Big Data Congress, New York, NY, USA, 2015, pp. 526-534. doi: 10.1109/BigDataCongress.2015.83

Case study – initializing database triggers (nearly) endless partition re-balancing

Result:

- It took about 2 hours to write 10 million records
- It took the database about 24 hours to complete all the rounds of re-balancing

Work-arounds:

- Turn off rebalancing during the loading, then turn it on and let it run once
- Snapshot the storage image after the database was loaded (need to be careful with this – data contains write timestamps that may introduce new issues when reused later)

Cloud Native Services

In this module, we discussed

- Cloud platforms include many out-of-the-box services
- Architectures can trade off cloud native vs. portable implementations
- Impact on testing/assurance approach

Cloud Computing: An Architecture-centric View

Quality Attributes in the Cloud

Quality Attributes in the Cloud

In this module, we will discuss

- How cloud-based architectures promote and inhibit quality attributes
- What are the assurance considerations for several quality attributes

What is a quality attribute?

Quality attributes are properties of work products or goods by which stakeholders judge their quality.

Some examples of quality attributes by which stakeholders judge the quality of software systems are

- performance
- security
- modifiability
- reliability
- usability
- calibrateability
- availability
- throughput
- configurability
- subsetability
- reusability
- scalability

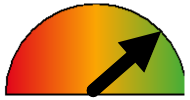
Quality attributes in cloud-based systems

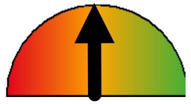
In cloud-based systems, some quality attributes are promoted, some are inhibited, and some are unaffected

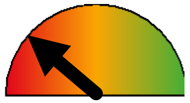
We'll assess the cloud's impact on several quality attributes

These are sweeping generalities

- With most architecture decisions, the real answer is "it depends"


Easier in the cloud


Unchanged


Harder in the cloud

Quality Attributes for Discussion

Security – we’ll cover this separately

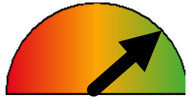
Scalability

Performance

Availability

Maintainability/Sustainability

Scalability



What do we mean?

- Add capacity or deliver very high capacity
  - Processing
  - Storage
  - Interactions

Storage scalability is easiest to achieve – essentially built-in

Processing and interaction scalability is relatively easy

- Cloud native autoscaling and load balancing services
- Does require some software architecture support to allow workload to be partitioned
  - Approaches include: Stateless, limited coordination or synchronization, dynamic cluster membership and leader election

Scalability – Assurance

Primary concern is processing/interactions

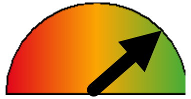
What are the scalability mechanisms used by the system?

- What are the triggers to scale up? Scale down?
- Test that scaling works correctly when it should, and doesn't happen when it shouldn't (see earlier case study)

What are the scalability limits imposed by cloud service provider?

- AWS has hard limits – see http://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html
- E.g., default is 20 VM reserved instances, 1-20 VM spot instances
- How close is the system to the limits? How does the system handle hitting a limit? Can separate parts of the system combine to hit a limit?

Performance
(Separate from Scalability)



What do we mean?

- Throughput – ability to process a quantity incoming events (requests, messages, targets, …)
- Latency – time needed to respond to an event

Easy to deliver and manage very large systems

- Infrastructure as code to create and deploy VM instances
- Very capable instance types available (see https://aws.amazon.com/ec2/instance-types/)
- Cloud native services for coordination and integration of instances
- Cloud native services for high performance architecture models (e.g., MapReduce)

Performance – Assurance

We’ll cover testing at scale in more detail later.

Challenges:

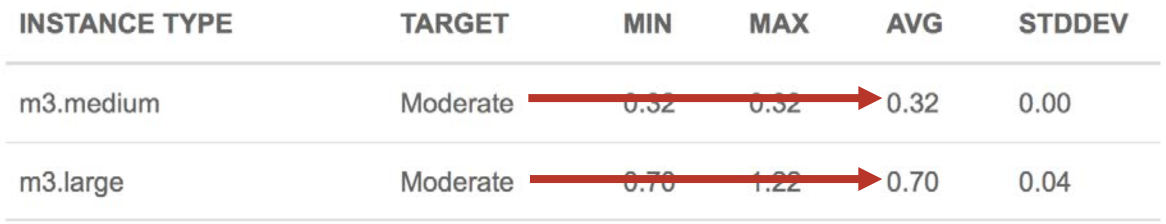
- Usual performance testing concerns – e.g., defining the workload, defining the background
- Executing the workload at scale
- Generating test data sets at scale (and getting that data into the cloud)
- Observing, collecting results, and verifying results at scale

(Continuous) verification of QoS of cloud provider services

- E.g., benchmark found twin-peak distribution on AWS VM performance – traced to physical hardware was some AMD, some Intel processors*

* D. Bermbach, “Quality of Cloud Services: Expect the Unexpected,” *IEEE Internet Computing,* vol. 21, no. 1, pp. 68-72, Jan 2017, doi: 10.1109/MIC.2017.1

Case Study* - Capacity Planning
What does “moderate” really mean?

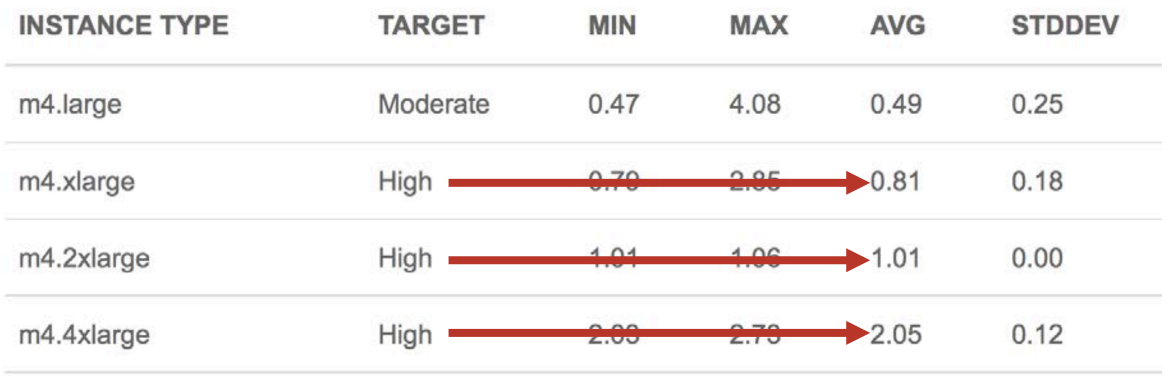


INSTANCE TYPE	TARGET	MIN	MAX	AVG	STDDEV
m3.medium	Moderate	~~0.32~~	~~0.32~~	0.32	0.00
m3.large	Moderate	~~0.70~~	~~1.22~~	0.70	0.04

AWS measured network I/O (Gbps)

* Andreas Wittig, https://cloudonaut.io/ec2-network-performance-demystified-m3-m4/

Case Study* - Capacity Planning
What does "high" really mean?

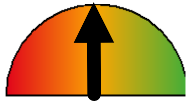


INSTANCE TYPE	TARGET	MIN	MAX	AVG	STDDEV
m4.large	Moderate	0.47	4.08	0.49	0.25
m4.xlarge	High	0.79	2.85	0.81	0.18
m4.2xlarge	High	1.01	1.06	1.01	0.00
m4.4xlarge	High	2.03	2.73	2.05	0.12

AWS measured network I/O (Gbps)

* Andreas Wittig, https://cloudonaut.io/ec2-network-performance-demystified-m3-m4/

Availability



What do we mean?

- System can detect, isolate, and mask or recover from faults, so that service delivery is uninterrupted

We are calling this “unchanged” for cloud-based systems, with a couple of caveats

- Not considering that Security, e.g., DOS attack, is linked to availability and performance (this concern *is better* in the cloud)
- Multi-region solutions are possible, but can be challenging (see Netflix Engineering Blog)

What is a cloud region?

Terminology and definition varies somewhat across cloud service providers, but most have this construct

E.g., “An **AWS Region** is a geographical location with a collection of availability zones mapped to physical data centers in that region. Every Region is physically isolated from and independent of every other Region in terms of location, power, water supply, etc…An **Availability Zone** is a logical data center in a Region available for use by any AWS customer. Each zone in a Region has redundant and separate power, networking and connectivity to reduce the likelihood of two zones failing simultaneously. *A common misconception is that a single zone equals a single data center. In fact, each zone is backed by one or more physical data centers, with the largest backed by five*.”*

* https://blog.rackspace.com/aws-101-regions-availability-zones

Regions and Availability Zones

Example, AWS `us-east-1` is a region

Note that AWS GovCloud is (currently) a single region

You must choose a region when launching a VM instance and most cloud native services

- Choosing an availability zone is usually optional

Elastic Load Balancer – Distribute requests across availability zones *within a region*

Route 53 DNS – use to balance across regions

Building cross-region systems is hard

- see e.g., R. Meshenberg, N. Gopalani, and L. Kosewski. *Active-Active for Multi-Regional Resiliency.* http://techblog.netflix.com/2013/12/active-active-for-multi-regional.html

Availability is about Faults – Faults in the Cloud

Root causes of unplanned outages*:

- infrastructure or software failures
- planning mistakes
- human error
- external attacks

Cloud infrastructure does fail, e.g.,

- After AWS physical reboot, Netflix had 22 out of 218 servers fail to restart (D. Harris. *Netflix lost 218 database servers during AWS reboot and stayed online* [Online]. https://gigaom.com/2014/10/03/netflix-lost-218-database-servers-during-aws-reboot-and-stayed-online/)
- Christmas Eve 2012 (https://medium.com/netflix-techblog/a-closer-look-at-the-christmas-eve-outage-d7b409a529ee)

* P. T. Endo, G. L. Santos, D. Rosendo, et al., "Minimizing and Managing Cloud Failures," *Computer,* vol. 50, no. 11, pp. 86-90, November 2017, doi: 10.1109/MC.2017.4041358.

Mitigation Approaches*

Monitoring

- In the cloud, verification is never finished**

Geo-distributed Storage and Redundancy

- Can achieve some geo-distribution within a region
- Requires careful design and configuration (opening the door to human error)

Disaster Recovery

- Cross-region failover – note that this is not (currently) an option for government cloud deployments

P. T. Endo, G. L. Santos, D. Rosendo, et al., "Minimizing and Managing Cloud Failures," *Computer,* vol. 50, no. 11, pp. 86-90, November 2017, doi: 10.1109/MC.2017.4041358.
J. Klein and I. Gorton, "Runtime Performance Challenges in Big Data Systems," in *Proc. Workshop on Challenges in Performance Methods for Software Development (WOSP-C'15)*, Austin, TX, 2015. doi: 10.1145/2693561.2693563

Availability – Assurance

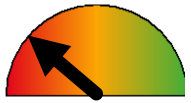
Certain types of faults cannot be directly induced

- E.g., you can’t pull out a network cable – need to use intrusive tools like *netem* (https://wiki.linuxfoundation.org/networking/netem) to simulate network failures
- Generally, cloud testing relies more on simulated faults – need to assess the quality of the simulation → quality of the evidence

Need for practices and procedures that bridge between cloud provider’s QoS guarantees and evidence that you collect directly

* Intrusive Tools = Install on target system or change configuration

Maintainability/Sustainability



What does it mean?

- Required changes can be made to the software to keep the system secure and operating

We’re calling this worse in the cloud for stable systems

- From a purely technical perspective, some things are better, some worse
- No real experience with long-lived static systems deployed to the cloud
- **Assurance is never finished – this can be a big change in mindset, policy, funding, …**

If you are continually evolving your system and you’ve embraced DevOps, then this quality is probably better in the cloud

- Environment parity between production and development
- You are continuously testing/integrating/delivering

Maintainability/Sustainability

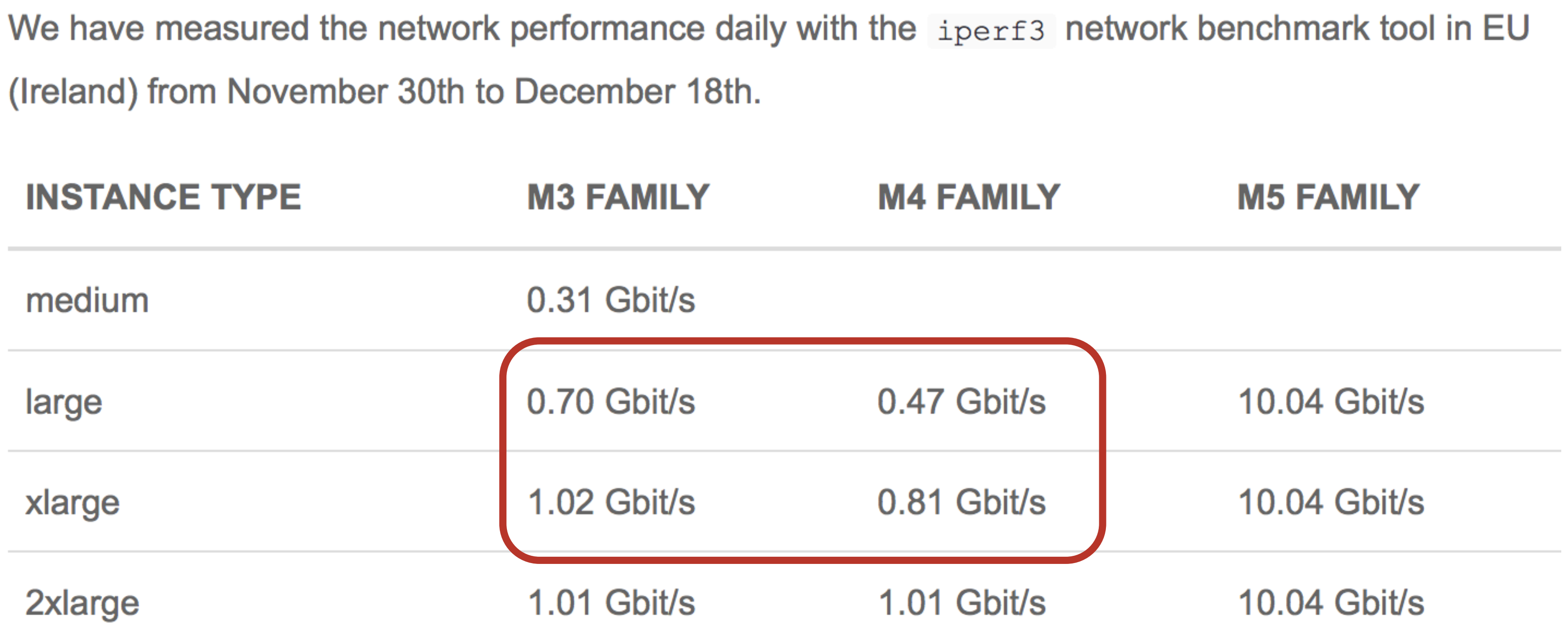
Easier:

- Infrastructure as code practices improve the repeatability of deployment
- Virtualization allows development environment to be identical to production environment
- Cloud should impose higher degree of uniformity of deployment configurations (IaaS)
- No infrastructure patching or management concerns at all (PaaS and SaaS)

Harder:

- Cloud provider can change the infrastructure in ways that impact your system but still satisfy QoS guarantees
- Cloud provider offerings evolve over time – issue for cloud native services, PaaS, and SaaS
- Tempo difference between your system and cloud providers – there is no experience with long-lived static systems deployed to the cloud

Case Study* – Newer may not be better



We have measured the network performance daily with the `iperf3` network benchmark tool in EU (Ireland) from November 30th to December 18th.

INSTANCE TYPE	M3 FAMILY	M4 FAMILY	M5 FAMILY
medium	0.31 Gbit/s		
large	0.70 Gbit/s	0.47 Gbit/s	10.04 Gbit/s
xlarge	1.02 Gbit/s	0.81 Gbit/s	10.04 Gbit/s
2xlarge	1.01 Gbit/s	1.01 Gbit/s	10.04 Gbit/s

* Andreas Wittig, https://cloudonaut.io/evolution-of-the-ec2-network-performance-m3-m4-m5/

Maintainability/Sustainability – Assurance

In the cloud, Test and Evaluation is never finished

- Continuous assessment that QoS guarantees are being met
  - Monitoring and trending
  - Within a system and across systems
- Continuous assessment that the delivered infrastructure remains compatible with your systems
  - E.g., Netflix's Chaos Engineering (more about this later)
- Working with cloud service providers to understand their roadmaps and assess impact on systems in production AND in development

Take-aways

The cloud makes some things better, some things worse.

Some of these impacts are intrinsic to any cloud computing (i.e. performance)

Other impacts are more specific to your system context, especially US government systems (i.e. availability, maintainability)

Quality Attributes in the Cloud

In this module, we discussed

- How cloud-based architectures promote and inhibit quality attributes
- What are the test and evaluation considerations for several quality attributes

Cloud Computing: An Architecture-centric View

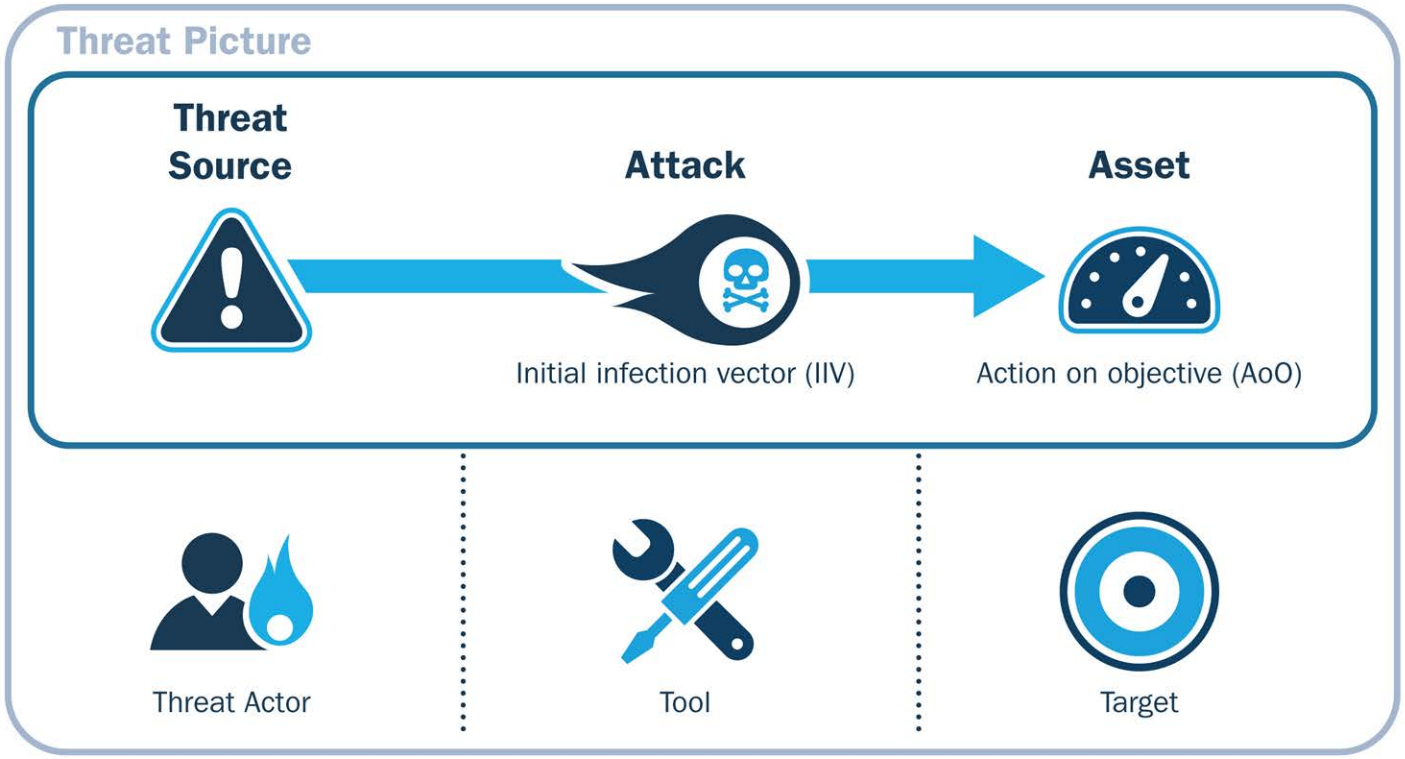
Introduction to Cloud Security

Cloud Security

In this module, we will discuss

- Threats and infection points
- Examples of different views using AWS
- Hybrid cloud example and its associated different views
- Cloud unique and cloud/on-premise threats/vulnerabilities
- Four key security practices

Setting the Context



Threat Terminology

Threat source – a method by which a vulnerability is triggered or exploited

Attack (initial infection vector) – method used to gain access to system

Asset – the object of the attack

Threat actor – an entity that is partially or wholly responsible for an incident that impacts or has the potential to impact an organization's security.

Tool – e.g., phishing email, remote access Trojan (RAT), SQL injection

Target – e.g., personally identifiable information (PII) data, trade secrets, network configuration information

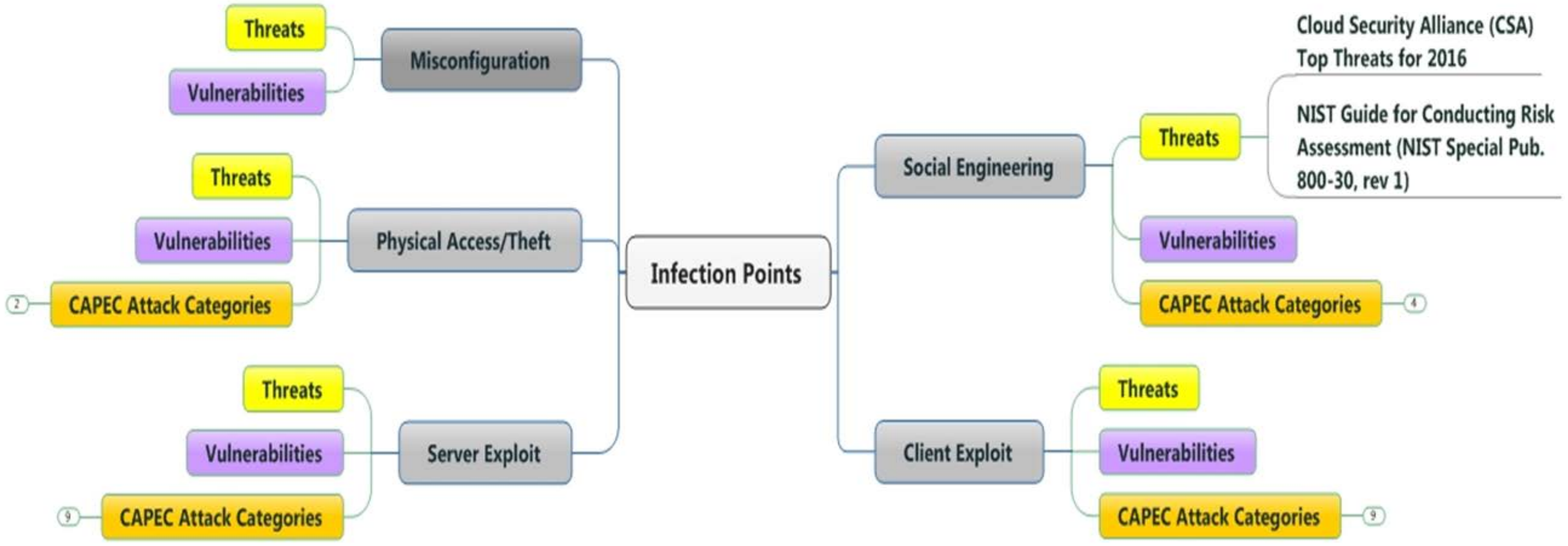
Data Center Threats

The SEI developed a holistic approach when considering attacks on computer systems which is based on the following two questions.

- “How did they get in?”
- “What did they do after they were in?”

To answer the first question, five ways to get into a computer system (infection points) were identified.

Five Infection Points



Cloud Incidents Examples Associated with Infection Points

Social engineering – “How Apple and Amazon Security Flaws Led to my Epic Hacking”, “In the space of one hour, my entire digital life was destroyed.”, (http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/)

Client exploit – AWS OpenSSL Security Advisory - May 2016; “AWS will appropriately update OpenSSL to improve security for AWS customers who are utilizing outdated web browsers that cannot negotiate the AWS preferred and recommended AES-GCM TLS/SSL cipher suites when interacting with the AWS Management Console.”, (https://aws.amazon.com/security/security-bulletins/openssl-security-advisory-may-2016/)

Misconfiguration – Amazon ELB Service Event in the US-East Region on December 24, 2012, portion of ELB state data was logically deleted which is used and maintained by the ELB control plane to manage the configuration of the ELB load balancers in the region. (https://aws.amazon.com/message/680587/);

Cloud Incidents Examples Associated with Infection Points

Server exploit – AWS CVE-2015-7547 Advisory - ”We have reviewed the issues described in CVE-2015-7547 and have determined that AWS Services are largely not affected. The only exception is customers using Amazon EC2 who’ve modified their configurations to use non-AWS DNS infrastructure should update their Linux environments immediately following directions provided by their Linux distribution. EC2 customers using the AWS DNS infrastructure are unaffected and don’t need to take any action. A fix for CVE-2015-7547 has been pushed to the Amazon Linux AMI repositories, with a severity rating of Critical. Instances launched with the default Amazon Linux configuration on or after 2016/02/16 will automatically include the required fix for this CVE.” (https://aws.amazon.com/security/security-bulletins/cve-2015-7547-advisory/)

Physical access/theft – AWS service event in the Sydney region due to loss of power on June 6, 2016. Unusually long voltage sag caused the loss of both primary and secondary power. (https://aws.amazon.com/message/4372T8/).

So What Else Do We Need to Understand?

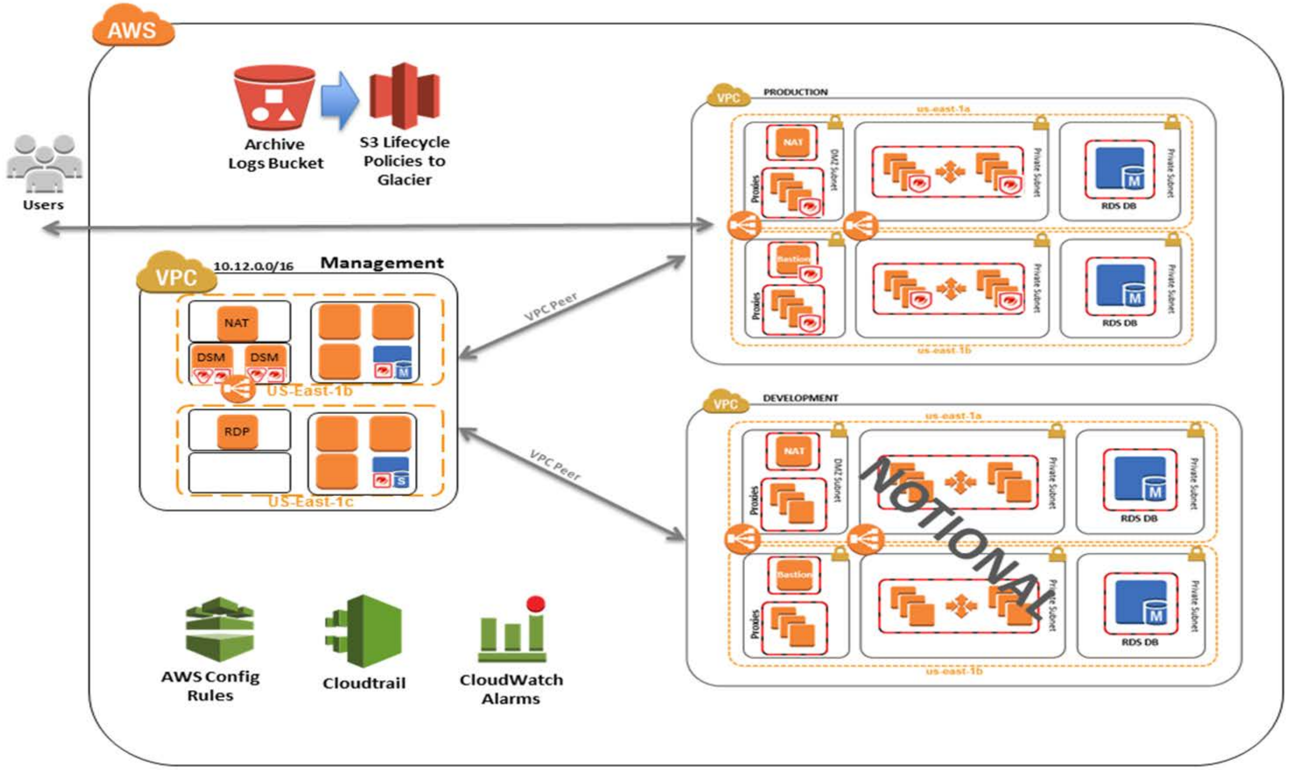
We now have a good grasp of the threat picture which can be applied to data centers, a cell phone, refrigerator, and clouds.

But how do I apply it to do analysis, testing, risk identification, and risk mitigation?

You will need architecture documentation to support these efforts.

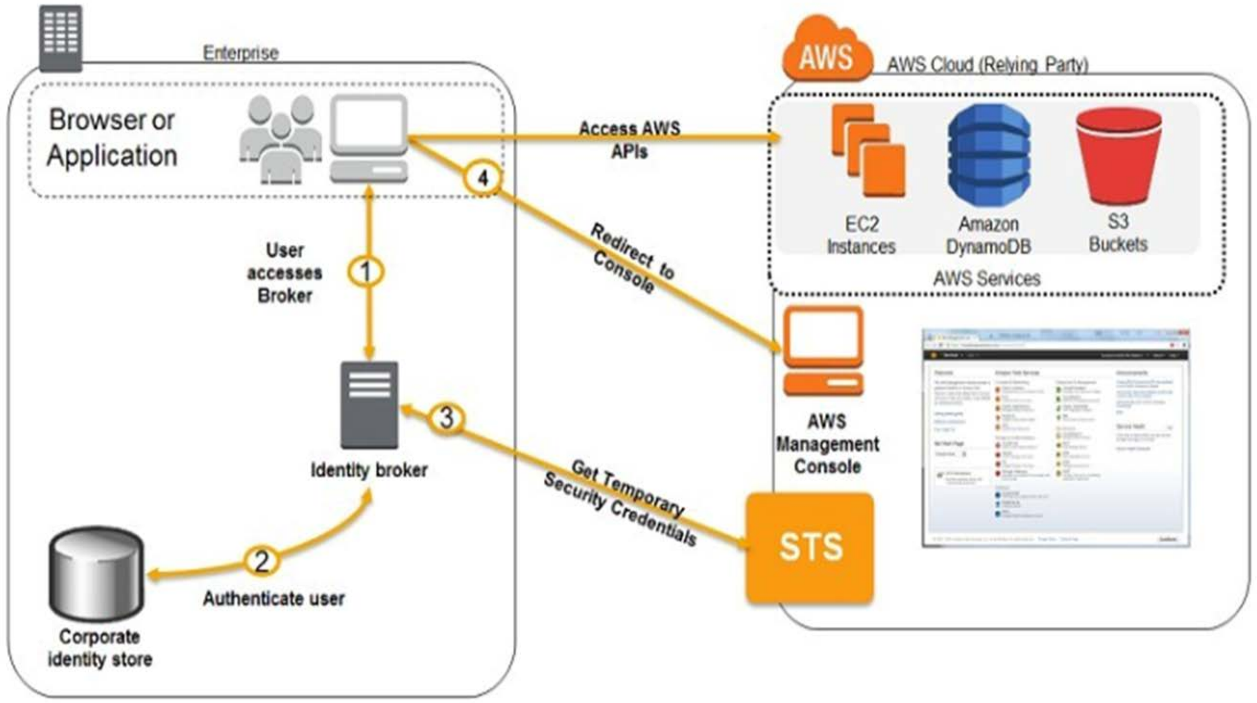
Architecture documentation will need to be developed that provides multiple views of the system to satisfy different stakeholders.

Example – Cloud Deployment View of a Web Application Which Supports NIST Compliance

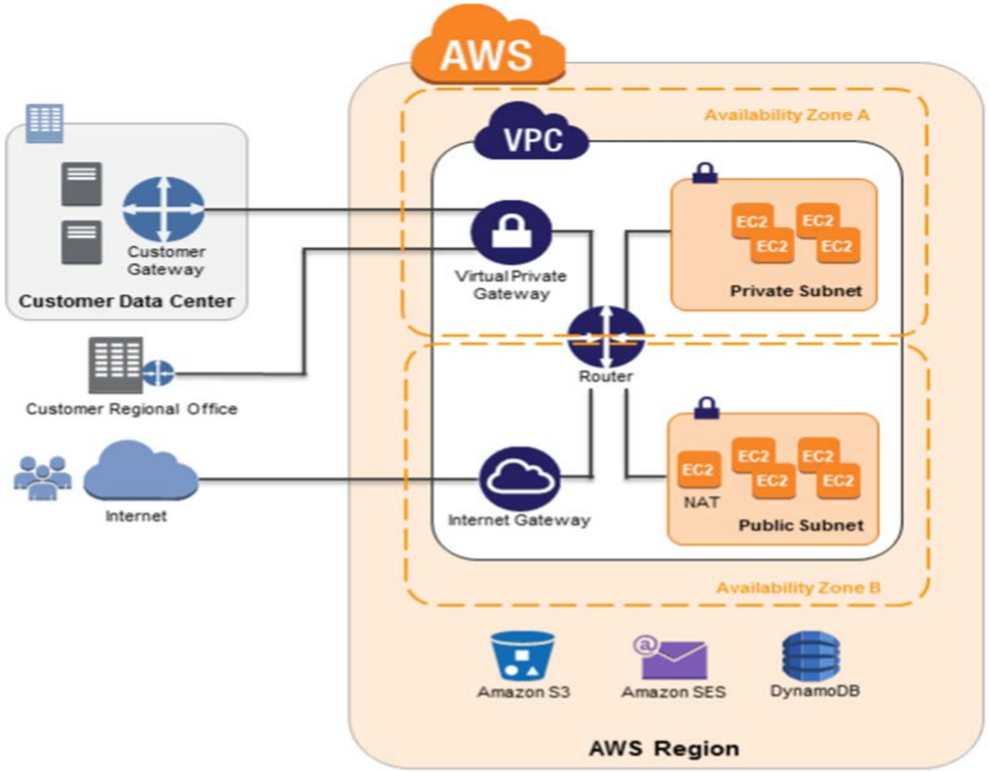


https://aws.amazon.com/quickstart/architecture/accelerator-nist/

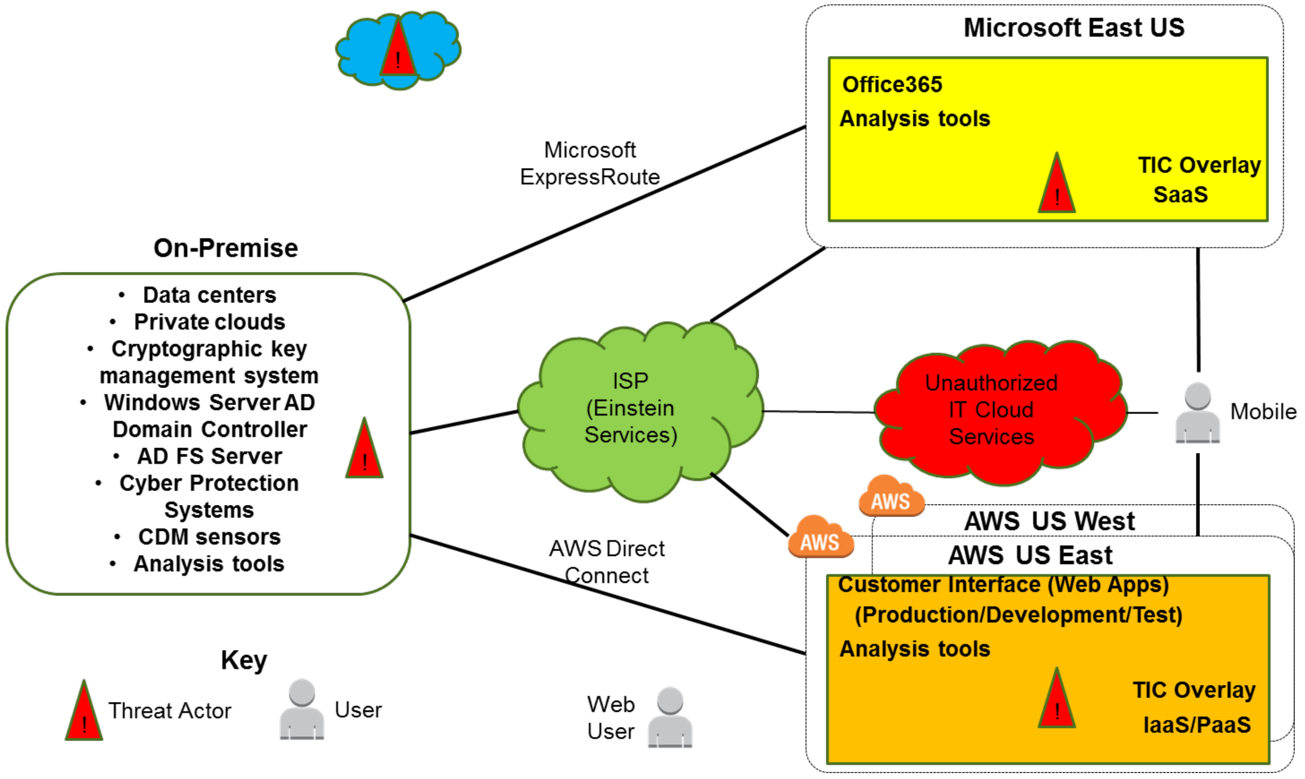
Example – Identity and Access Management (IAM) Service View for Modeling Threat Events



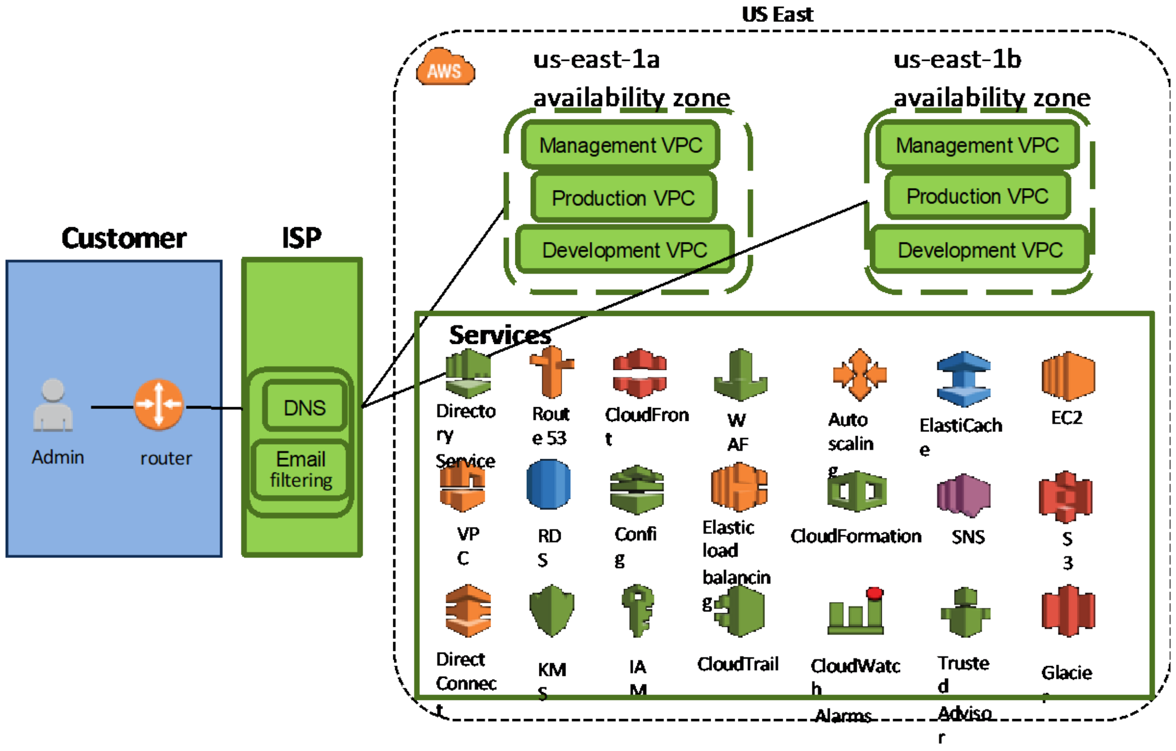
Example – Virtual Private Cloud (VPC) View for Modeling Threat Events



Example – Hybrid Cloud



Example – Customer's Administrator's View AWS IaaS



Cloud Vulnerabilities/Threats

Cloud Unique

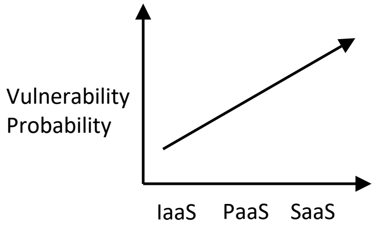
1. Reduced Visibility & Control
2. Ability to Self Provision Resources & Services
3. Management API Compromise
4. Multi-Tenancy Security
5. Secure Data Deletion

Cloud & On-Premise

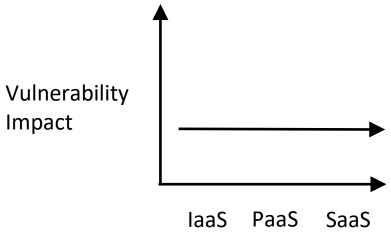
6. Stealing Credentials
7. Vendor Lock-in
8. Increased Complexity Strains IT Staff
9. Insider Threat
10. Data Recovery
11. Supply Chain

#1 Reduced Visibility & Control

When transitioning assets/operations to the cloud, agencies will lose some visibility and control over the assets/operations because the CSP is now handling aspects via its infrastructure and policies. Paradigm shift is needed by agencies to focus on attaining monitoring and logging information about applications, services, data and users, rather than the network focus of on-premise IT.



- As the CSP assumes more responsibilities, an agency will need to find different ways to attain the information to successfully monitor IT operations and satisfy security and compliance requirements.
- Agency must work jointly (can't direct) with CSP via their service level agreement (SLA).

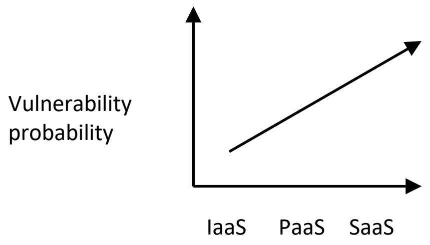


#2 Ability to Self Provision Resources & Services

Self provisioning capabilities of cloud enable agency personnel to:

- Provision extra services not originally planned for with the agency's CSP and that don't have IT consent.
- Individually use SaaS products (Dropbox, iCloud, OneDrive, …) independent of IT.

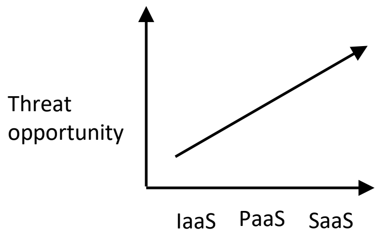
These services are unknown risks to an agency. (cloud scope creep)



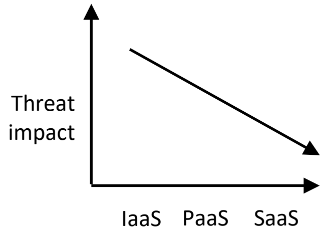
- Due to the lower costs and ease of implementing PaaS and SaaS products, the probability increases.

#3 Management API Compromise

CSPs expose a set of application programming interfaces (APIs) that customers use to manage and interact with cloud services. Agencies use these APIs via the internet to provision, manage, orchestrate and monitor their assets and users. The vulnerability is that these APIs have the same software vulnerabilities that an API for an operating system, library, etc. could have.

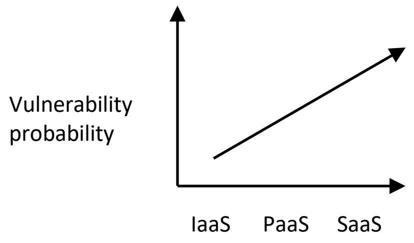


- Threat actor is looking for vulnerabilities in management APIs.
- If vulnerability can be turned into an attack, then this could be used against other customers of the CSP.
- Vulnerability focus more on configuration/provisioning.



#4 Multi-Tenancy Security

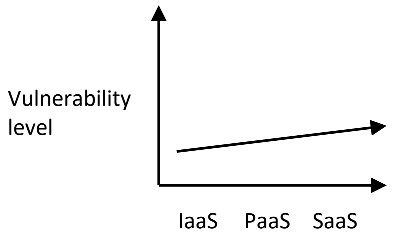
System and software vulnerabilities within a CSP's infrastructure, platforms or applications which supports multi-tenancy can lead to isolation failure where an attacker exploits the vulnerability to access to another user's or agency's assets/data.



- Different than vulnerability #3 because this focuses on how the CSP implements the agency's desired capabilities.
- Examples:
  - IaaS – VMs, OS's
  - PaaS – app servers, Java VM
  - SaaS – databases, business logic, workflow, user interface

#5 Secure Data Deletion

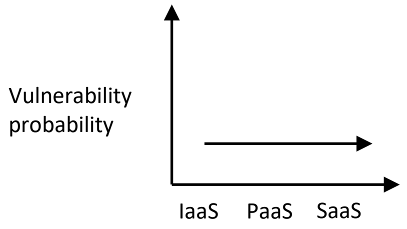
CSP’s ability to securely delete and verify when an agency deletes data. This is a concern due to the data being spread over a number of different storage devices within the CSP’s infrastructure in a multi-tenancy environment.

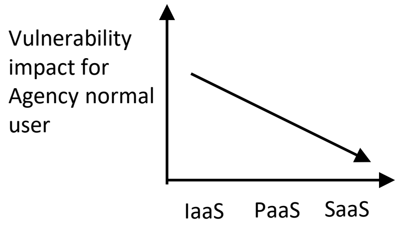


- Vulnerability increases as an agency uses more CSP services.

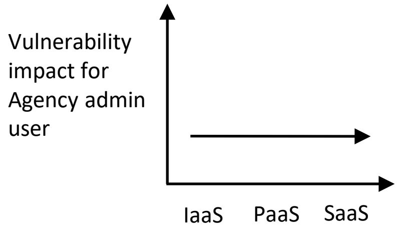
#6 Stealing Credentials

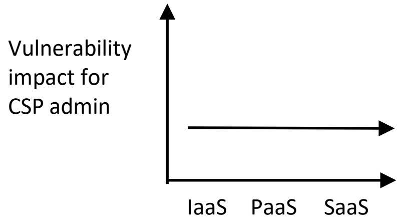
If an attacker gains access to your cloud credentials, the additional vulnerability in the cloud is that the attacker would have access to the CSP’s services to provision additional resources, as well as target agency’s assets. The attackers could leverage cloud computing resources to target users, organizations or other cloud providers.





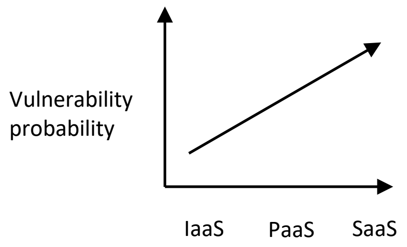
- Admin roles vary between CSP and agency.
- CSP admin would address more than one customer and probably handle all the CSP’s services offered.
- Vetting processes for becoming a CSP admin may be different than the process used for an agency’s admin. Need to be aware of the differences and assess their impact.





#7 Vendor lock-in

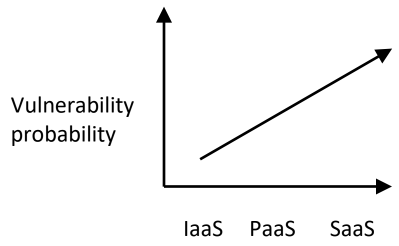
This vulnerability could occur when an agency considers moving its assets/operations from one CSP to another CSP. The agency finds out than the cost/effort/schedule time necessary for the transition is much higher that initially considered due to non-standard data formats, non-standard APIs, high cost charged to remove presence with original CSP, inability to transfer large amounts of data out of a CSP in a timely manner, reliance on one CSP’s proprietary tools, and CSP’s unique APIs.



- Vulnerability increases as the CSP takes more responsibility. As more features/services/APIs are used, there is increased exposure to CSP’s unique implementations.
- If selected CSP goes out of business, it becomes a major problem.

#8 Increased Complexity Strains IT Staff

This vulnerability is concerned with an existing agency's IT staff having the capacity and skill level to manage, integrate and maintain the transition of assets and data to the cloud in addition to their current responsibilities for on-premise IT. The services/techniques/tools available to log and monitor them typically vary across CSPs, further increasing complexity. Also, there may be emergent vulnerabilities/risks in hybrid cloud implementations due to technology, policies, implementation methods add complexity.



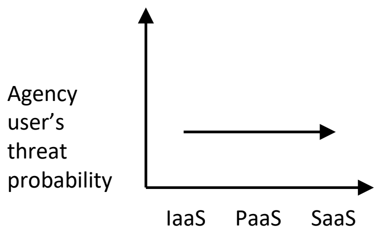
- Increased potential for coverage gaps between the layers.
- Probability increases if agency pursuing hybrid cloud implementation.

#9 Insider Threat

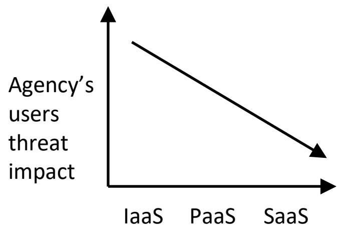
A malicious insider is defined as a current or former employee, contractor, or business partner who meets the following criteria:

- has or had authorized access to an organization's network, system, or data
- has intentionally exceeded or intentionally used that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization's information or information systems

This applies to staff and administrators for both agencies and CSPs.

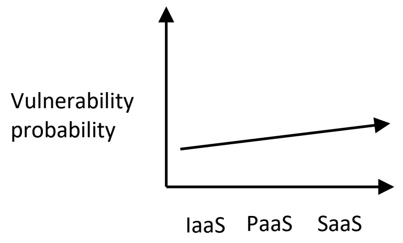


- Likely worse for IaaS because of the ability to provision resources or possibly perform nefarious activities that will require forensics that may not be available with cloud resources vis-a-vis on-premise resources.
- CSPs' users threat impact will depend upon their organization's employee vetting process (background checks) and controls implementation.



#10 Data Recovery

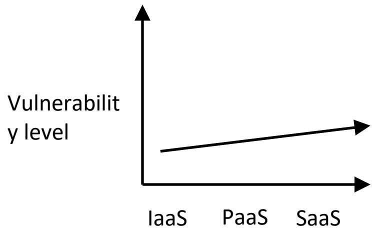
Data stored in the cloud can be lost for reasons other than malicious attacks. An accidental deletion by the cloud service provider or worse, a physical catastrophe such as a fire or earthquake, can lead to the permanent loss of customer data. The burden of avoiding data loss does not fall solely on the provider's shoulder. If a customer encrypts his or her data before uploading it to the cloud but loses the encryption key, the data will be lost as well.



- Vulnerability increases as an agency uses more CSP services.
- Data recovery for a CSP is may be better than that of an agency due to SLA designating availability/uptime percentages.

#11 Supply Chain

This vulnerability is concerned with the supply chain that a CSP uses to support its services. If the CSP outsources parts of its supply chain, then these third parties may not satisfy/support the requirements that the CSP is contracted to support with an agency. An agency would need to check to see if the CSP flows its own requirements down to their third party and see how it enforces compliance. If the requirements are not being flowed down, then there is an increased threat to the agency.



- Vulnerability increases as an agency uses more CSP services.
- This is very dependent on individual CSPs and their supply chain policies.

Four Important Cloud Security Practices

1. Perform due diligence

2. Manage access

3. Protect data

4. Monitor and defend

Cloud Security Practices

1) Perform due diligence

Encourages cloud consumers to fully understand their current network and applications to better appreciate the functionality, resilience, and security of cloud services before migrating to cloud-deployed application and system.

2) Manage access

Describes the different categories of users in a cloud-based IT environment and explains the responsibilities of both CSP and cloud consumers in managing these user’s access to resources.

Cloud Security Practices

3) Protect data

Describes the two consumer challenges of preventing the accidental disclosure of data that was supposedly deleted and ensuring continued access to critical data in the event of errors, failures, and compromise.

4) Monitor and defend

Describes the shared responsibility of the CSP and cloud consumer in monitoring the cloud-based system and applications to detect unauthorized access to data or unauthorized use of resources.

Conclusions

While potential cloud consumers often worry about the security risk of trusting a CSP to perform some security functions, experience has shown that security incidents are more often the result of consumer failing to use the security tools provided.

The need to cloud consumers to develop a deep understanding of the services they are buying and to use the security tools provided by the CSP.

Like any new technology or approach, using it effectively and securely requires knowledge and practice. Use of well-established, mature CSPs will help reduce risk associated with transitioning application and data to the cloud.

Cloud Vulnerabilities

In this module, we discussed

- Threats and infection points
- Examples of different views using AWS
- Hybrid cloud example and its associated different views
- Cloud unique and cloud/on-premise threats/vulnerabilities
- Four key security practices

Cloud Computing: An Architecture-centric View

Distributed Systems Concepts

Distributed Systems Concepts

In this module, we will discuss

- Clouds are distributed software systems
- The “laws of physics” that limit the visibility and capabilities of distributed software systems
- Impact on testing approach

Deutsch’s Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn’t change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

See https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Deutsch's Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

In this section

In other sections

See https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Communication and Coordination

The “FLP” result

- Michael J. <u>F</u>ischer, Nancy A. <u>L</u>ynch, and Michael S. <u>P</u>aterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374–382, 1985. doi:10.1145/3149.214121.

Conclusions (in an asynchronous environment – no timeouts)

- You can’t distinguish a crashed process from a broken network link
- You can’t distinguish a broken link from a really slow link

Communication and Coordination – Practical Implications

Guaranteed message delivery is impossible

- Does the system impose timeouts? In one layer? Multiple layers? How long is the timeout?
- Does the system design assume that messages are never lost?
- Does the system design assume that messages will arrive in-order?

Exactly-once delivery is tricky but possible

- What happens if a message is repeated?

Atomic broadcast (think “guaranteed one-to-many”) is impossible without application-level cooperation

- If a system design claims this feature, it warrants some testing

Replicated State

If we have more than one copy of a data element in our system, we have to be concerned about whether they are consistent.

- Simple state – Who is the current master? What mode are we in?
- Complex state – a distributed database or file system
- Distributed caching to improve performance

The CAP Theorem

- E. A. Brewer, “Towards robust distributed systems,” in *Proc. 19th Ann. ACM Symp. on Principles of Distributed Computing (PODC '00)*, 2000, pp. 7. doi: 10.1145/343477.343502

Tradeoff among <u>C</u>onsistency, <u>A</u>vailability, <u>P</u>artition-tolerance

CAP

Consistent - All requests will return the same value (note that this is different from the “C” in SQL ACID transactions)

Available – All requests return some value

Partition-tolerant – System continues to operate when there is a network partition between stateful nodes

Possibilities:

- CP – Sacrifice availability – e.g., most SQL implementations
- AP – Sacrifice consistency – e.g., many NoSQL data stores
- CA – Sacrifice partition-tolerance - e.g., single node or single point of failure (SPOF) routing

CAP Implications

Recognize when this tradeoff is relevant – is there replicated state in a distributed software system?

What does the design accommodate? Is that reasonable?

Testing to validate the edge cases is REALLY hard

- Kyle Kingsbury, aka Aphyr, has made a career of this
- http://jepsen.io (We’ll talk about this in more detail later)
- Worth studying his approaches and results

Time in Distributed Systems

Operating system-level clock synchronization is not achievable for cloud applications

Cloud Service Providers CAN provide atomic/GPS clock synchronization for some nodes in their data centers

- E.g., Google’s Spanner distributed database relies on GPS clocks
- Applications can leverage cloud services that depend on tight time synchronization

Many distributed systems use software “clocks” (i.e. counters) to order events – this is usually good enough

- Lamport clocks or timestamps
- Vector clocks

Time Synchronization Implications

Be wary of systems that get time directly from the operating system to order or synchronize events

- E.g., comparing file timestamps across nodes

Log correlation across nodes is difficult without message IDs or similar tactics

A related issue: You can’t set the clock of a cloud server

- Testing cases like leap second handling gets tricky
- Designs that introduce a time abstraction layer to separate application time from OS time are more testable

Distributed Systems Concepts

In this module, we discussed

- Clouds are distributed software systems
- The “laws of physics” that limit the visibility and capabilities of distributed software systems
- Impact on testing approach

Cloud Computing: An Architecture-centric View

Testing at Scale in the Cloud

Testing at Scale in the Cloud

In this module, we will discuss

- Challenges of testing cloud-based software
- Examples of commercial leading practices for cloud testing

You have to test cloud software in the cloud

“… asking to boot a cloud on a dev machine is equivalent to becoming multi-substrate, supporting more than one cloud provider, but one of them is the worst you’ve ever seen”

- Fred Hébert*

* Quoted in https://medium.com/@copyconstruct/testing-microservices-the-sane-way-9bb31d158c16

Definition of Testing

In this section, we take a broad view – *testing* is the collection of evidence about the quality of a system

Encompasses both cyber assurance and operational effectiveness activities

Test activities usually involve making compromises due to constraints on *controllability* and *observability*.

How does the cloud affect testing practices?

Controllability:

+ Easy to exactly reproduce environment (infrastructure as code)

+ Easy and affordable to scale up workload (requests and data sets)

- Time-consuming to transfer big test sets into the cloud – try to generate in the cloud

- Hard/impossible to break some things for testing (e.g., network, power, …) – need to simulate these

+/- “Automate all the things” – can add complexity

Observability:

+ Easy and affordable to save everything

- Expensive and time-consuming to get big result sets out of the cloud – need to summarize/analyze in the cloud

+ There are cloud-based tools to help summarize and analyze

- Cloud native services are opaque black-boxes – may need to test for longer periods or multiple times to adequately characterize

What have we said already about testing

Infrastructure as code, versioned environments

- For deploying the target system
- For deploying the test and data analysis environment

Cloud-based software is a distributed system

- All the principles of testing distributed systems still apply, even though the control mechanisms may change

Consider unintended side effects during testing (e.g., triggering autoscaling)

- Impacts fidelity
- May impact testing cost

Fault simulation instead of fault creation

- Usually intrusive – impacts fidelity

Common Infrastructure →
Reuse Test and Assurance Evidence

Within a particular cloud provider environment (e.g., Amazon EC2), you can reuse some test results and evidence related to cloud native services

- E.g., everyone is using the *same* S3 Simple Storage Service, so results about performance, availability, etc. should be reusable across systems

Validate service configuration instead of runtime behavior

Test Data Sets

It is time-consuming to get big test sets into the cloud, so if you have to upload a data set, plan to do it only once

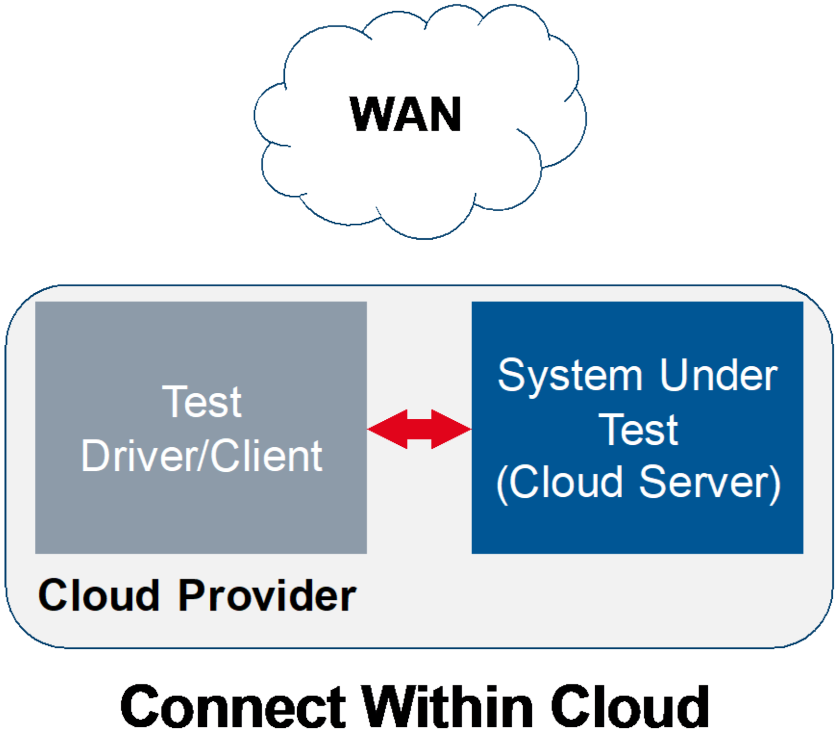
Avoid uploading:

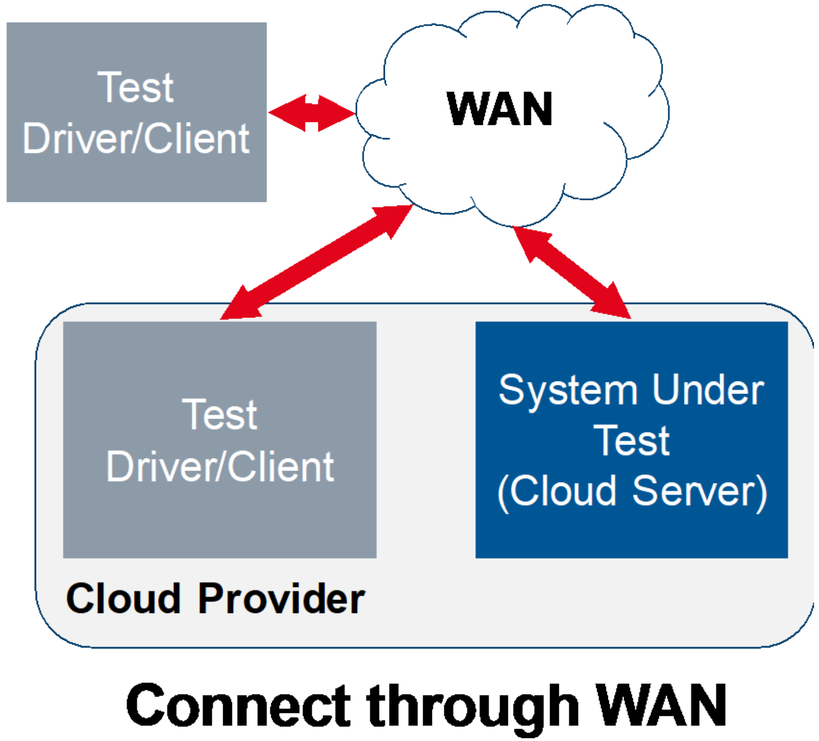
- Generate and save the data set in the cloud
- Generate the data set on-the-fly (compute resources are cheap)

Strategies to save data sets

- In block storage (e.g., AWS S3), and read into VM instance (slower, cheaper, scalable)
- As snapshot'd read-only volume attached to VM instance file system (faster, more expensive, attach to single VM)

Test clients/workload driver connectivity





Which client configuration should I use?

Connect Within Cloud

Use for when real client will be in the same cloud as the system-under-test (duh!)

Use to stress performance

- Scale up clients
- Optimal network capacity

Connect through WAN

Use when the real client will access the system-under-test over the WAN (duh!)

Use when it is not feasible to host the test client in the cloud (e.g., hardware-in-the-loop)

Can require careful configuration if the client is in the same cloud

- CSPs try to optimize to keep traffic off the WAN
- Consider putting test clients in another cloud (e.g., test AWS system using Azure clients)

Example of Commercial Testing Practice
“Chaos Engineering”

Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system’s capability to withstand turbulent conditions in production.

- http://principlesofchaos.org

Closed loop – develop, test, refactor…

Originated at Netflix – Chaos Monkey and the Simian Army

- Test in production
- Randomly select and crash servers
- Use robust observability framework to assess impact

Chaos Engineering Principles*

Start by defining ‘steady state’ as some measurable output of a system that indicates normal behavior.

- Note that this depends on having a well-instrumented system-under-test

Hypothesize that this steady state will continue in both the control group and the experimental group.

Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.

- In the cloud, we will have to simulate much of this

Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

* http://principlesofchaos.org

Example of Commercial Testing Practice
“Jepsen”

Jepsen is an effort to improve the safety of distributed databases, queues, consensus systems, etc.

- https://jepsen.io

Focused on properties of distributed storage systems

- Durability, atomic writes, replica consistency

Applies knowledge of where the edge cases are and how you get there

- E.g, faulty networks, unsynchronized clocks, and partial failure

Code at https://github.com/jepsen-io/jepsen

- Control node
- Clients that generate workload (write and read)
- “Nemesis” - inject (simulate) faults under control of Control node
- Checker

Comments on Jepsen

Included as an example

- This is how experts are testing software in the cloud
- Use the cloud to test the cloud - cost-effective elastic capacity to generate scalable workloads
- Open source
- Applies domain knowledge of both
  - cloud (what are the possible faults?) and
  - system-under-test (what are the edge cases?)

We don’t expect that you would ever use the tool directly

And one more time…

We’re never finished saying that **testing cloud-based software is never finished**

- Cloud services evolve independently of your systems
- Cloud services can evolve silently
- Cloud infrastructure evolves – networks, ingress/egress, performance

Assurance is not a one-time event

Testing at Scale in the Cloud

In this module, we discussed

- Challenges of testing cloud-based software
- Examples of commercial leading practices for cloud testing

Cloud Computing: An Architecture-centric View

Wrap-Up

Perspectives on Cloud-based Systems

There are useful perspectives that can provide insights when considering cloud-based systems

- Cloud as COTS (Commercial off the Shelf Software)
- Cloud as Common Platform
- Cloud as System of Systems

Adapting existing practices, processes, and knowledge can help us in the cloud

Cloud as COTS

Adopting cloud computing introduces many of the concerns that we are familiar with from COTS

- Supply chain integrity
- Vendor lock-in
- Lack of transparency
- Mismatch between vendor’s evolution direction and system evolution direction
- Mismatch between vendor’s evolution cadence and system evolution cadence
- Need for vendor-specific skills for development and test

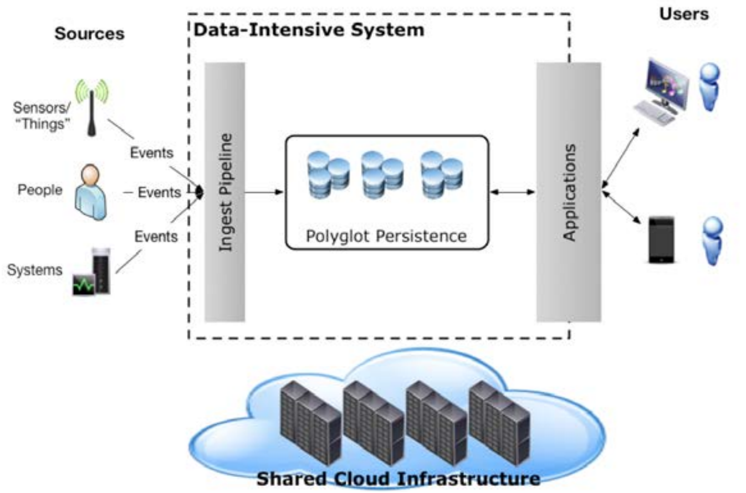
Cloud as Common Platform

DoD seems to view this as a benefit of cloud adoption

Common platform concerns:

- Cost/benefit of system-optimized platform vs. common platform
- Establishing and maintaining common baseline across programs
- Sharing knowledge and experience about the platform across programs
- Migration from system-unique to common platform, short-term or long-term use of hybrid deployment

Cloud as System of Systems



Sources evolve independently

User workloads change over time

- New uses
- New mix of operations

Cloud quality of service varies

Partly inherent in any cloud-based system, but also due to the type of data-intensive systems that we deploy to the cloud (e.g., situational awareness, decision support, business analytics)

Cloud as System of Systems

Concerns:

- Definition of system boundary for design and for T&E
- Ongoing monitoring of deployed system – is it operating within its design envelope?
    - Initial T&E of that monitoring
    - Who is responsible for watching the deployed system?
- Coordination of evolution (similar to common platform concern)

Final Take-aways

We covered:

- How cloud computing is different from traditional data center system deployment
  - Virtualization, cloud-native services
- Controllability and observability in the cloud impacts test and evaluation
- Cloud computing improves some system qualities while inhibiting others – this affects test and evaluation
- Cloud-based systems introduce some new cybersecurity risks

Questions and Discussion