



CLOUD COMPUTING

Introduction to Cloud Computing
Terminologies
Parallel, Distributed

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Cloud Computing

- Cloud computing is basically delivering computing at the Internet scale.(T2)
- Compute, storage, networking infrastructure as well as development and deployment platforms are made available on-demand within minutes(T2)
- *Cloud computing is a model for enabling ubiquitous (**where-ever**), convenient, **on-demand** (**when-ever**) network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that may be shared (across applications) and can be rapidly provisioned and released (**quickly**) with minimal management effort or service provider interaction*
- **Data centers** house the physical compute, storage and networking infrastructure (along with components which enable functioning or running of these active IT components). These Data center's are also called *Cloud Data center's* when they host these physical resources and enable Cloud computing over the Internet.
- Large clouds, predominant today, often have functions distributed over multiple locations from central servers
- If the connection to the user is relatively close, it may be designated an edge server.
(https://en.wikipedia.org/wiki/Cloud_computing)

Introduction to Cloud Computing

Features/Characteristics of Cloud Computing(T2)

On demand self-service: The compute, storage or platform resources needed by the user of a cloud platform are self-provisioned or auto provisioned with minimal configuration.

Broad network access: Ubiquitous access to cloud applications from desktops, laptops to mobile devices is critical to the success of a Cloud platform

Resource pooling: Cloud services can support millions of concurrent users; cloud services need to share resources between users and clients in order to reduce costs.

Scalability: Cloud Services can accommodate larger or smaller loads while supporting some of the expectations of QoS like response time

Rapid elasticity: A cloud platform should be able to rapidly increase or decrease computing resources as needed.

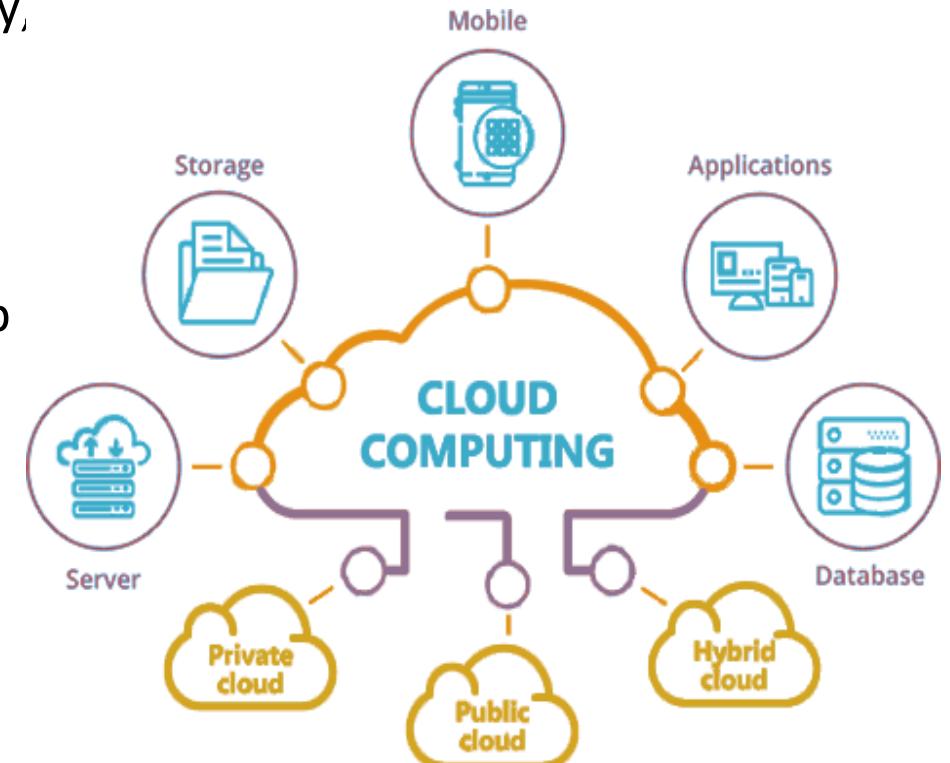
Measured service: One of the compelling business use cases for cloud computing is the ability to “pay as you go,” where the consumer pays only for the resources that are actually used by her applications

Introduction to Cloud Computing

Cloud computing usage (<https://aws.amazon.com/what-is-cloud-computing/>)

Organizations of every type, size, and industry are using the cloud for a wide variety of use cases, such as data backup, disaster recovery, email, virtual desktops, software development and testing, big data analytics, and customer-facing web applications.

- For example, healthcare companies are using the cloud to develop more personalized treatments for patients.
- Financial services companies are using the cloud to power real-time fraud detection and prevention.
- And video game makers are using the cloud to deliver online games to millions of players around the world.



Agility

Quickly spin up resources as you need them—from infrastructure services, such as compute, storage, and databases, to Internet of Things, machine learning, data lakes and analytics, and much more.

Elasticity

Scale these resources up or down to instantly grow and shrink capacity as your business needs change.

Cost savings

The cloud allows you to trade capital expenses (such as data centers and physical servers) for variable expenses, and only pay for IT as you consume it.

Deploy globally in minutes

With the cloud, you can expand to new geographic regions and deploy globally in minutes

- Over the past 60 years, computing technology has undergone a series of platform and environment changes
- There have been number of changes in machine architecture, operating system platform, network connectivity, and application workload.
- We have evolved from the use of a centralized computer to solve computational problems, to the use of parallel and distributed computing with multiple computers to solve large-scale problems over the Internet.
- The general computing trend is to leverage shared web resources and massive amounts of data over the Internet.

Evolution of Cloud Computing (T1 1.1.1.1)

- Billions of users use the internet everyday and need to be provisioned concurrently with High performance computing capability to a large number of users
- High performance computers (HPC) providing this capability for short amounts of time is no longer optimal.
- High throughput computing (HTC) is what is needed using distributed and parallel computing.
- Since 1990, we see the proliferation and the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds.
- These systems are employed by both consumers and high-end web-scale computing and information services
- The general computing trend is to leverage shared web resources and massive amounts of data over the Internet

High performance computing (HPC) is the ability to process data and perform complex calculations at high speeds (mostly for short amount of time .. Hours/days)

High Performance Compute (HPC) could also be viewed as a system which has components which can provide high performance to applications using it.

High Throughput Compute (HTC) could be viewed as ability to handle large amounts of computing, for much longer times (months and years, rather than hours and days)

Evolution of Cloud Computing (T1 1.1.1.1)

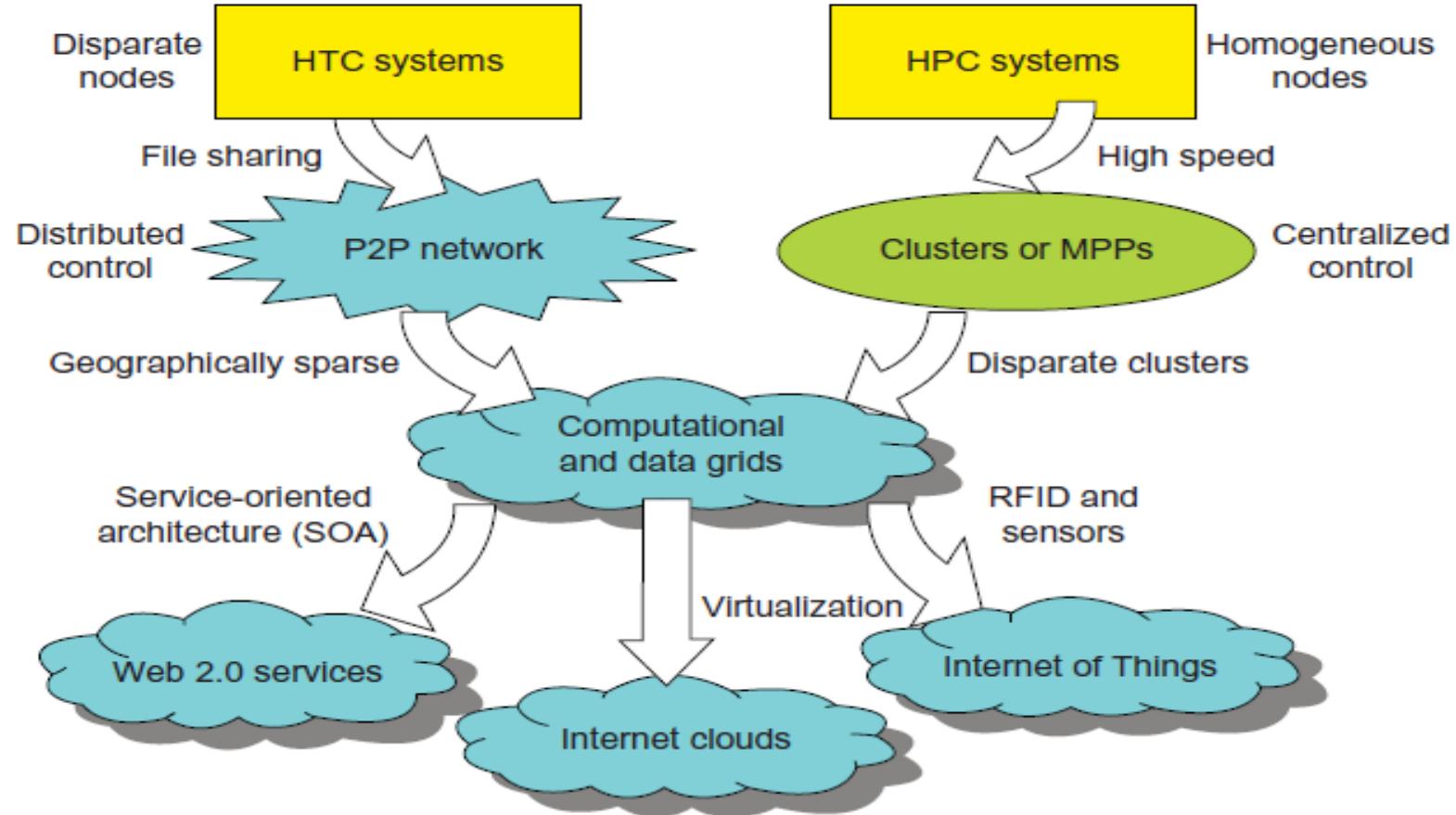


FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

Evolution of Computing - High Performance Computing - Context

- HPC generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.
- A helpful way to help understand what high performance computers are is to think about the what's in them. You have all of the elements you'd find on your desktop — processors, memory, disk, operating system — just more of them
- High performance computers, which are of interest to small and medium-sized businesses today are really *clusters* of computers.
- People often refer to the individual *computers* in a cluster as *nodes*
- HPC systems are mostly expected to support for short amount of time .. In Hours/days

Evolution of Computing : Distributed Computing

Distributed computing (multiple complete systems)

A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through message passing

Distributed system components can be located on different networked computers that coordinate their actions by communicating via pure HTTP, RPC-like connectors or message queues.

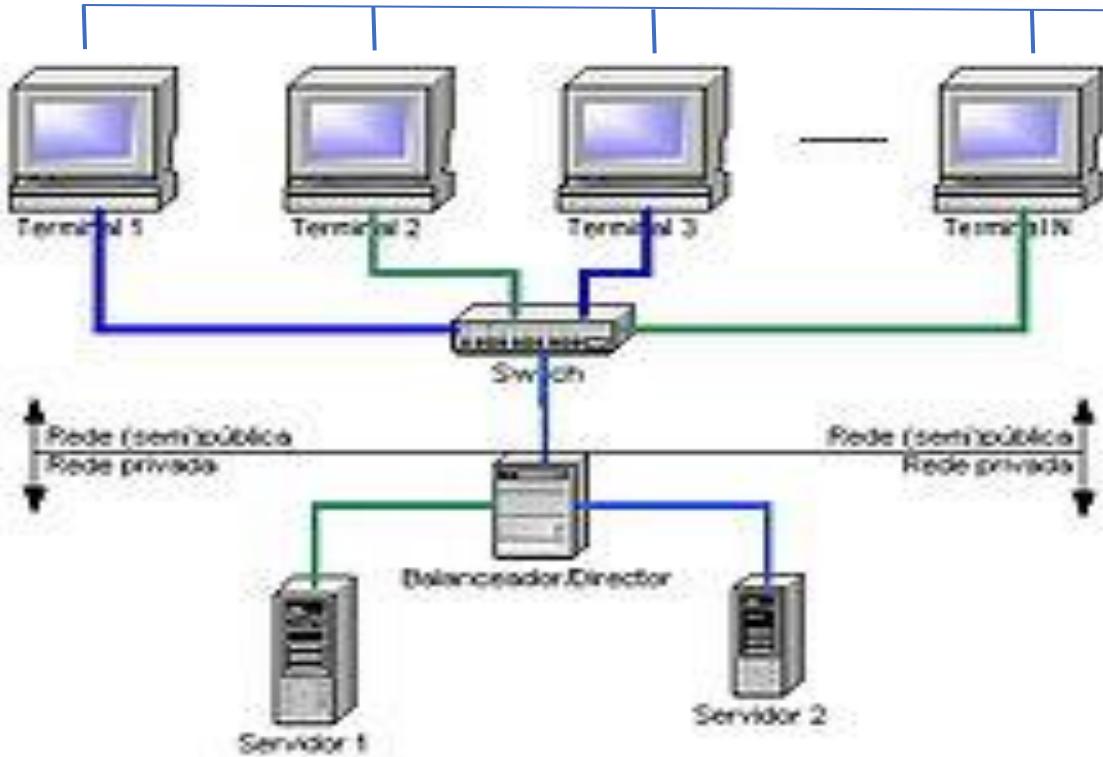
Significant characteristics of distributed systems include independent failure of components and concurrency of components



Distributed Computing

Evolution of Computing : Clusters

A computer cluster is a set of loosely or tightly connected computers that work together so that, in many aspects, they can be viewed as a single system. Clusters have each node set to perform the same task, controlled and scheduled by software.

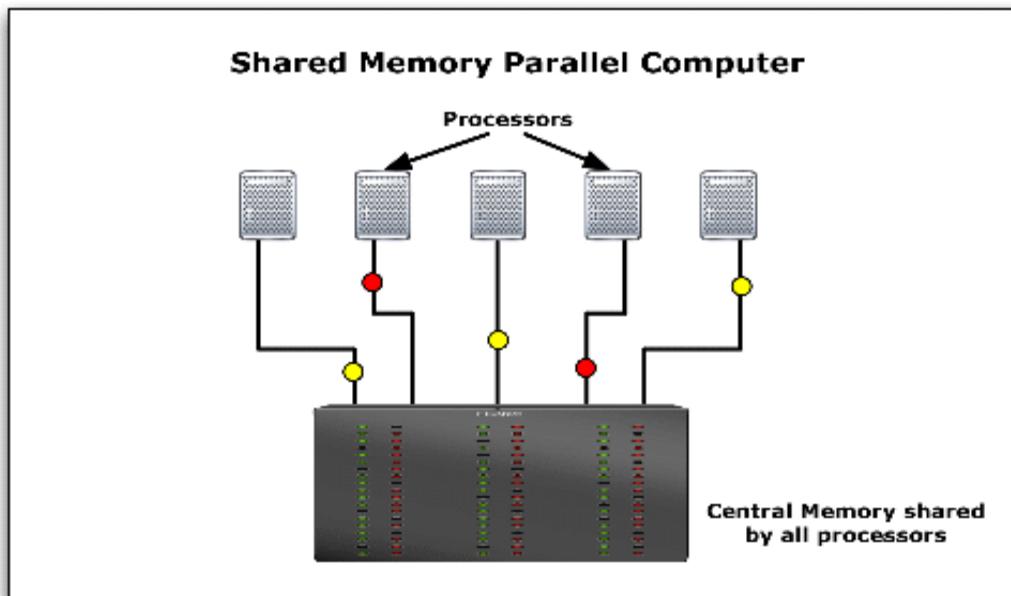


Mostly have similar hardware configuration and run the same OS (different instances though)

Evolution of Computing : Parallel Computing

Parallel Computing (Single system with multiple processors)

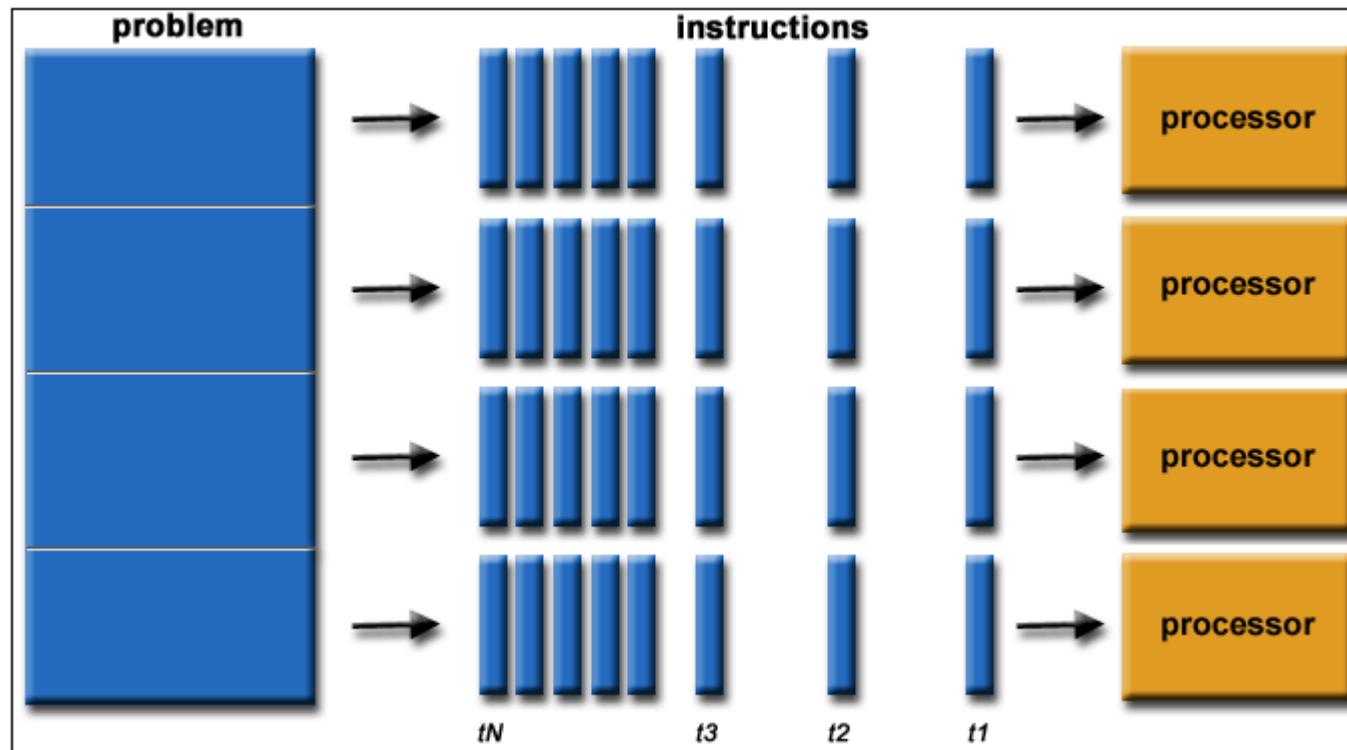
In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Inter- processor communication is accomplished through shared memory or via message passing.



The primary goal of parallel computing is **to increase available computation power** for faster application processing and problem solving

Evolution of Computing : Parallel Computing

In **Parallel Computing** several processors simultaneously execute multiple, smaller calculations broken down from an overall larger, complex problem. These smaller calculations communicate through shared memory and the results of these can be combined as part of an overall program



There are various degrees of parallelism

- Bit Level Parallelism
- Instruction Level Parallelism
- Task Level Parallelism
- Data Parallelism

Evolution of Computing : Parallel Computing

Bit Level Parallelism (<https://www.tutorialspoint.com/types-of-parallelism-in-processing-execution>)

Bit-level parallelism is a form of parallel computing which is based on **increasing processor word size**. In this type of parallelism, with increasing the word size **reduces the number of instructions** the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word.

E.g., consider a case where an 8-bit processor must add two 16-bit integers. First the 8 lower-order bits from each integer were must added by processor, then add the 8 higher-order bits, and then two instructions to complete a single operation. A processor with 16- bit would be able to complete the operation with single instruction.

Instruction Level Parallelism

A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. So we achieve concurrent execution of multiple instructions from a program. While pipelining is a form of ILP, we must exploit it to achieve parallel execution of the instructions in the instruction stream.

Example

```
for (i=1; i<=100; i= i+1)
    y[i] = y[i] + x[i];
```

This is a parallel loop. Every iteration of the loop can overlap with any other iteration, although within each loop iteration there is little opportunity for overlap.

Data Parallelism

Data Parallelism means concurrent execution of the same task on each multiple computing core.

Let's take an example, summing the contents of an array of size N. For a single-core system, one thread would simply sum the elements $[0] \dots [N - 1]$. For a dual-core system, however, thread A, running on core 0, could sum the elements $[0] \dots [N/2 - 1]$ and while thread B, running on core 1, could sum the elements $[N/2] \dots [N - 1]$. So the Two threads would be running in parallel on separate computing cores.

Task Parallelism

Task Parallelism means concurrent execution of the different task on multiple computing cores.

Consider again our example above, an example of task parallelism might involve two threads, each performing a unique statistical operation on the array of elements. Again The threads are operating in parallel on separate computing cores, but each is performing a unique operation.

Evolution of Computing : Parallel Computing terminologies

Parallel Computing Architecture

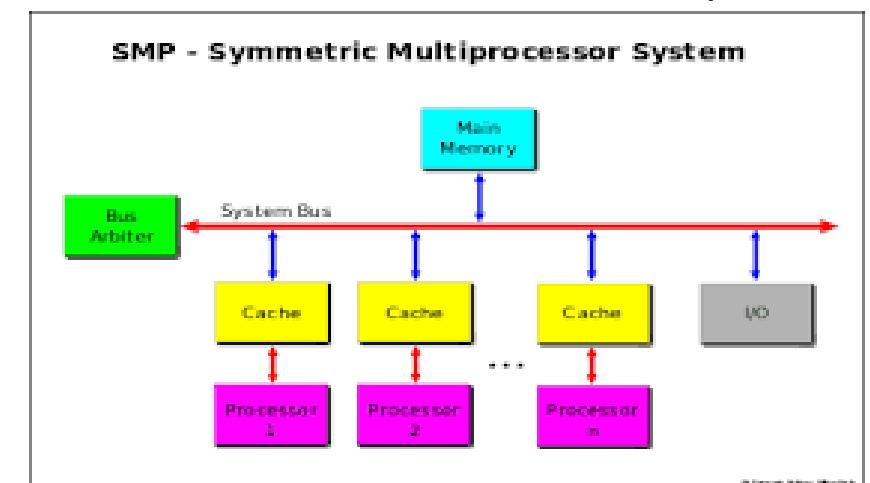
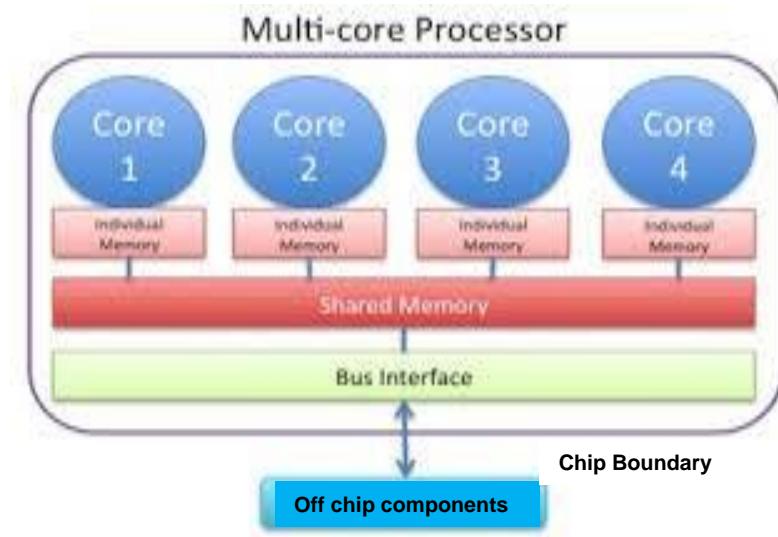
Multi-core computing: A multi-core processor is a computer processor integrated circuit with two or more separate processing cores, each of which executes program instructions in parallel.

Symmetric multiprocessing: multiprocessor computer hardware and software architecture in which two or more independent, homogeneous processors are controlled by a **single operating system** instance that treats all processors equally, and is connected to a **single, shared main memory** with full access to all common resources and devices

Each processor has a private cache memory, may be connected using on-chip mesh networks, and can work on any task no matter where the data for that task is located in memory.

Massively parallel computing: refers to the use of numerous computers or computer processors to simultaneously execute a set of computations in parallel.

One approach involves the grouping of several processors in a tightly structured, centralized computer cluster. Another approach is grid computing, in which many widely distributed computers work together and communicate via the Internet to solve a particular problem.



Parallel Computing Software Solutions and Techniques

- **Application checkpointing:** a technique that provides fault tolerance for computing systems by recording all of the application's current variable states, enabling the application to restore and restart from that point in the instance of failure.
- **Automatic parallelization:** refers to the conversion of sequential code into multi-threaded code in order to use multiple processors simultaneously in a shared-memory multiprocessor (SMP) machine.
- **Parallel programming languages:** Parallel programming languages are typically classified as either distributed memory or shared memory. While distributed memory programming languages use message passing to communicate, shared memory programming languages communicate by manipulating shared memory variables.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Introduction to Cloud Computing Terminologies (Cont.) Grid & Cloud Computing

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

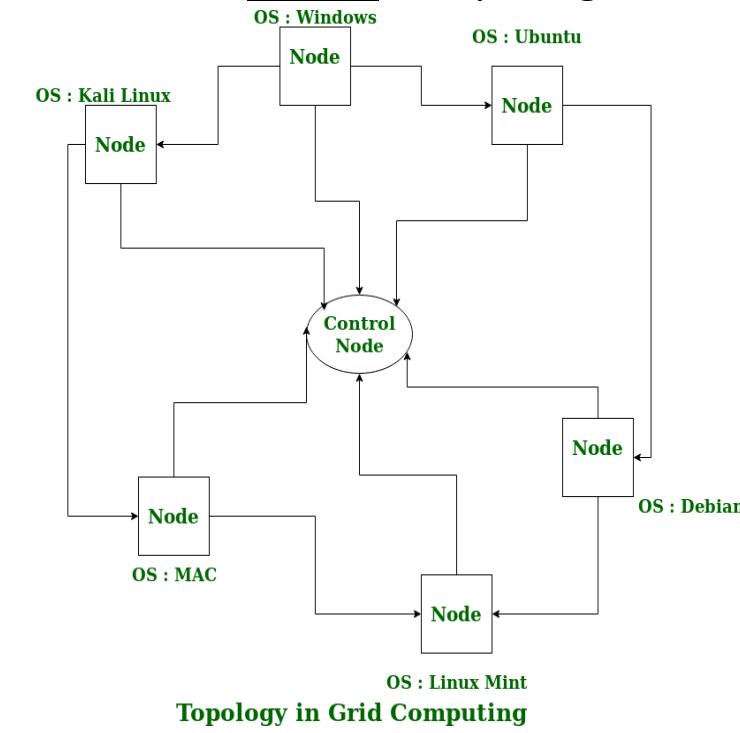
Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Evolution of Computing : Grid Computing

Grid computing is defined as “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*”

- Grid computing is the use of widely distributed computer resources to reach a common goal.
- A computing grid can be thought of as a distributed system with non-interactive workloads that involve many files.
- Grid computing is distinguished from conventional high-performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application.
- Grid computers also tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers
- Establishing **trust and security models** between infrastructure resources pooled from two different administrative domains become even more important.
- **Resource sharing agreements** needed to be formed among a set of participating parties where sharing meant **DIRECT ACCESS** to the resources.
- Sharing was **highly controlled** with resource providers and consumers grouped into virtual organizations primarily based on sharing conditions.



CLOUD COMPUTING

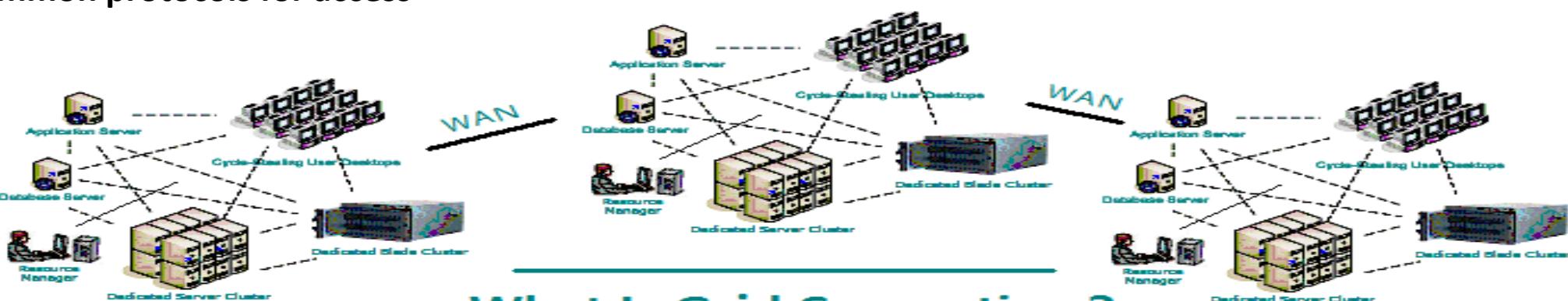
Grid Computing

Cloud computing is frequently compared to grid computing but here are the features of Grid which constrasts.

- The vision of grid computing is to enable **computing to be delivered as a utility**
- grid computing was meant to be used by individual users who gain access to computing devices **without knowing where the resource is located or what hardware it is running**, and so on.
- Initial technological focus of grid computing was **limited to enabling shared use of resources with common protocols for access**

Features of a Grid

1. Co-ordinates resources that are not subject to centralized control
2. Using standard, open, general purpose protocols and interfaces
3. To deliver non-trivial quality of service
4. Uses a common standard for authentication, authorization, resource discovery and resource access



What Is Grid Computing ?

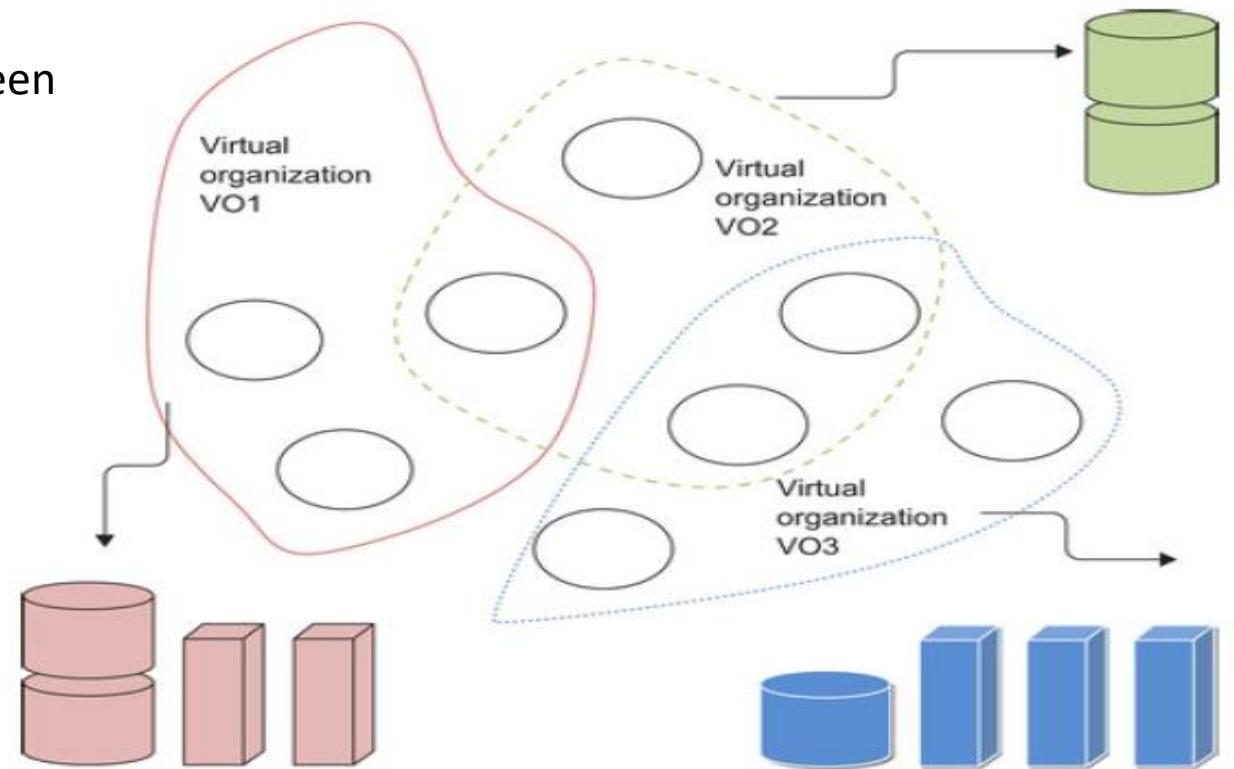
CLOUD COMPUTING

Grid Computing

- Uses the **Concept of Virtual Organization (VO)** which is basically a dynamic collection of individuals or institutions from multiple administrative domains is used to enable flexible, coordinated, secure resource sharing among participating entities (T2)
- A VO forms a **basic unit for enabling access to shared resources** with specific resource-sharing policies applicable for users from a particular VO
- Grid Technology enables sharing of resource between parties who may not have any trust earlier

Benefits

- Exploit Underutilized resources
- Resource load Balancing
- Virtualize resources across an enterprise
- Data Grids, Compute Grids
- Enable collaboration for virtual organizations



Grid Computing : Computational and Data Grid

“A **computational grid** is a **hardware** and **software** infrastructure that provides dependable, consistent, pervasive, and inexpensive access to **high-end computational capabilities**.”

Example : Science Grid (US Department of Energy)

A **data grid** is the storage component of a grid computing system that deals with the **controlled sharing and management of large amounts of distributed data**.

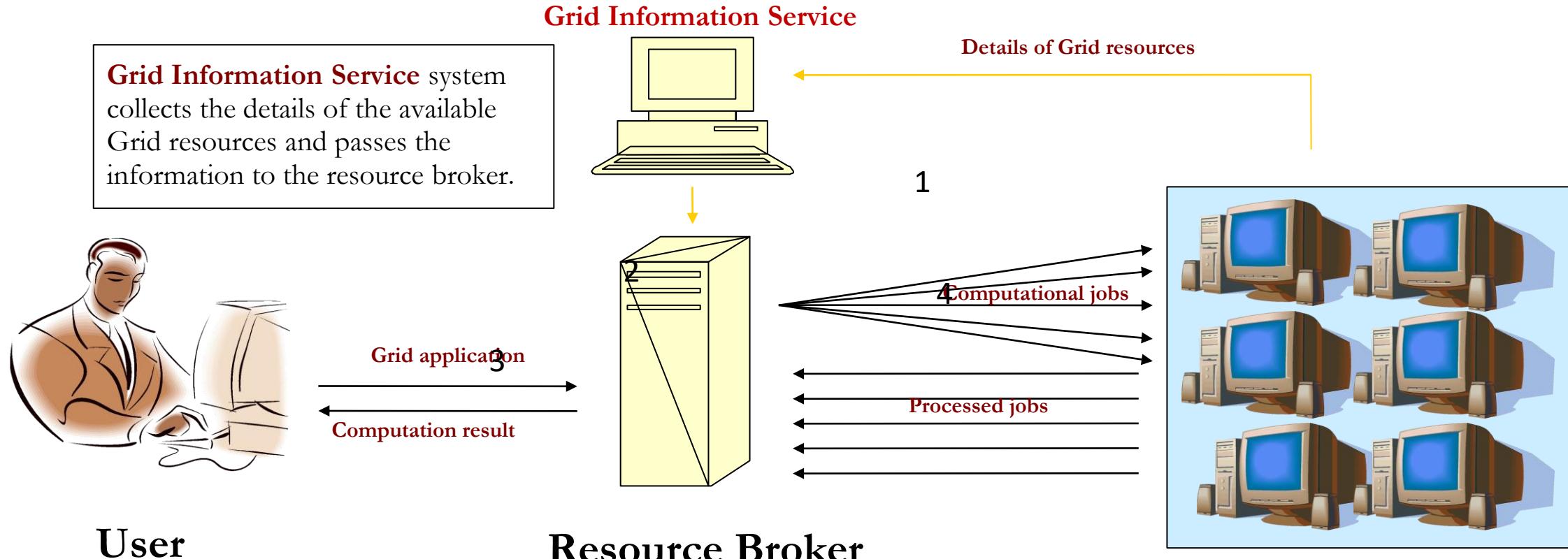
Scientific and engineering applications require access to large amounts of data, and often this data may be widely distributed. A data grid provides seamless access to the local or remote data required to complete compute intensive calculations.

Examples of Data Grid's :

Biomedical informatics Research Network (BIRN),

Southern California earthquake Center (SCEC).

Grid Computing : A typical view of the Grid environment



A **User** sends computation or data intensive application to Global Grids in order to speed up the execution of the application.

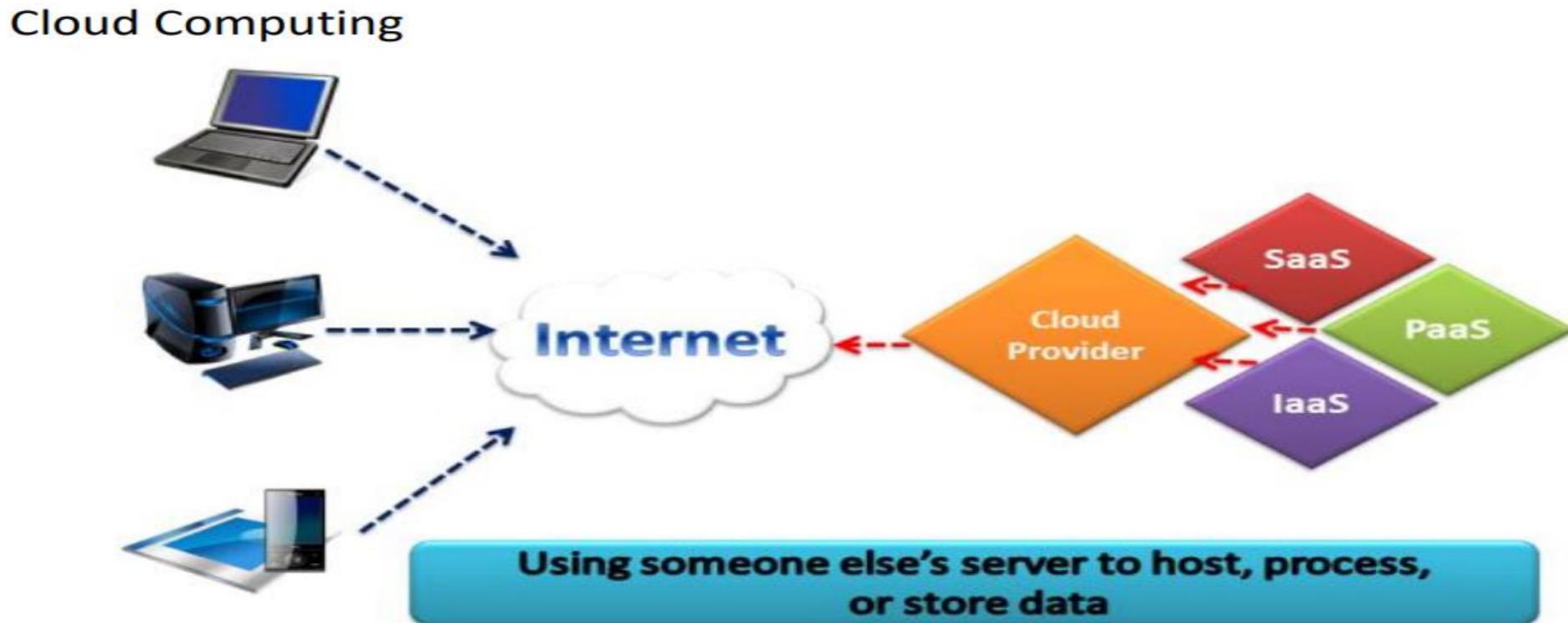
A **Resource Broker** distribute the jobs in an application to the Grid resources based on user's QoS requirements and details of available Grid resources for further executions.

Grid Resources (Cluster, PC, Supercomputer, database, instruments, etc.) in the Global Grid execute the user jobs.

Evolution of Computing : Cloud Computing

Cloud Computing

An Internet cloud of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed.



Evolution of Computing : Cloud Computing

Applications of Cloud Computing (T1 1.1.2.2)

Domain	Specific Applications
Science and engineering	Scientific simulations, genomic analysis, etc. Earthquake prediction, global warming, weather forecasting, etc.
Business, education, services industry, and health care	Telecommunication, content delivery, e-commerce, etc. Banking, stock exchanges, transaction processing, etc. Air traffic control, electric power grids, distance education, etc. Health care, hospital automation, telemedicine, etc.
Internet and web services, and government applications	Internet search, data centers, decision-making systems, etc. Traffic monitoring, worm containment, cyber security, etc. Digital government, online tax return processing, social networking, etc.
Mission-critical applications	Military command and control, intelligent systems, crisis management, etc.

Cloud Computing Design Objectives

Design objectives of cloud computing

- **Efficiency** measures the utilization rate of resources in an execution model by exploiting massive parallelism in HPC. For HTC, efficiency is more closely related to job throughput and power efficiency. All of these at lowest cost.
- **Dependability** measures the reliability and self-management from the chip to the system and application levels. The purpose is to provide high-throughput service with Quality of Service (QoS) assurance, even under failure conditions.
- **Adaptation** in the programming model measures the ability to support billions of job requests over massive data sets and virtualized cloud resources under various workload and service models.
- **Flexibility** in application deployment measures the ability of distributed systems to run well in both HPC (science and engineering) and HTC (business) applications.

Evolution of the Web (A key enabler in cloud computing)

Web 1.0

Four design essentials of a Web 1.0 site include:

1. Static pages.
2. Content is served from the server's file-system.
3. Pages built using Server Side Includes or Common Gateway Interface (CGI).
4. Frames and Tables used to position and align the elements on a page.

Web 2.0 –

Web 2.0 refers to world wide website which highlight user-generated content, usability & interoperability for end users. Web 2.0 is also called participative social web. Ajax & JavaScript frameworks extensively used in Web 2.0

Major features of Web 2.0 –

1. Permits users to retrieve and classify the information collectively.
2. Dynamic content that is responsive to user input.
3. Information flows between site owner and site users by means of evaluation & online commenting.
4. Developed APIs to allow self-usage, such as by a software application.

Web 3.0

1. Semantic Web

Generate, share and connect content through search and analysis based on **the ability to understand the meaning of words**, rather than on keywords or numbers.

2. Artificial Intelligence

Combining this capability with natural language processing, in Web 3.0,

3. 3D Graphics

The three dimensional design is being used extensively in websites and services in Web 3.0. Museum guides, computer games, ecommerce, geospatial contexts, etc. are all examples that use 3D graphics.

4. Ubiquity

Content is accessible by multiple applications, every device is connected to the web, the services can be used everywhere.

Contrasting Cloud Computing and Grid Computing

Cloud Computing	Grid Computing
Cloud Computing follows client-server computing architecture.	Grid computing follows a distributed computing architecture.
Scalability is high.	Scalability is normal.
Cloud Computing is more flexible than grid computing.	Grid Computing is less flexible than cloud computing.
Cloud operates as a centralized management system.	Grid operates as a decentralized management system.
In cloud computing, cloud servers are owned by infrastructure providers.	In Grid computing, grids are owned and managed by the organization.
Cloud computing uses services like IaaS, PaaS, and SaaS.	Grid computing uses systems like distributed computing, distributed information, and distributed pervasive.
Cloud Computing is Service-oriented.	Grid Computing is Application-oriented.
It is accessible through standard web protocols.	It is accessible through grid middleware.

Compute as an Utility (T1 1.1.2.3)

Utility computing focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers.

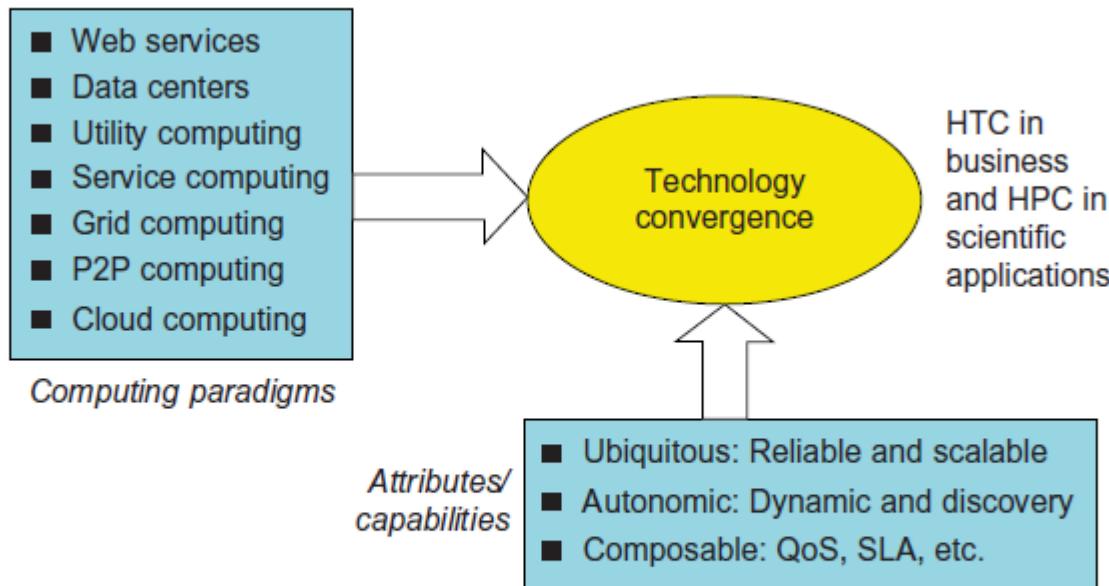


FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

(Modified from presentation slide by Raj Buyya, 2010)



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

CLOUD COMPUTING MODELS

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Cloud Computing Models

Recap: Cloud computing is a model for enabling ubiquitous , convenient, on demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction

- NIST (National Institute of Standards)

Key Terms

Ubiquitous - Wherever

On Demand - When

Shared Pool of configurable resources

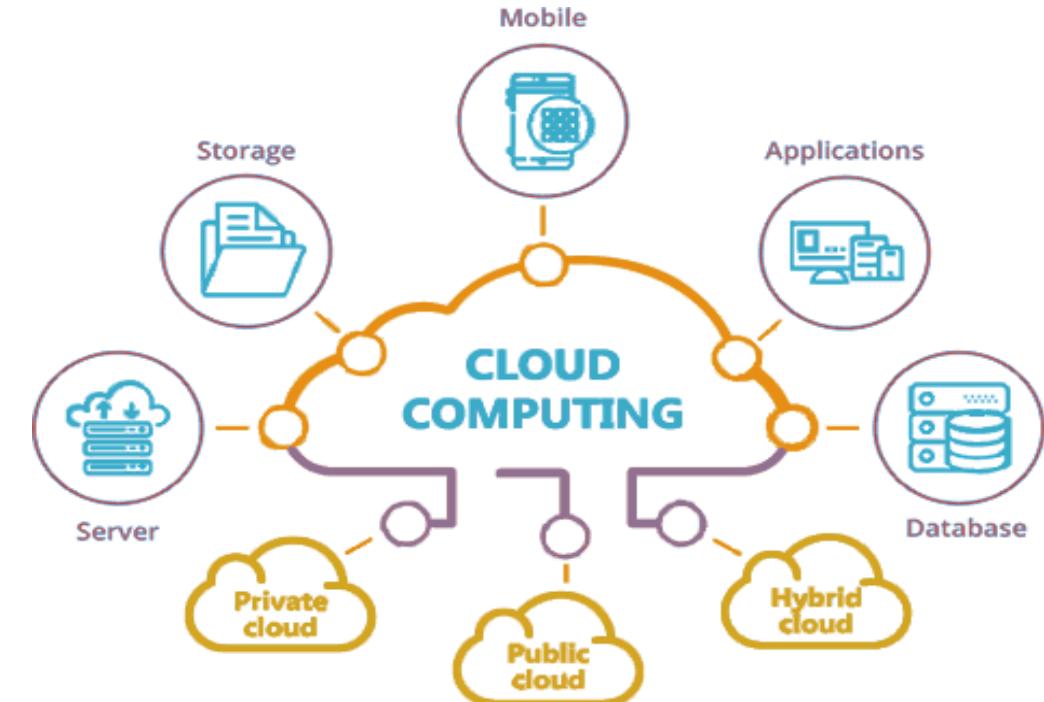
Rapidly Provisioned

Deployment Models

1. Private Cloud – where computing resources are deployed for one particular organization and are governed, owned and operated by the same organization.

2. Public Cloud – where the computing resource is owned, governed and operated by government, an academic or business organization and the resources are available for any individual/organization willing to pay

3. Hybrid Cloud where the cloud resources can be used for both type of interactions – B2B (Business to Business) or B2C (Business to Consumer). This deployment method is called hybrid cloud as the computing resources are bound together by different clouds.

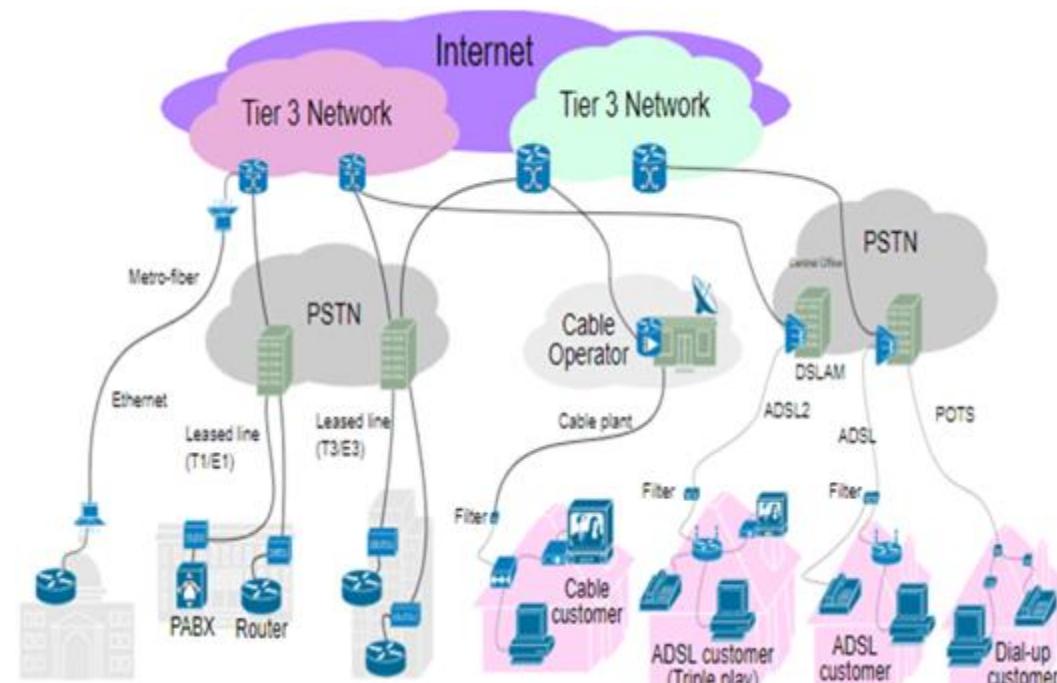


Cloud Computing Models : Enabling Technologies

Technologies that enable cloud computing

1. Broadband networks and internet architecture

1. All clouds must be connected to a network
2. Internet – can be visualized as a network of **autonomous co-operative** networks which are interconnected by routers
3. Core of the internet would be a connectionless packet switching Router based network. The edge devices like systems and desktops connected through switches and routers to local ISPs, which in turn are connected to regional ISPs and finally to a series of large backbone ISPs who interconnect through cables across ocean floors/countries.



2. Data center technology (# of Devices .. could be commodity devices forming the Infrastructure)

3. Virtualization technology (Server, Storage and Network)

4. Web technology (Uniform resource locator (URL), Hypertext transfer protocol (HTTP), Markup languages (HTML, XML))

5. Multitenant technology (Tenants on the same server)



Cloud Computing Models

There are typically 3 Cloud Service Models/Classifications (T2)

- **Infrastructure as a service (IAAS)**

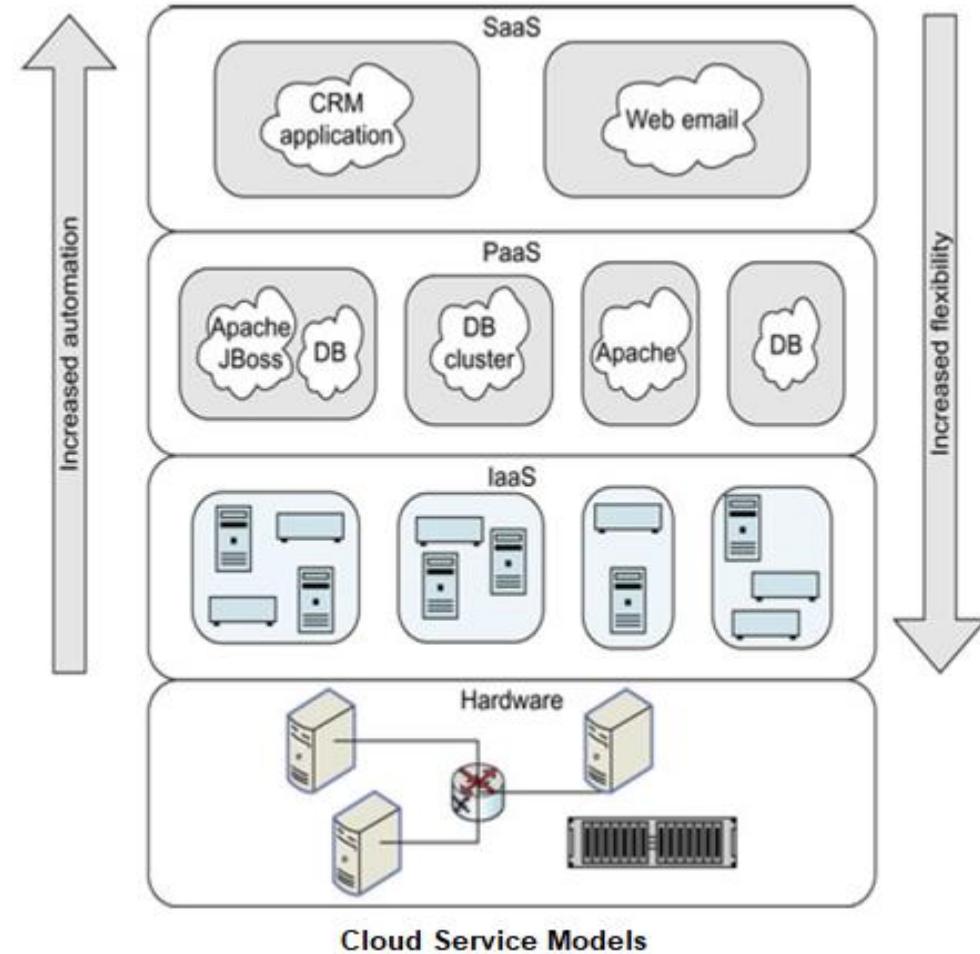
The physical hardware (servers, disks, and networks) is abstracted into virtual resources and allocated

- **Platform as a service (PAAS)**

Provides a platform built on top of the abstracted hardware that can be used by developers to create cloud applications
Commands provided allow allocation of middleware servers (e.g., a database of a certain size), configure and load data into the middleware

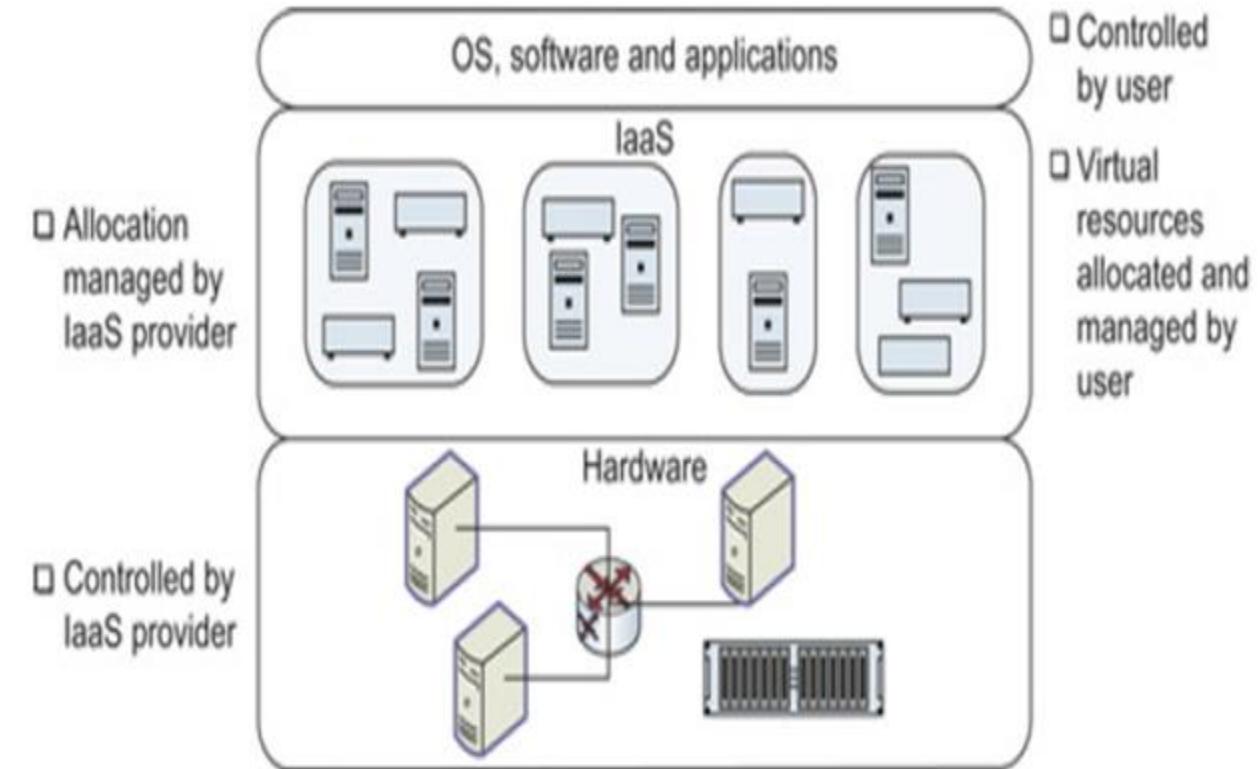
- **Software as a service (SAAS)**

Provides the complete application (or solution) as a service, enabling consumers to use the cloud without worrying about all the complexities of hardware, OS or even application installation



Cloud Computing Models – IaaS (Infrastructure as a Service)

- IaaS is a way to deliver a cloud computing infrastructure like server, storage, network and operating system.
- The customers can access these resources over cloud computing platform i.e Internet as an on-demand service.
- The customer buy's these cloud resources rather than buying space, server, software, or network equipment.
- The customer then deploys and runs arbitrary software, which can include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls)
- The IaaS provider has control over the actual hardware and the cloud user can request allocation of virtual resources



Cloud Computing Models – IaaS (Infrastructure as a Service) (Cont.)

- The user of IaaS has single ownership of the hardware infrastructure allotted to him (may be a virtual machine)
- It is a Cloud computing platform based model.

Advantages of IaaS

- In IaaS, user can dynamically choose a CPU, memory storage configuration according to need.
- Users can easily access the vast computing power available on IaaS Cloud platform.
- IaaS is well suited for users who want complete control over the software stack that they run

Disadvantages of IaaS

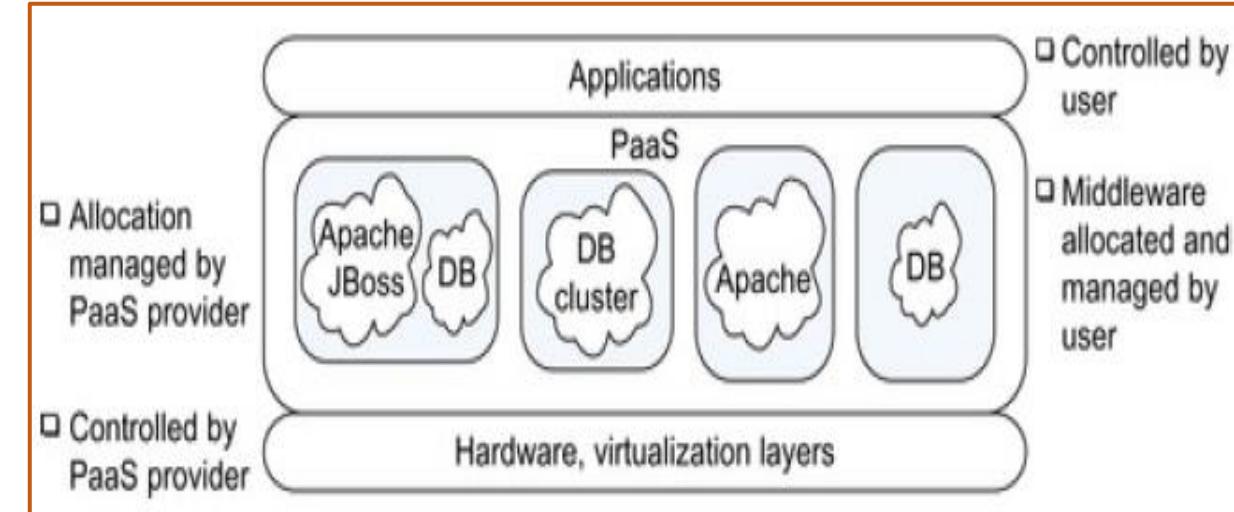
- IaaS cloud computing platform model is dependent on availability of Internet & virtualization services.

Examples of IaaS Service Providers

- Amazon Web Services.
- Microsoft Azure.
- Google Cloud.
- IBM Cloud.
- Oracle Cloud.

Cloud Computing Models – PaaS (Platform as a Service)

- The PaaS model is to provide a system stack or platform for application deployment as a service
- PaaS is a programming platform for developers for developing applications. This platform is generated for the programmers to create, test, run and manage the applications.
- The cloud user using the PaaS can configure and build on top of this middleware, such as define a new database table in a database
- A developer can easily write the application and deploy it directly into PaaS layer.
- Summarily the user does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment (say the database) configurations
- The hardware, as well as any mapping of hardware to virtual resources, such as virtual servers, is controlled by the PaaS provider. The PaaS provider also provides the supported middleware, such as a database, web application server



Cloud Computing Models – PaaS (Platform as a Service) (Cont.)

PaaS platforms are well suited to those cloud users who find that the middleware they are using matches the middleware provided by one of the PaaS vendors

Advantages of PaaS

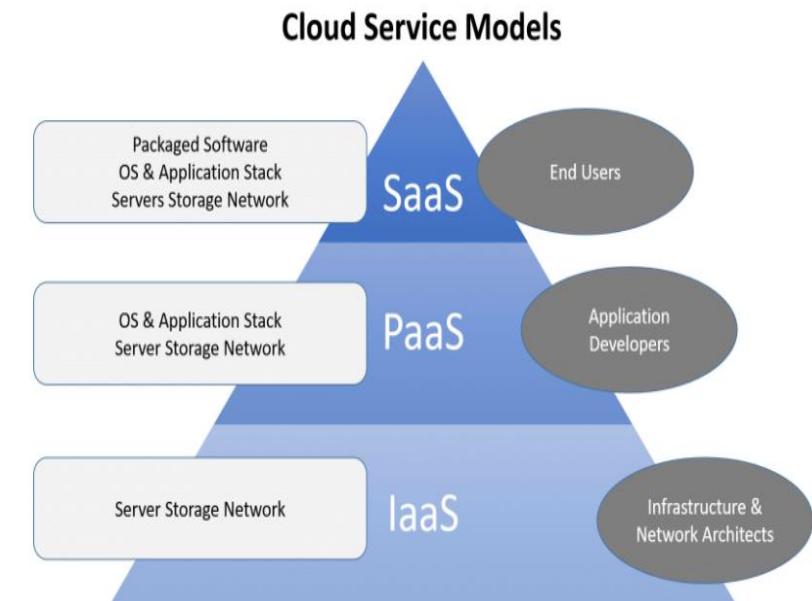
- PaaS platforms are well suited to those cloud users who find that the middleware they are using matches the middleware provided by one of the PaaS vendors
- PaaS is easier to develop. Developer can concentrate on the development and innovation without worrying about the infrastructure.
- In PaaS, developer only requires a PC and an Internet connection to start building applications.

Disadvantages of PaaS

- One developer can write the applications as per the platform provided by PaaS vendor hence the moving the application to another PaaS vendor is a problem.

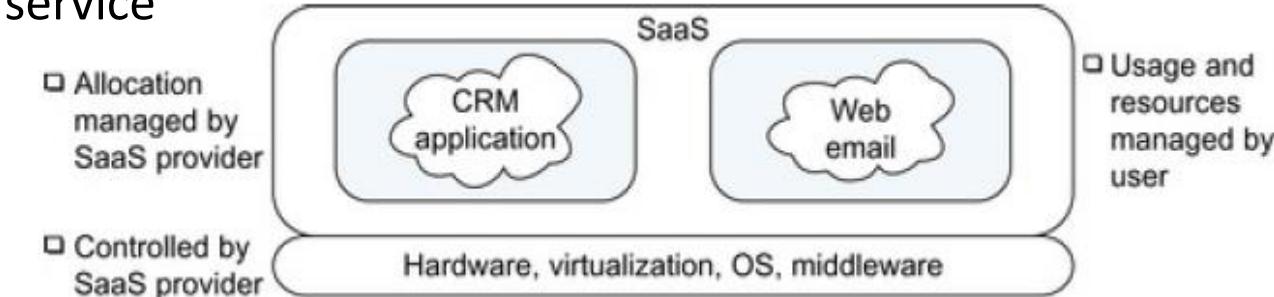
Examples

- Google Apps Engine(GAE), Windows Azure, SalesForce.com are the examples of PaaS.



Cloud Computing Models – SaaS (Software as a Service)

- SaaS is about providing the complete application as a service
- In this model, the provider's application running on a Cloud Infrastructure is publicized and would be used by the users over the Internet.
- The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email).
- In SaaS, associated data and software are also mostly hosted centrally on the cloud server and typically can be considered as a software distribution model
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.
- Users who log into the SaaS service can both use the application as well as configure the application for their use
- When configuration settings are changed, the SaaS infrastructure performs any management tasks needed (such as allocation of additional storage) to support the changed configuration



Advantages of SaaS

- SaaS is easy to buy because the pricing of SaaS is based on monthly or annual fee and it allows the organizations to access business functionalities at a small cost, which is less than licensed applications.
- SaaS needed less hardware, because the software is hosted remotely, hence organizations do not need to invest in additional hardware.
- Less maintenance cost is required for SaaS and do not require special software or hardware versions.
- SaaS is almost a no programming Model

Disadvantages of SaaS

- SaaS applications are totally dependent on Internet connection. They are almost un-useable without Internet connection.
- It is difficult to switch amongst the SaaS vendors.

Examples

- CRM, Office Suite, Email etc. are software applications which are provided as a service through Internet.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

TECHNOLOGIES CHALLENGES & BUSINESS DRIVERS

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

CLOUD COMPUTING

Technology Challenges

- Cloud Computing as discussed earlier with so many advantages also has many challenges.
- These changes could come up from the infrastructure to support requests, or on account of the data-information in the cloud or it could be for effective usage of the same in Businesses
- We will look at few of these listed below
 - Scalability
 - Elasticity
 - Performance Unpredictability
 - Reliability and Availability
 - Security
 - Compliance
 - Multi-Tenancy
 - Lack of Expertise
 - Cost Management
 - Internet Connectivity
 - Password Security
 -
- These are challenges for all the 3 computing models

As and when the challenges occur these are getting addressed and these are active research areas

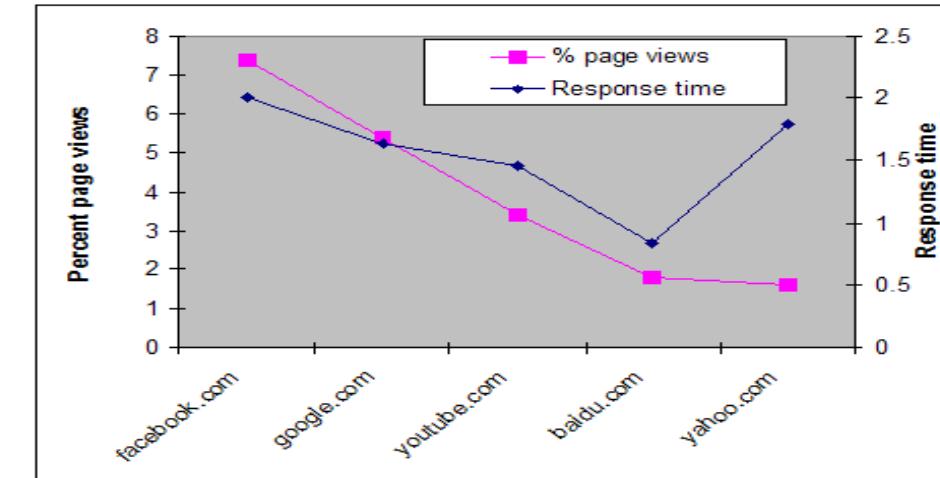
Recap - Scalability is the ability to accommodate larger or smaller loads while supporting some of the expectations of QoS like response time.

This will need to happen supporting

- Scale with the spread of wide range of environments
- Sharing of the same with many users
- Across significantly large computing environments

Consider as an example Facebook or Google,

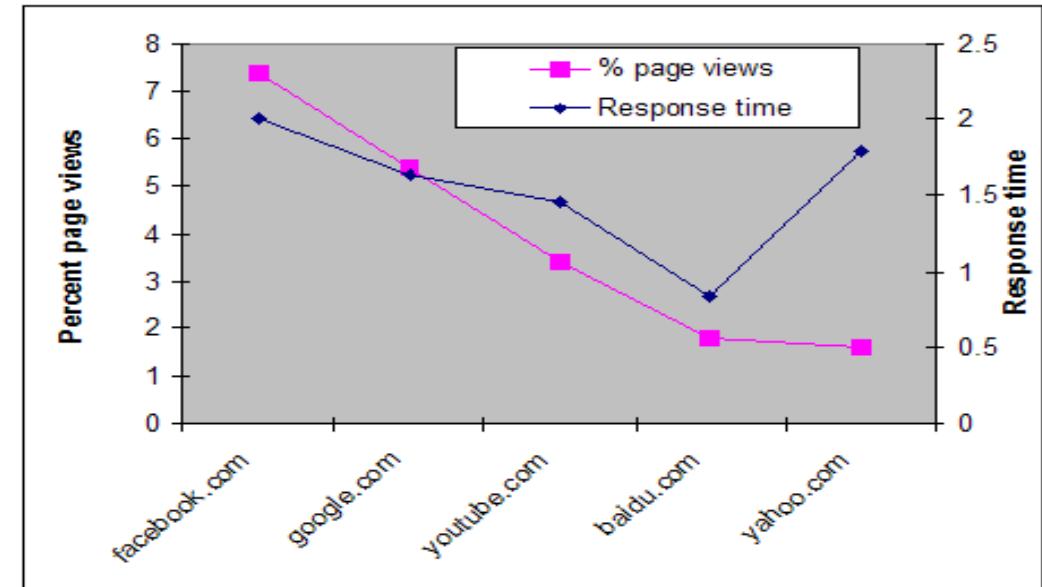
- Many Web sites have high traffic
 - Facebook – 7.5%
 - Google – 5.5%
- But they still maintain good response time
 - Facebook ~2 sec
 - Google ~1.7 sec
- Large amounts of storage
 - Google has many copies of Web
 - Facebook adds ~TB per day of storage



Recap - Elasticity is all about the actual increasing or decreasing of resources to cope with loads dynamically

Continuing with the example of Facebook or Google

- How to scale up (scale out) and scale down quickly?
 - Scale up (Vertical Scaling)
 - Scale Out (Horizontal Scaling)
 - Scale down
- Resource allocation and workload placement algorithms are required.
- Not happening quickly can impact QoS or Cost



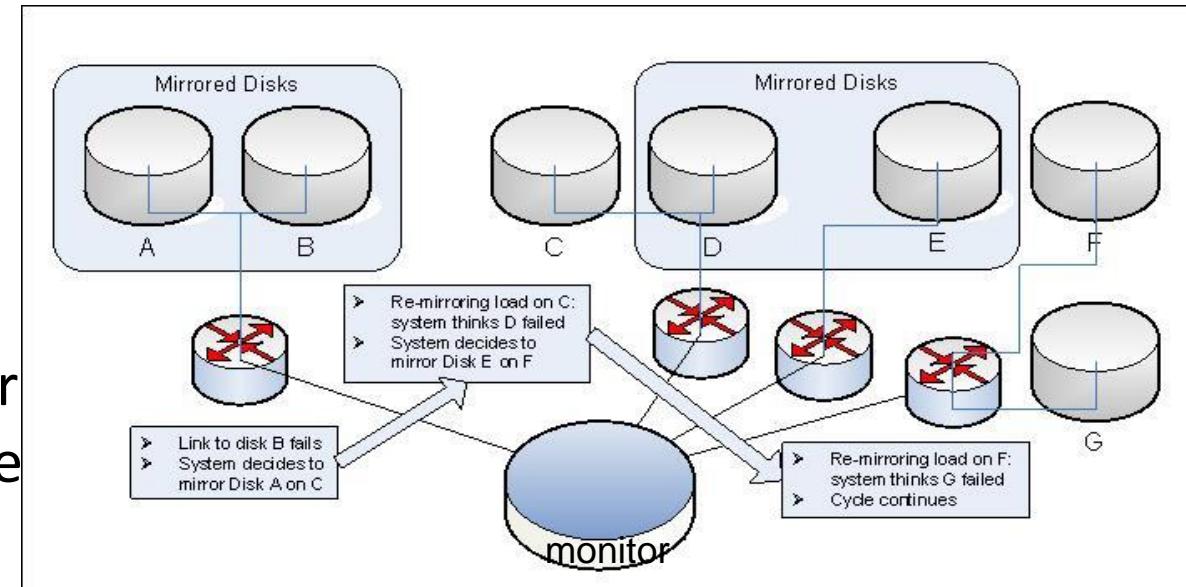
Technology Challenges : Performance Unpredictability

- Resources are shared
- How to guarantee performance isolation
 - Since physical hardware is shared.
 - Many VMs may run concurrently.
 - And affect each other at run time



- Most of the businesses are dependent on services provided by third-party, hence it is mandatory for the cloud systems to be reliable and robust.
- In large-scale environments, hardware failures and software bugs can be expected to occur relatively frequently.
- The problem is complicated by the fact that failures can trigger other failures, leading to an avalanche of failures that can lead to significant outages.
- Factors affecting reliability and availability
 - High number of components
 - Complexity

Eg. Consider diagram where few of the disks are mirrored and we have a process monitoring the load and behavior of disks



Availability

Service is important and many businesses run mission critical applications on the cloud

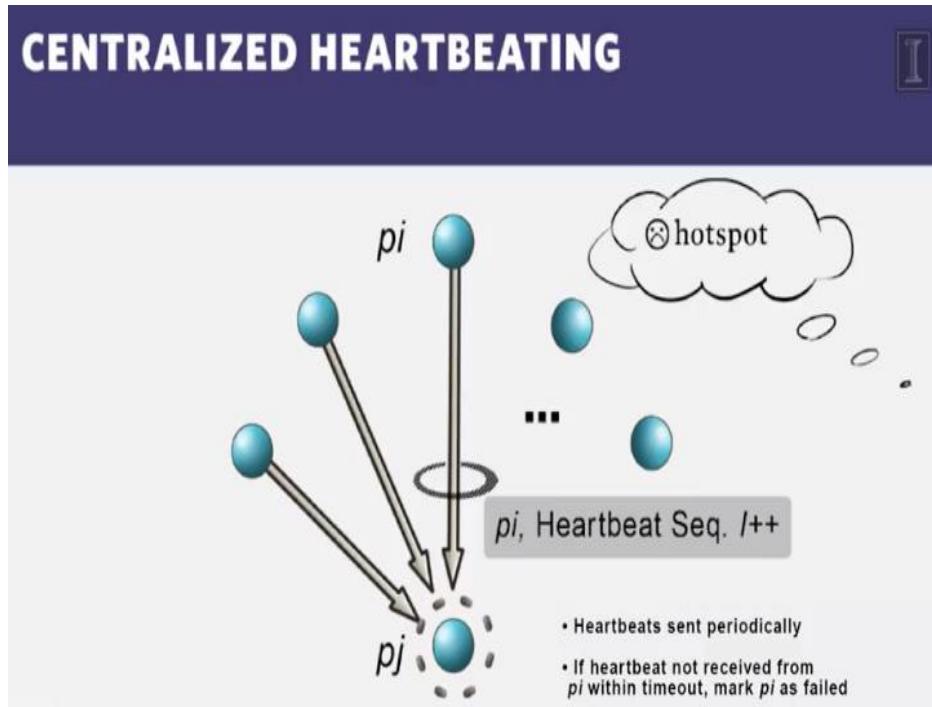
Availability is often achieved by using redundancy at the infrastructure , middleware or at the application level

There are 2 techniques that cloud models use for ensuring high availability

1. The first technique is **failure detection**, where the cloud infrastructure detects failed application instances, and avoids routing requests to such instances
2. The second technique is **application recovery**, where failed instances of application are restarted.

There are two techniques **of failure monitoring** [T2].

Failure Monitoring Heart Beats



The first method is **heartbeats**, where each application instance periodically sends a signal (called a heartbeat) to a monitoring service in the cloud. If the monitoring service does not receive a specified number of consecutive heartbeats, it may declare the application instance as failed.

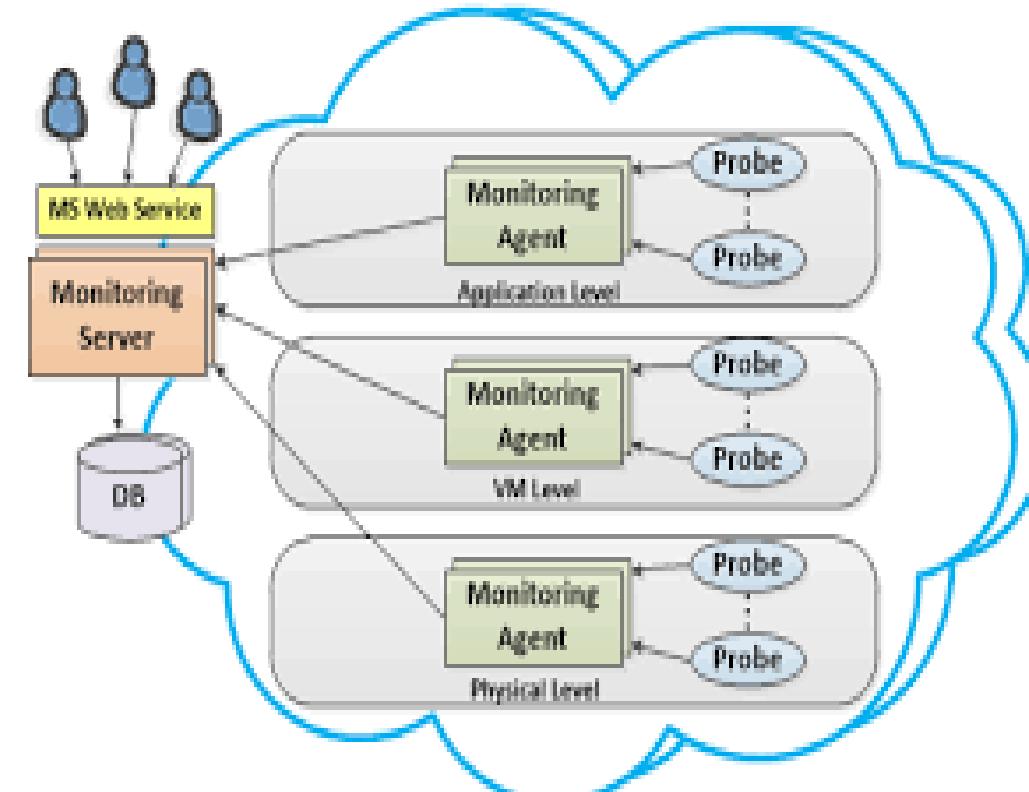
Probes

The second is the method of **probes**. Here, the monitoring service periodically sends a probe, which is a lightweight service request, to the application instance. If the instance does not respond to a specified number of probes, it may be considered failed.

There is a trade-off between speed and accuracy of detecting failures. To detect failures rapidly, it may be desirable to set a low value for the number of missed heartbeats or probes.

Recovery

Redirection: After identifying failed instances, it is necessary to avoid routing new requests to these instances. A common mechanism used for this in HTTP-based protocols is HTTP-redirection.



https://www.researchgate.net/figure/Abstract-Architecture-View-monitoring-metrics-from-the-Cloud-element-they-reside-on-and_fig1_271428939

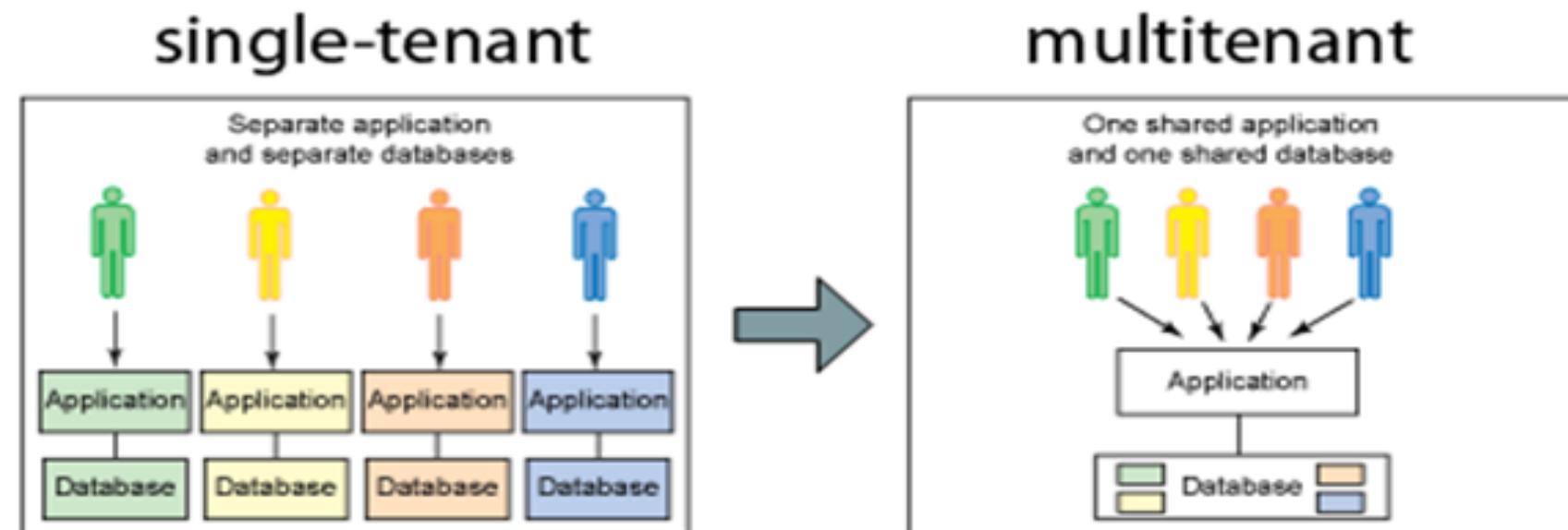
- Considerations towards violation of confidentiality, data privacy, and data leakage and loss
- Is security possible?
 - Data is getting stored and processes by a third party vendor whom you cannot see and a lot of times you are unaware of the location where its stored or processed
 - Vast amount of data will need to be travelling on networks
- Sony incident
 - Hacker used Amazon to hack Sony web site
 - Can we be secure if we are in same cloud as hacker?
- Isolation of users
- Legal and process issues
 - Physical security
- Cloud Service providers are providing auditable and safe identity management, access control procedures for authentication and authorization, Use firewalls, encryptions, privacy protocols, recovery policies, SLAs etc.

Technology Challenges : Compliance

- Compliance is the process of meeting the requirements of the service users or it could be the laws of the country.
- Cloud technologies will need to enable business operations to comply with the expectations of a customer or the laws of the land.
- The challenge for an user would be to know if a cloud provider is complying with privacy rules or the laws and for the Cloud service provider to be enabled by technology for compliance.
- E.g.
Healthcare organizations in the USA for **protecting and securing health Information** have to comply with HIPAA (Health Insurance Portability and Accountability Act).
- Sarbanes Oxley 2002 which sets the process and oversight mechanisms for **audit and reporting standards** for public companies to **prevent accounting frauds** post Enron and few scandals.
- India SEBI Clause 49 (**Corporate Governance Practices**)

Multitenancy is the mode of operation where a single instance of the component serves multiple tenants or groups of users. The instances (tenants) are logically isolated, but physically integrated and the architecture provides every tenant with a dedicated share of the instance, including configuration and data.

- Sharing of same database by multiple users
- Share without compromising security



Interoperability

- The application on one platform should be able to incorporate services from the other platform. This is known as Interoperability.
- It is becoming possible through web services, but to develop such web services is complex.

Portability (Migration)

- The applications running on one cloud platform can be moved to new cloud platform and it should operate correctly without making any changes in design, coding.
- The portability is not possible, because each of the cloud providers uses different standard languages for their platform.

Network Capability and Computing Performance

- High network bandwidth is needed for data intensive applications on cloud, this results in high cost.
- In cloud computing, low bandwidth does not meet the desired computing performance.

Application Recovery

- In addition to directing new requests to a server that is up, it is necessary to recover old requests
- An application independent method of doing this is **checkpoint/restart**. Here, the cloud infrastructure periodically saves the state of the application
- If the application is determined to have failed, the most recent checkpoint can be activated, and the application can resume from that state.
- Checkpointing is also present in certain middleware such as Dockers.
- Global and local snapshots are taken from which recovery is performed. There are off course challenges associated with Checkpointing.

CLOUD COMPUTING

Business Drivers

Benefits	Group	Drawbacks
Speed to Market	O P E R A T I O N A L	Security
Scalability		Compliance
Agility to new requests and need for capacity		Vendor Lock-In
Automated		
Flexibility to support changing scales		
Assurance – Assurance on account of the expertise and experience in technical and management of the Cloud Infrastructure by the specialized service provider		
Supports increase of Total Customer Experience		
Different Deployment Models like Private Cloud for Single Enterprise, Public Cloud for Service providers and users, Hybrid Cloud for a Mix and Commodity for a group		
Pay Per Use		
Lower Capital Investment		
Reduced Financial Risk	C O S T	
Reduced Cost		
Significantly Automated Management		
Expansion of the customer global footprint and thus Business		

Worldwide Public Cloud Revenue Forecast (Billions of US\$)

Table 1. Worldwide Public Cloud Service Revenue Forecast (Billions of U.S. Dollars)

	2017	2018	2019	2020	2021
Cloud Business Process Services (BPaaS)	42.2	46.6	50.3	54.1	58.1
Cloud Application Infrastructure Services (PaaS)	11.9	15.2	18.8	23.0	27.7
Cloud Application Services (SaaS)	58.8	72.2	85.1	98.9	113.1
Cloud Management and Security Services	8.7	10.7	12.5	14.4	16.3
Cloud System Infrastructure Services (IaaS)	23.6	31.0	39.5	49.9	63.0
Total Market	145.3	175.8	206.2	240.3	278.3

BPaaS = business process as a service; IaaS = infrastructure as a service; PaaS = platform as a service; SaaS = software as a service

Note: Totals may not add up due to rounding.

Source: Gartner (September 2018)

Comparison of public vs private cloud

Table 1.1 Hypothetical Cost of Public Vs Private Cloud

(in USD)	Private Cloud			Public Cloud		
	Year 1	Year 2	Year 3	Year 1	Year 2	Year 3
Hardware	70,000	40,000	20,000			
Setup Costs	30,000			5,000		
Software (Licensing)	200,000	400,000	700,000			
Labor costs	200,000	200,000	200,000			
Service costs				300,000	600,000	1,000,000
WAN costs				15,000	30,000	56,000
Cost for year	500,000	640,000	920,000	305,000	600,000	1,000,000
Total	2,060,000			2,006,000		



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

**Public, Private and Hybrid Cloud
Cloud Deployment Models**

Dr. H.L. Phalachandra

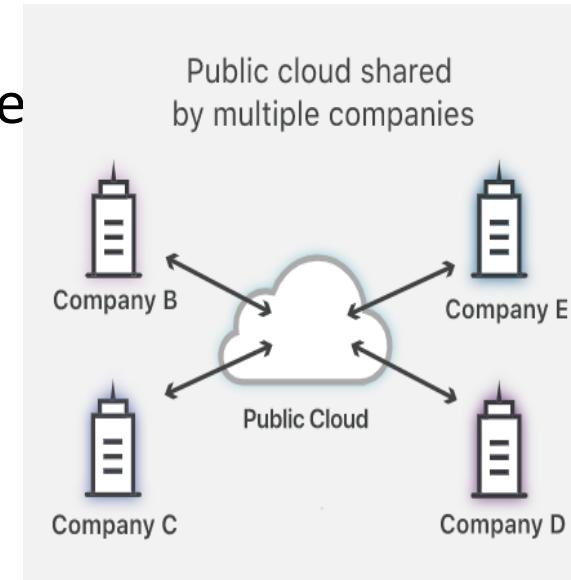
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Public Cloud: It's a platform or an IT model where a set of infrastructure resources located in one or more different physical locations are made available as a service in a pay-as-per-use model to multiple general public customers by a cloud service provider.

- The infrastructure resources are provided as a service, say as software (SaaS), as software platforms on which applications can be built and run (PaaS) or as virtual IT components like compute or storage (IaaS)
- Any user who wants to use it is provided with this shared public platform accessible through the internet
- The cloud user pays the cloud vendor for using the infrastructure.
- The resources of the public cloud is owned and managed by the cloud vendor or also called service provider.
- In a public cloud, these are made available through APIs or through a self-service interface.



A public cloud is like renting an apartment. It's public from the perspective that the resources are publicly available to anyone, available to be used on a cost basis and is available on the public Internet.

Characterizing this a little more :

1. **Shared Resource Allocation** – Users or tenants outside the provider's firewall share cloud services and virtual resources that come from the provider's set of infrastructure, platforms, and software.
2. **Usage Agreements** – Some of the resources available could be costed and may cost based on the usage in the pay-as-you-use, there could be some which may be available at no cost. (Example: Massachusetts Open Cloud).
3. **Management** - At a minimum, the provider maintains the hardware underneath the cloud, supports the network, and manages the virtualization software.

Advantages/Motivations for a Public Cloud

1. Cost :

- Public cloud has a **lower** cost than private, or hybrid cloud, as it shares the same infrastructure resources with a large number of consumers and thus gets the benefit of the cost getting distributed
- Public cloud service providers given their volumes and specialization, would handle it more efficiently
- Public cloud costs are also **expense** thus reducing the need for significant investments.

2. Less server management - If an organization uses a public cloud, internal teams don't have to spend time managing servers – as they do for legacy on-premises data centers or for internal private clouds.

3. Time Saving - The cloud service provider is responsible for the management and maintenance of data centers, the client can save the time required to establish connectivity, deploy new products, release product updates, configure servers etc.

4. Location Independence – Given that the resources can exist anywhere, it provides location independence

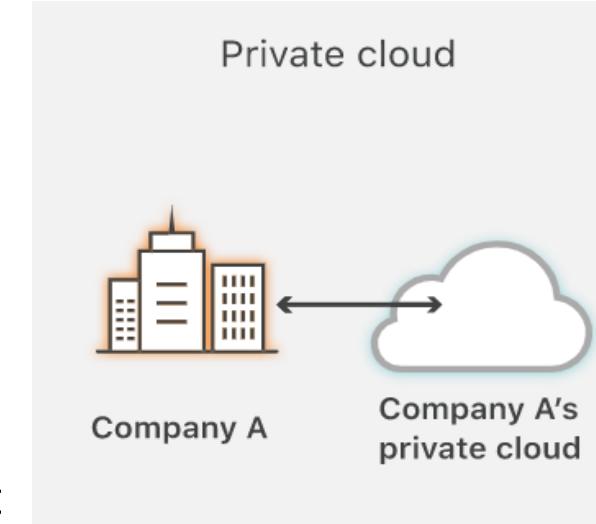
5. Analytics - Public cloud services can perform analytics on high volumes and accommodate a variety of data types to present business insights.

6. Virtually unlimited scalability - Cloud capacity & resources rapidly expand to meet user demands and traffic spikes. Users also achieve greater redundancy and high availability due to the providers' various, logically separated cloud locations.

Disadvantages of Public Clouds

1. **Security** - Verification of the security of data arises as a concern in public clouds, since the data is not being stored by the enterprise. Cloud service providers have attempted to address this problem by acquiring third-party certification.
Multitenancy adds to the security concern due to the probability of data leakage
2. **Compliance** - Many companies might question if the cloud provider is complying with the security rules relating to data. Cloud service providers have attempted to address these issues through certification as well.
3. **Interoperability and vendor lock-in** - Once a particular public cloud has been chosen, it would not be easy to migrate away, since the software and operating procedures would all have been tailored for that particular cloud.
4. Users may be **limited** in terms of the control on the **Infrastructure configurations**

- This is also referred to as an internal or a corporate cloud.
- The private cloud model utilizes the in-house or proprietarily hosted infrastructure to host the different cloud services.
- It enjoys the benefits of cloud computing—including elasticity, scalability, and ease of service delivery—with the access control, security, and resource customization of on-premises infrastructure
- Private cloud is a computing model that offers a proprietary environment dedicated to a single business entity.
- Private cloud could be considered as single tenant from business perspective but could have multiple users using it from within the single business entity and within the intranet cloud
- A private cloud strategy may be comprised of hardware hosted locally at a facility owned by a business, or it may be hosted by a cloud service provider.
- Virtual private clouds are typically paid for on a rolling basis, but provisioned hardware and storage configurations maintain the benefits of a secure, exclusive network.



Internal vs Hosted Private Cloud

Internal Private Cloud



Hosted Private Cloud



Best Private Cloud Providers.

HPE. By most estimates, Hewlett Packard Enterprise (HPE) is a key leader in the private cloud market. ...

VMware. ...

Dell. ...

Oracle. ...

IBM / Red Hat. ...

Microsoft. ...

Cisco.

Advantages/Motivations for a Private Cloud

What motivates organizations to use private cloud?

- 1. More Control** - Private clouds have more control over their resources and hardware than public clouds because it is only accessed by a single organization.
- 2. Security** - Provides the enhanced security of dedicated, physically isolated network, compute and storage.
- 3. Compliance** - For businesses operating in heavily regulated industries, this provides organizations the ability to comply with strict regulations because sensitive data is held on hardware that cannot be accessed by anyone else, thus enabling for necessary policies to support compliance.
- 4. Customization** - Private clouds are fully configurable by the organization, thus by modelling the environment as a Private Cloud enables designing the infrastructure that is best suited for the organizations needs for both Agility and performance.

Hosted private clouds offer the same advantages but require no on-site setup.

Disadvantages of Private Clouds

1. **Cost** - With exclusivity comes increased cost. If you plan to build your own private cloud, you face a large capital outlay.
2. **Under-utilization** - With a private cloud, the cost of capacity underutilization is a cost to you, not to your provider. Therefore managing and maximizing utilization becomes your concern.
3. **Platform scaling** - Large upward changes in your requirements are likely to require scaling of the physical infrastructure. This is fine but may take longer than simply scaling a virtual machine within existing capacity.

What factors influence whether an organization use a private cloud or public cloud?

1. Infrastructure -

- The private cloud model utilizes the in-house infrastructure to host the different cloud services. The cloud user here typically owns the infrastructure. Scaling of the infrastructure is inflexible and incurs capital cost
- The infrastructure for the public cloud on the other hand, is owned by the cloud vendor. The cloud user pays the cloud vendor for using the infrastructure. Public cloud is much more amenable to provide elasticity and scaling-on-demand since the resources are shared among multiple users

2. Network bandwidth constraints and cost - Disruptions in the connectivity between the client and the cloud service will affect the availability of cloud-hosted applications. On a low bandwidth network, the user experience for an interactive application may also get affected. If the storage is intended to be used for a longer term, then it may be more cost-effective to buy storage and compute and use it as a private cloud. Thus, it can be seen that one of the factors dictating the use of a private cloud or a public cloud for storage is how long the resources are needed.

3. Control and Security - Some businesses may prefer to use a private cloud, especially if they have extremely high security standards. Using a private cloud eliminates intercompany multitenancy (there will still be multitenancy among internal teams) and gives a business more control over the cloud security measures that are put in place.

Public Cloud vs Private Cloud

Public Cloud	Private Cloud
Cloud Computing infrastructure shared to public by service provider over the internet. It supports multiple customers i.e, enterprises.	Cloud Computing infrastructure shared to private organisation by service provider over the internet. It supports one enterprise.
Multi-Tenancy i.e, Data of many enterprise are stored in shared environment but are isolated. Data is shared as per rule, permission and security.	Single Tenancy i.e, Data of single enterprise is stored.
Cloud service provider provides all the possible services and hardware as the user-base is world. Different people and organization may need different services and hardware. Services provided must be versatile.	Specific hardware and hardware as per need of enterprise are available in private cloud.
It is hosted at Service Provider site.	It is hosted at Service Provider site or enterprise.
It is connected to the public internet.	It only supports connectivity over the private network.
Scalability is very high, and reliability is moderate.	Scalability is limited, and reliability is very high.
Cloud service provider manages cloud and customers use them.	Managed and used by single enterprise.
It is cheaper than private cloud.	It is costlier than public cloud.
Security matters and dependent on service provider.	It gives high class of security.
Performance is low to medium.	Performance is high.
It has shared servers.	It has dedicated servers.
Example : Amazon web service (AWS) and Google AppEngine etc.	Example : Microsoft KVM, HP, Red Hat & VMWare etc.

Public Cloud vs Private Cloud

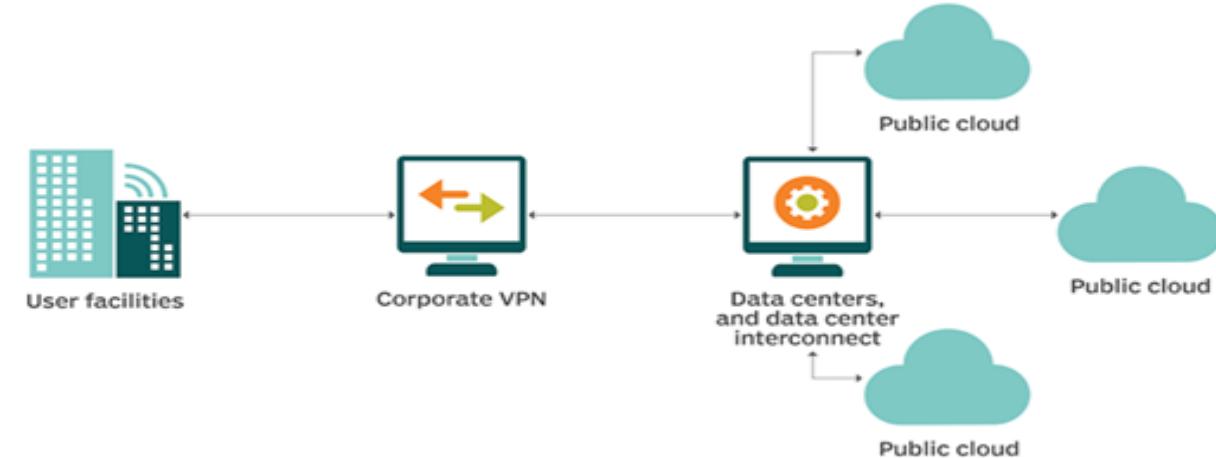
Best Suited for

Public Clouds provide **affordable solutions that offer room for growth**. It is thus ideal for application testing and cloud disaster recovery for small scale companies.

Private Clouds are for **high-performance security and customizable options**. It is, therefore, suitable for protecting sensitive data and applications.

Hybrid Cloud

- A hybrid cloud is a cloud computing environment that uses a mix of on-premises, private cloud and third-party, public cloud services with orchestration between these platforms.
- It combines these Public and Private clouds to create a unified, automated, and well-managed computing environment.
- A hybrid cloud model allows enterprises to deploy workloads in private IT environments or public clouds and move between them as computing needs and costs change.
- Typically non-critical activities are performed by the public cloud and critical activities are performed by the private cloud. This gives business'es greater flexibility and more data deployment options.
- A hybrid cloud workload includes the network, hosting and web service features of an application.



Public Cloud vs Private Cloud

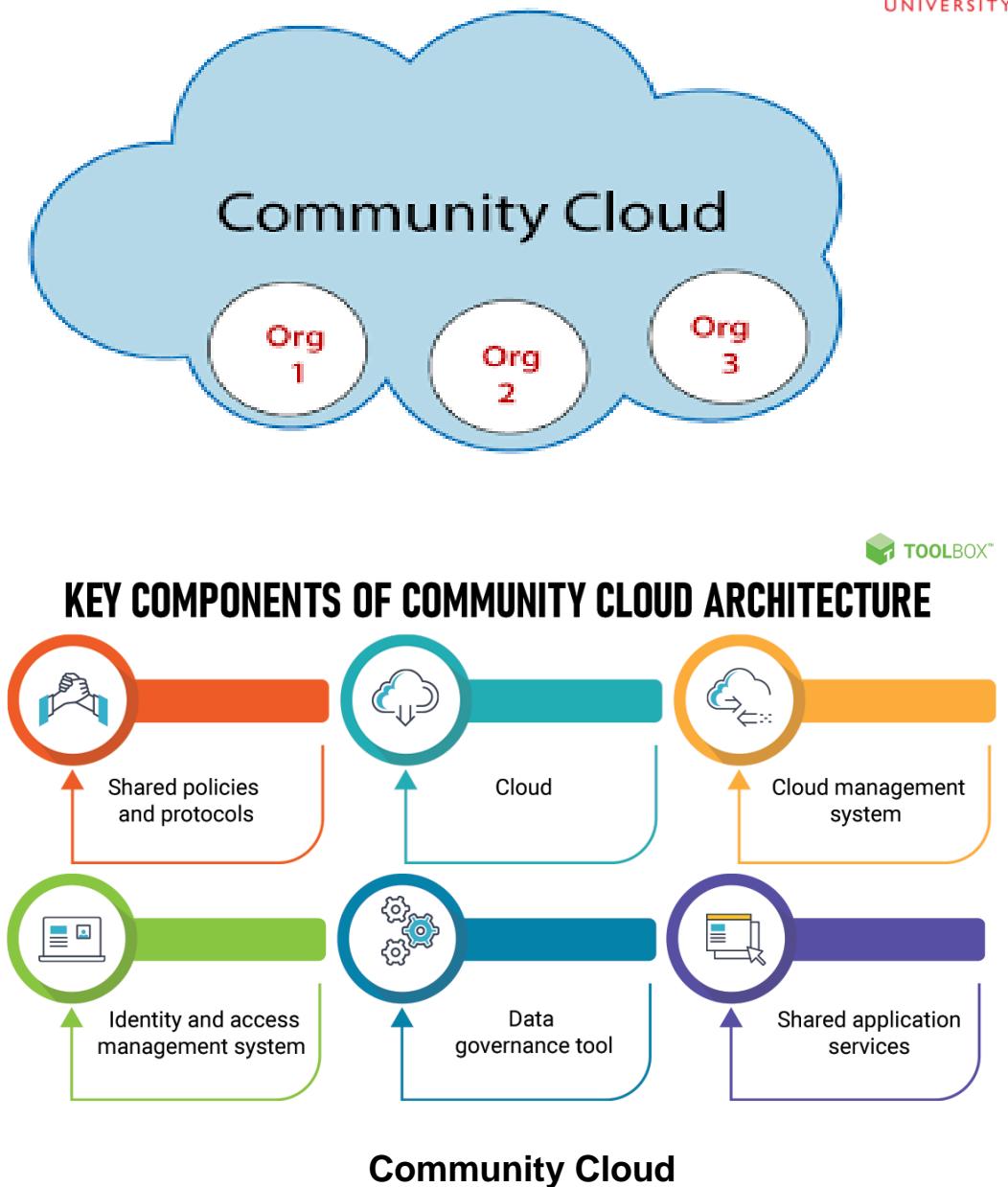
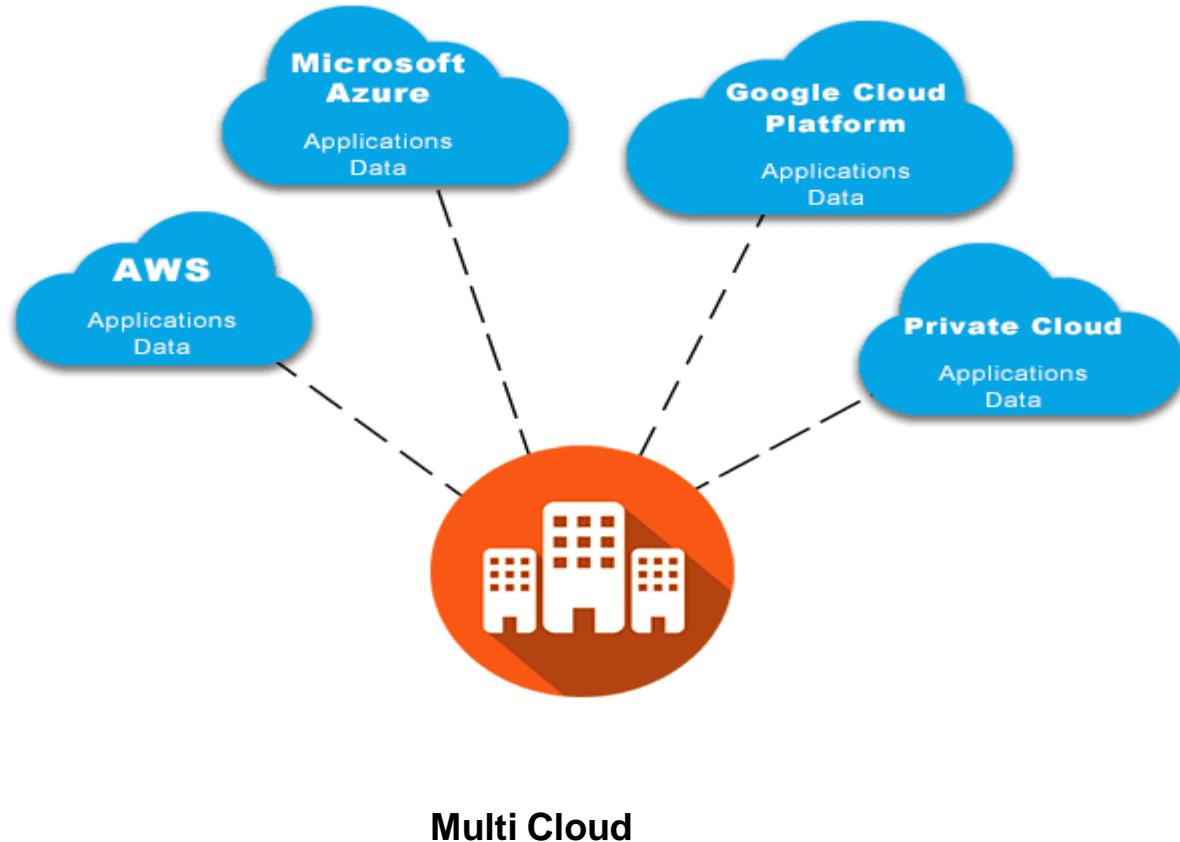
Public Cloud	Private Cloud	Hybrid Cloud
No maintenance costs	Dedicated, secure	Policy-driven deployment
High scalability, flexibility	Regulation compliant	High scalability, flexibility
Reduced complexity	Customizable	Minimal security risks
Flexible pricing	High scalability	Workload diversity supports high reliability
Agile for innovation	Efficient	Improved security
Potential for high TCO	Expensive with high TCO	Potential for high TCO
Decreased security and availability	Minimal mobile access	Compatibility and integration
Minimal control	Limiting infrastructure	Added complexity

Benefits

Drawbacks

CLOUD COMPUTING

Community Cloud & Multi Cloud





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Distributed System Models

Dr. H.L. Phalachandra

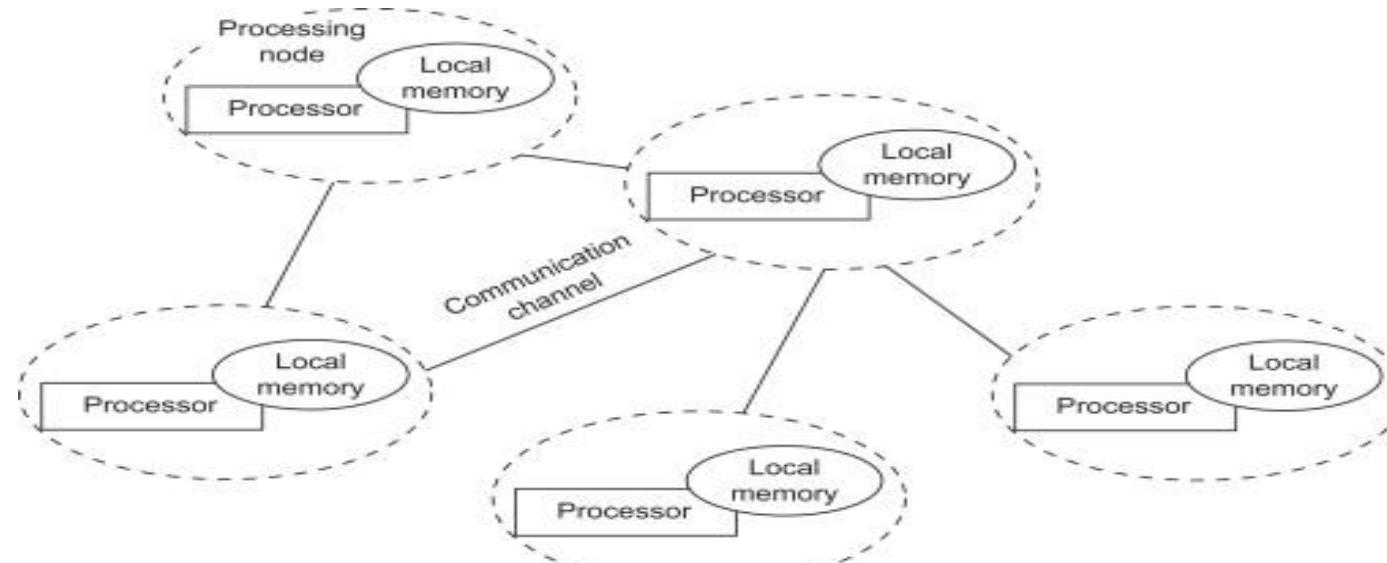
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

- A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network.
- Information exchange in a distributed system is accomplished through message passing.
- Distributed and cloud computing systems are built over a large number of autonomous computer nodes.
- These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner.

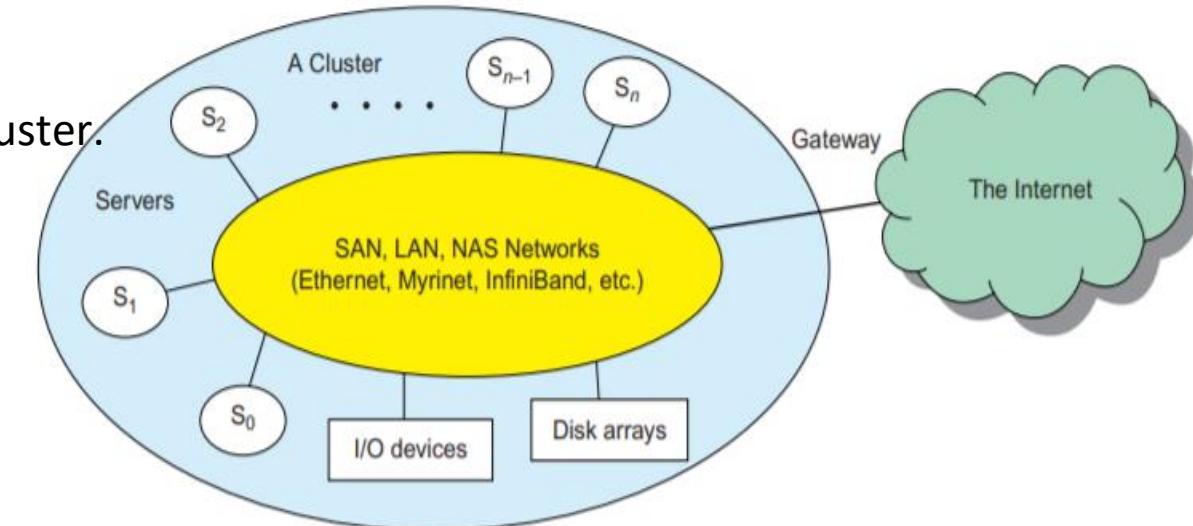
Eg. Scalar Multiplication of a matrix



What is Cluster?

Cluster as a low-latency, high-bandwidth interconnected network of standalone computers which work cooperatively as a single integrated computing resource

- A typical cluster architecture of distributed systems would show a number of hierarchically organized set of compute nodes connected by SAN/LAN or WAN as in the figure below.
- All nodes in a cluster are set to perform the same task with the resources of the node being managed by their own OS
- This cluster supports Scaling by increasing the number of nodes in the hierarchical organization of the systems
- The cluster is connected to the Internet via a virtual private network (VPN) gateway which provides secure connectivity using an encryption.
- The VPN gateway IP address helps to identify and locate the cluster.
- Most clusters will have multiple system images as a result of having many autonomous nodes under different OS control.
- Clustering could be with the focus of improving performance, supporting availability and error handling etc.



- In Cloud computing, resource requests are provisioned into system components which are configured and available as a distributed system
- Depending on the needs of the application, the system components which are part of the cloud infrastructure can be structured and classified as three different Distributed System Models

1. Architectural Models

- a. System Architecture
 - This indicates how the components of a distributed system are placed across multiple machines
 - How the responsibilities are distributed across system components
 - E.g. P2P Model or Client Server Model
- b. Software architecture
 - This indicates the logical organization of software components and their interactions/independence
 - Focusses about the components
 - E.g. 3 Tier Architecture

2. Interaction Models

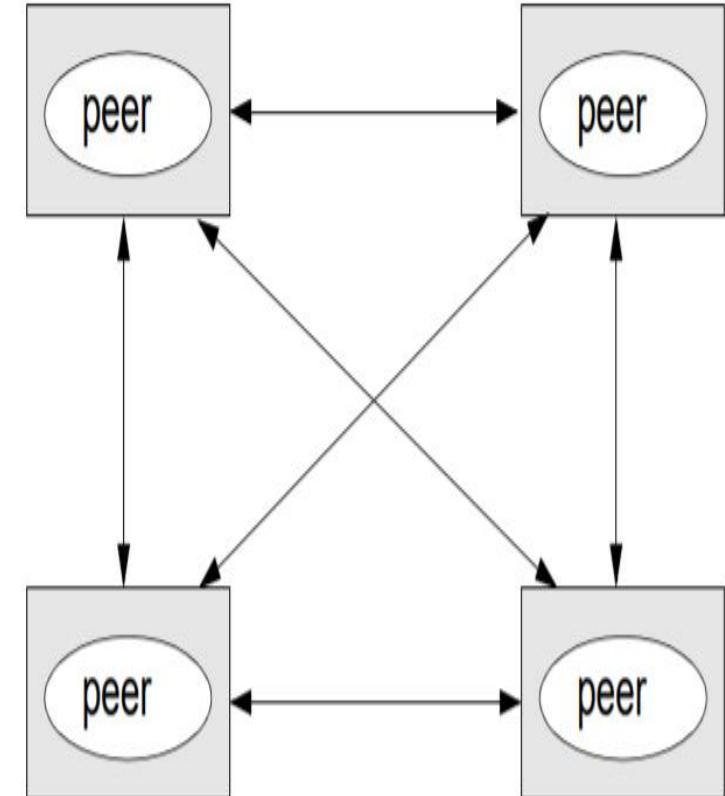
- a. How do we handle time? Are there time limits on process execution, message delivery, and clock drifts?
- b. Ex: Synchronous distributed systems, Asynchronous distributed systems

3. Fault Models

- - a. What kind of faults can occur and what are their effects?
 - b. Ex: Omission faults, Arbitrary faults, Timing faults

What does a Peer-to-Peer System mean?

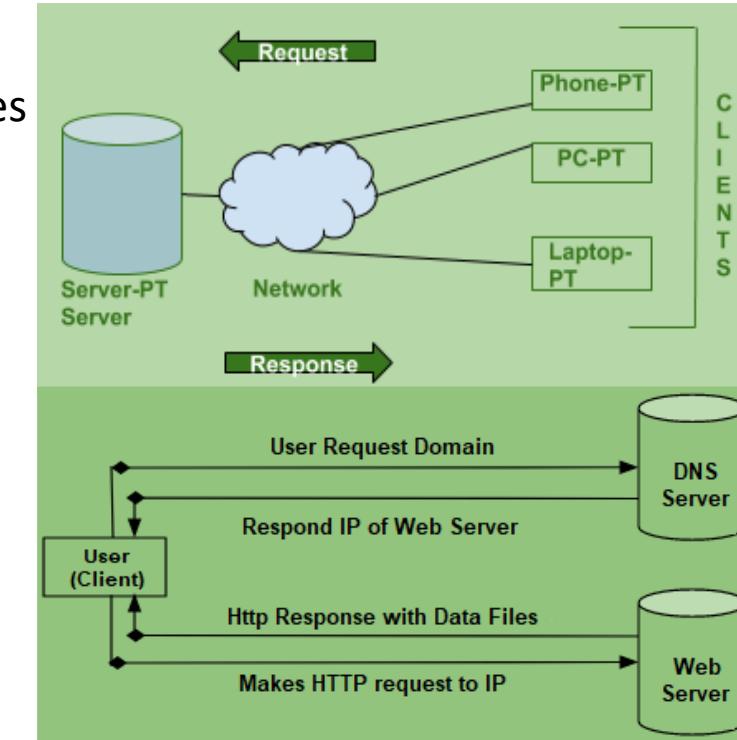
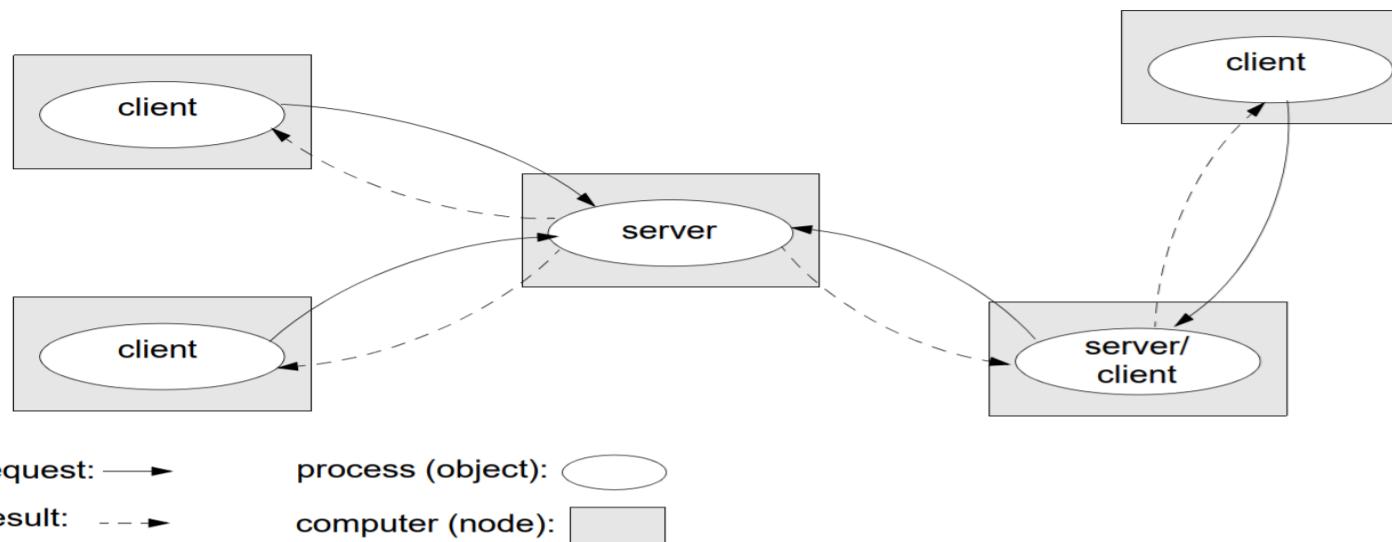
- In a P2P network, every node (peer) acts as both a **client and server**.
- Peers act autonomously to join or leave the network.
- No central coordination or central database is needed.
- This implies that no master-slave relationship exists among the peers.
- The system is self-organizing nodes (recognize and create relatively stable connections to other nodes with similar interests/requirements/capabilities) with distributed control. In other words, no peer machine has a global view of the entire P2P system
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model but securing the overall system would be more challenging as each of the peer would have their own data



What is the Client-Server model?

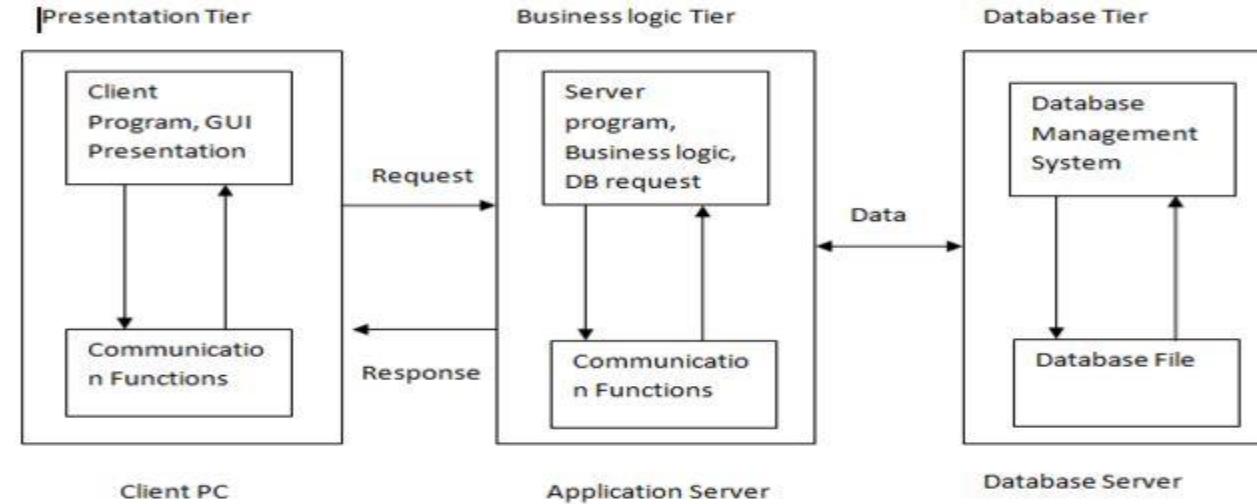
The system is structured where a set of machines called **servers** (which are performing some process), offer services to another set of machines called **clients** for their needs.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC).
- The client asks the server for a service, the server does the work and returns a result or an error code if the required work could not be done
- This organization by its structure distributes the functionality across different machines



Here are some other architecture models that you should know about:

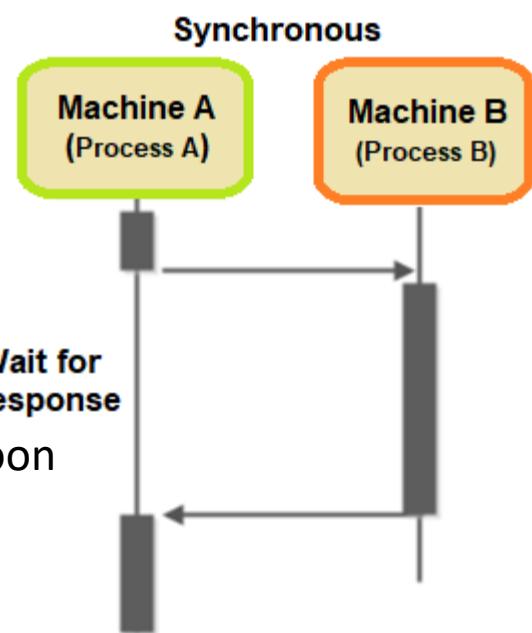
1. **Three-tier** - Architectures that move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.



2. **n-tier** - Architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

What are some features of a Synchronous Distributed System?

1. Systems within the Synchronous distributed system have a shared clock (same clock or different synchronized clocks, clocks with known offsets/bounds etc.)
2. Lower and upper bounds on execution time of systems/processes can be set
3. Transmitted messages to be received within a known bounded time.
4. Ordered message delivery or the network will deliver messages in the sent order
5. Lock step execution – all the nodes which are processing identical message would do so as soon as received and generate the output at the same time.
6. Allows us to make assumptions about time and order of events in a distributed system.
7. Not very practical



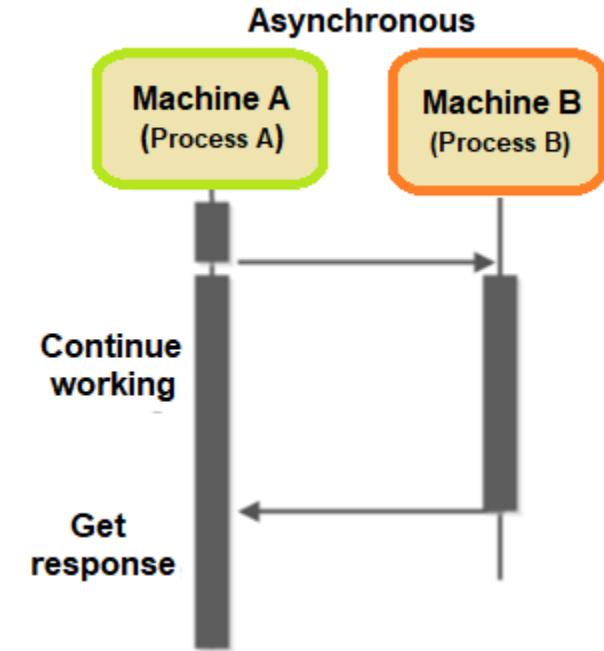
What are the consequences of having a Synchronous Distributed System?

1. Needs a global physical time
2. Needs predictability in terms of timing, as only such systems can be used for hard real-time applications.
3. It is possible and safe to use timeouts in order to detect failures of a process or communication link. (??)

NOTE: Clock drift refers to several related phenomena where a clock does not run at exactly the same rate as a reference clock (NTP). That is, after some time the clock "drifts apart" or gradually desynchronizes from the other clock.

What are some features of an Asynchronous Distributed System?

1. Clock may not be accurate, and can be out of sync
2. No bound on machine/process execution time (nothing can be assumed about speed, load, reliability of computers)
3. No bound on message transmission delays and can be delayed for arbitrary times
4. No constraints on time and ordering of events.
5. Each computer processes independently of others
6. Most suitable for real world scenarios



What are the consequences of having an Asynchronous Distributed System?

1. There is no global physical time, reasoning can be only in terms of logical time.
2. Unpredictable in terms of timing.
3. Cannot use timeouts to diagnose issues
4. They may use mechanisms like queue for asynchronous communication
5. Systems which are using Asynchronous distributed systems have to build algorithms which tolerate different kinds of failures

A system is said to “fail” when it *cannot meet* its promises. A failure is brought about by the *existence* of “errors” in the system. The *cause* of an error is called a “fault”.

A **failure** of a system is brought about due to an **error** in the system caused by a **fault**.

There can be different kinds of faults

- **Transient Faults** : Appears once, then disappears
- **Intermittent Faults** : Occurs, Vanishes, reappears, but no real pattern (worst form of faults)
- **Permanent Faults** : Once it occurs, only the replacement/repair of the faulty component will allow the Distributed System to function normally

What is the use of a Fault Model?

1. Faults can occur both in processes and communication channels. The reason can be both software and hardware.
2. **Fault models** are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).
3. A fault tolerant system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts.
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests. - A server fails to receive incoming messages. - A server fails to send outgoing messages.
Timing failure	A server's response lies outside the specified time interval.
Response failure <i>Value failure</i> <i>State transition failure</i>	The server's response is incorrect. - The value of the response is wrong. - The server deviates from the correct flow of control.
Arbitrary failure	A server may produce arbitrary responses at arbitrary times.

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a <i>send</i> , but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Process aka Machine

Channel aka Communication Path aka Network Path

<i>Class of Failure</i>	<i>Affects</i>	<i>Description</i>
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

- Applicable in synchronous systems with set time limits
- Not applicable in asynchronous systems since no time limits can be guaranteed.

Process aka Machine

Channel aka Communication Path aka Network Path

References for Distributed System Models

- [Distributed Architecture](#)
- [Distributed Computing Architectures - Wikipedia](#)
- ["Explain Distributed system models with diagram"](#)
- [Make your existing solution tastier with serverless salt: distributed system](#)
- [System Models for Distributed and Cloud Computing - UNF](#)
- [Fundamental Distributed System Models - RIT](#)
- [MODELS OF DISTRIBUTED SYSTEMS - Linköping University](#)
- [Distributed System Models](#)
- [A Brief Summary of Apache Hadoop: A Solution of Big Data Problem and Hint comes from Google](#)



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Cloud Architecture

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

CLOUD COMPUTING

Cloud Architecture

What is cloud architecture?

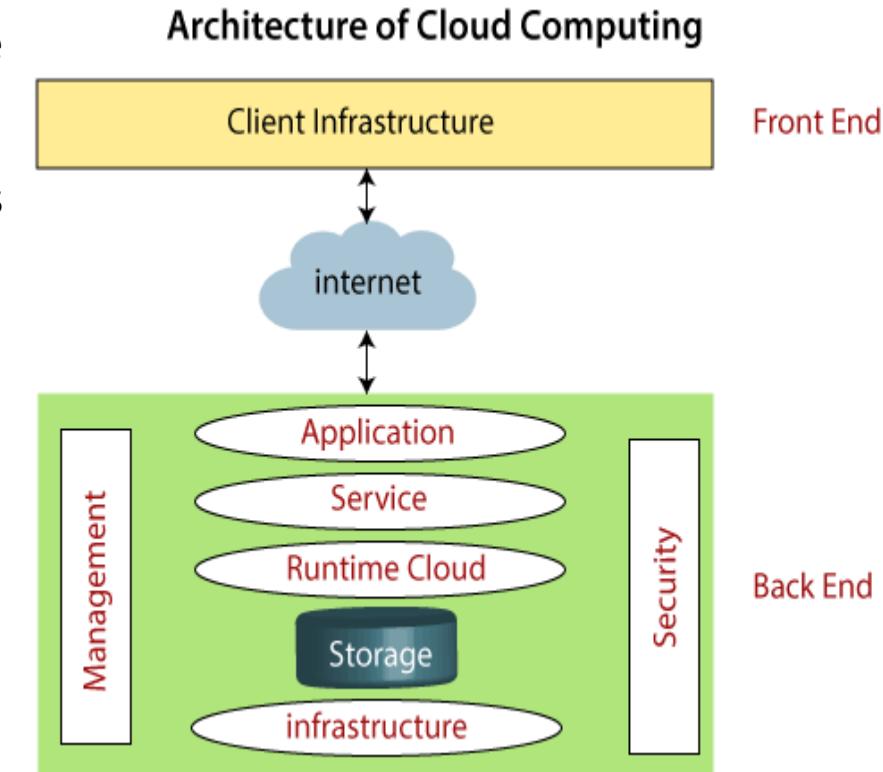
Cloud architecture refers to the way in which technology components which makeup the cloud environment are engineered or combined to leverage the power of cloud resources to solve business problems.

Cloud architecture defines the components as well as the relationships between them.

The components of a cloud architecture include:

- A. A **Front-end** : Client-side interfaces and applications that are required to access the cloud computing platforms. (Web clients, thin or fat clients, tablets, mobile devices)
- B. A **back-end** platform (servers and storage)
- C. A **network** (Internet, Intranet, Intercloud).

Together, these components create a cloud computing architecture on which applications can run, providing end-users with the ability to leverage the power of cloud resources.



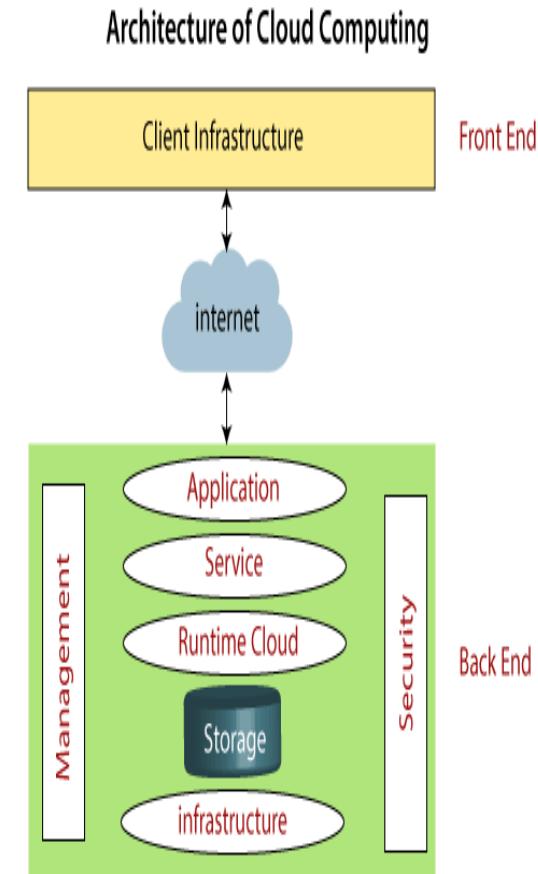
Cloud Architecture – In a little more detail

A. What is a front-end?

- This is the front end part of the cloud Architecture or the **Client Infrastructure** to interact with the cloud.
- This could be client-side interfaces and applications that are required to access the cloud computing platforms.
E.g. Web clients (including Chrome, Firefox, internet explorer etc.), thin & fat clients, tablets, and mobile devices.

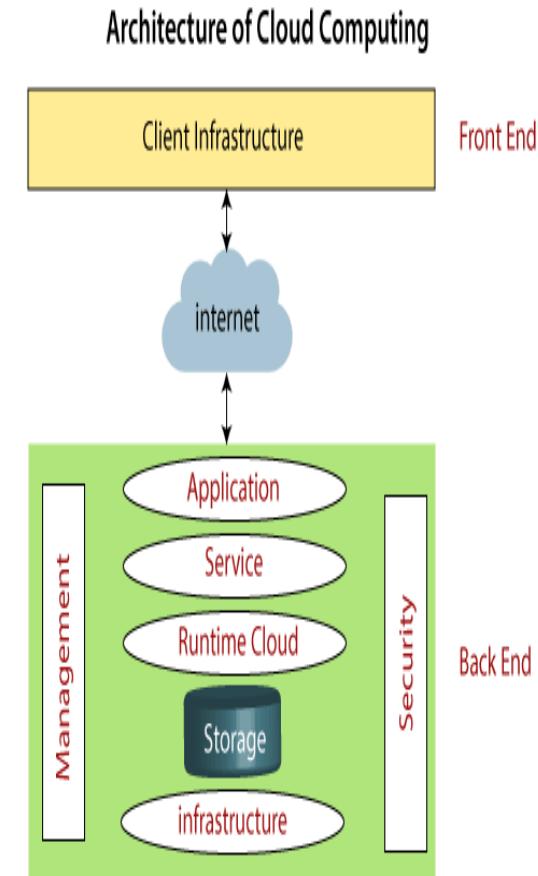
B. What is a back-end?

- The back end part of the cloud architecture is used by the **Service provider**.
- This manages all the resources that are required to provide cloud computing services.
- The detailed set of components involved in the back end are
 1. **Application** - It's the part offered for the Client application which will use the Cloud.
This can either be a software or a platform
 2. **Service** – This is a piece of software, which manages or based on the application needs, will determine and enable the appropriate service to be accessed. This could be an Infrastructure (IaaS) or platform (PaaS) or a software (SaaS) service.



Cloud Architecture – In a little more detail (Cont.)

3. **Runtime Cloud** – Provides the execution and runtime environment to the Virtual Machine dependent on the service model
 4. **Storage** - Storage is one of the most important components of cloud computing. It provides a huge amount of storage capacity in the cloud to store and manage data.
 5. **Infrastructure** - Cloud infrastructure includes hardware and software components such as servers, storage, network devices, virtualization software, and other storage resources that are needed to support the cloud computing model.
 6. **Management** - Components like application, service, etc in the backend need to be managed and coordination between them needs to be established.
 7. **Security** - It implements security mechanisms, for secure cloud resources, systems, files, and infrastructure to end-users.
- C. **Internet** - Internet connection acts as the medium or a bridge between frontend and backend and establishes the interaction and communication between frontend and backend.



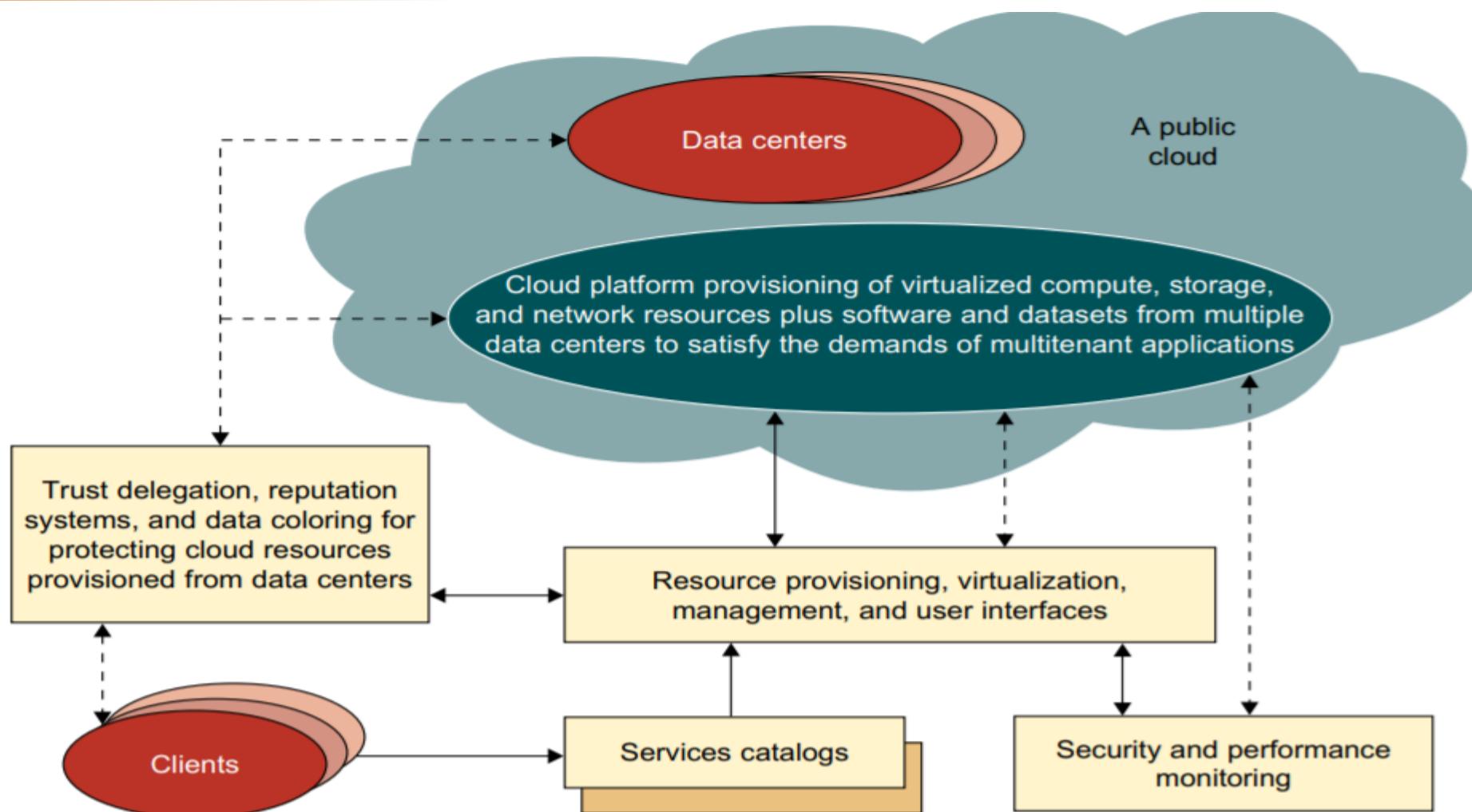
Cloud Platform Design Goals

We have been discussing number of value propositions of using cloud, and these would be the goals targeted for while designing and architecting a Cloud platform. A few of these are

1. Scalability - Up- Out etc
2. Efficiency - Do this quickly with minimal utilization of resources
 - Quickly could also mean bringing up the service which could be across the stack
3. Reliability and Availability
4. Simplifying the User experience

These are typically enabled by architectures like Clustering, Virtualization - Hypervisor, Replication, Elasticity, Simple deployment and scheduling approaches.

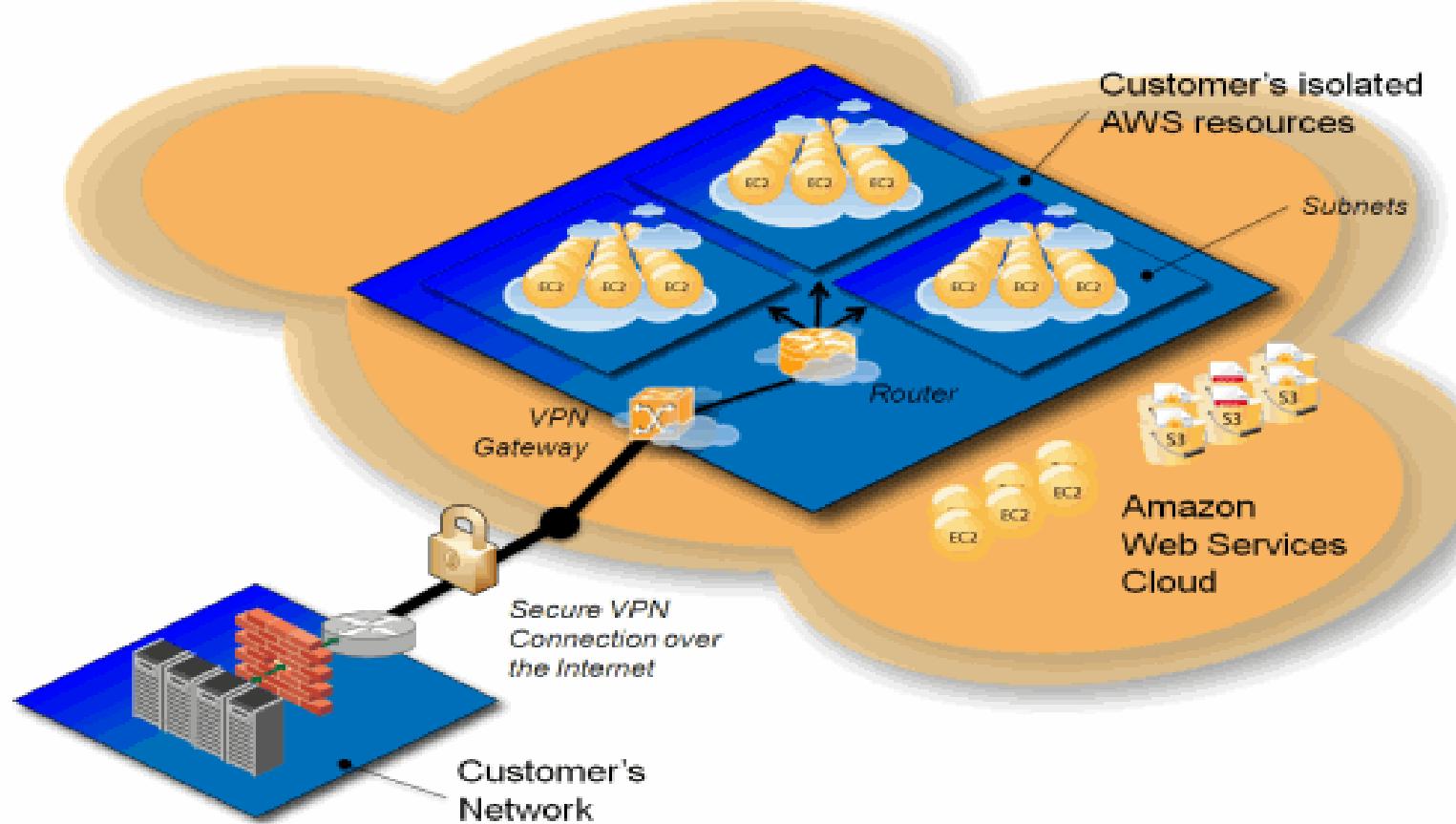
Generic Cloud Architecture Example



Generic cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers which also considers security.

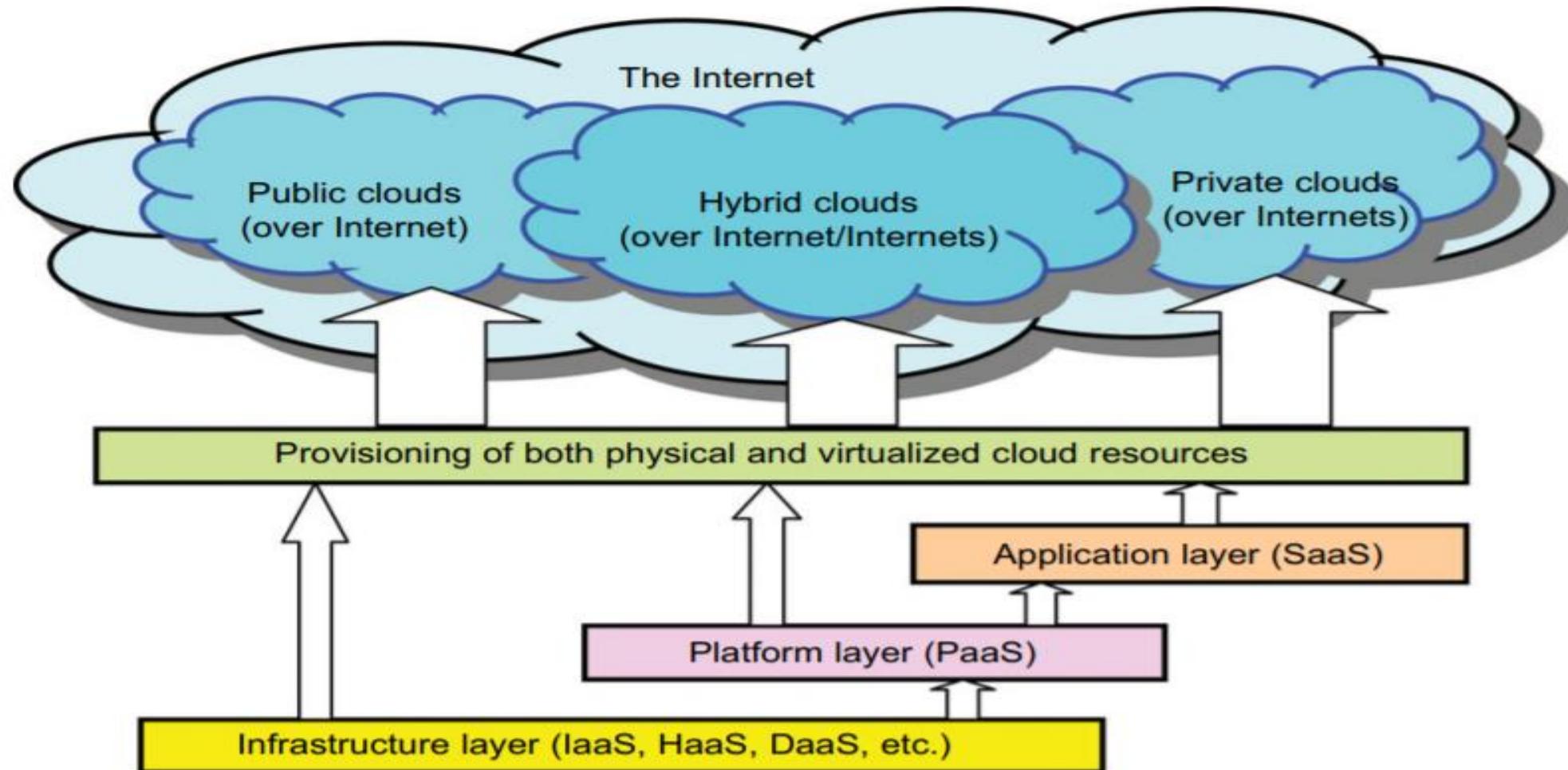
CLOUD COMPUTING

Cloud Architecture : Amazon Example



Other views : Layered Cloud Architecture

The architecture of a cloud can also be visualized into three layers: infrastructure, platform, and application.



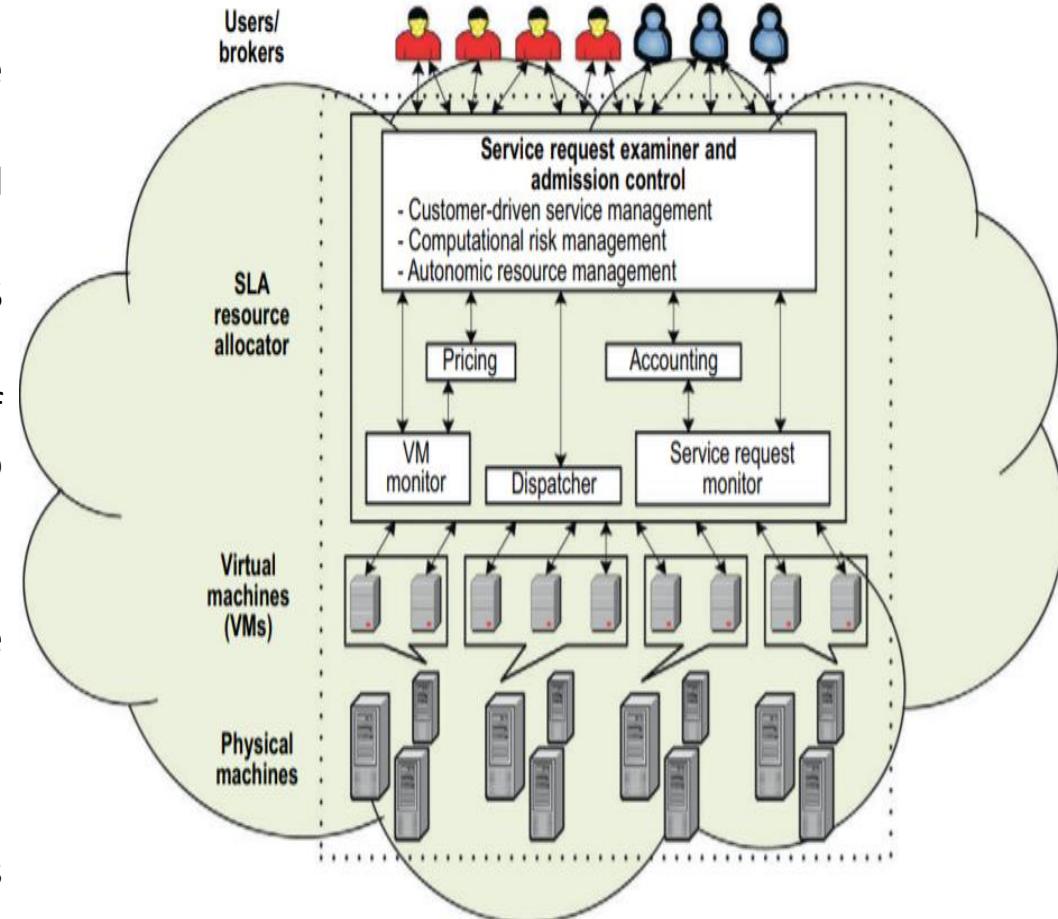
Market-Oriented Cloud Architecture

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations, which will need to be provisioned as part of deployment.

A market-oriented resource management system regulates the supply and demand of cloud resources to achieve an equilibrium between supply and demand. So the resource allocation mechanism will provision based on the QoS and economic incentives are provided proportional to the QoS needed/provided.

The request examiner ensures that there is no overloading or over-provisioning of resources whereby many service requests can be fulfilled successfully due to constraints in the limit of resources.

- The Pricing mechanism decides how service requests are charged. The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements.
- The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs.
- The Service Request Monitor mechanism keeps track of the execution progress of service requests.
- Multiple VMs can be started and stopped on demand on a single physical machine to meet accepted service requests, hence providing maximum flexibility.





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Infrastructure as a Service (IaaS)

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

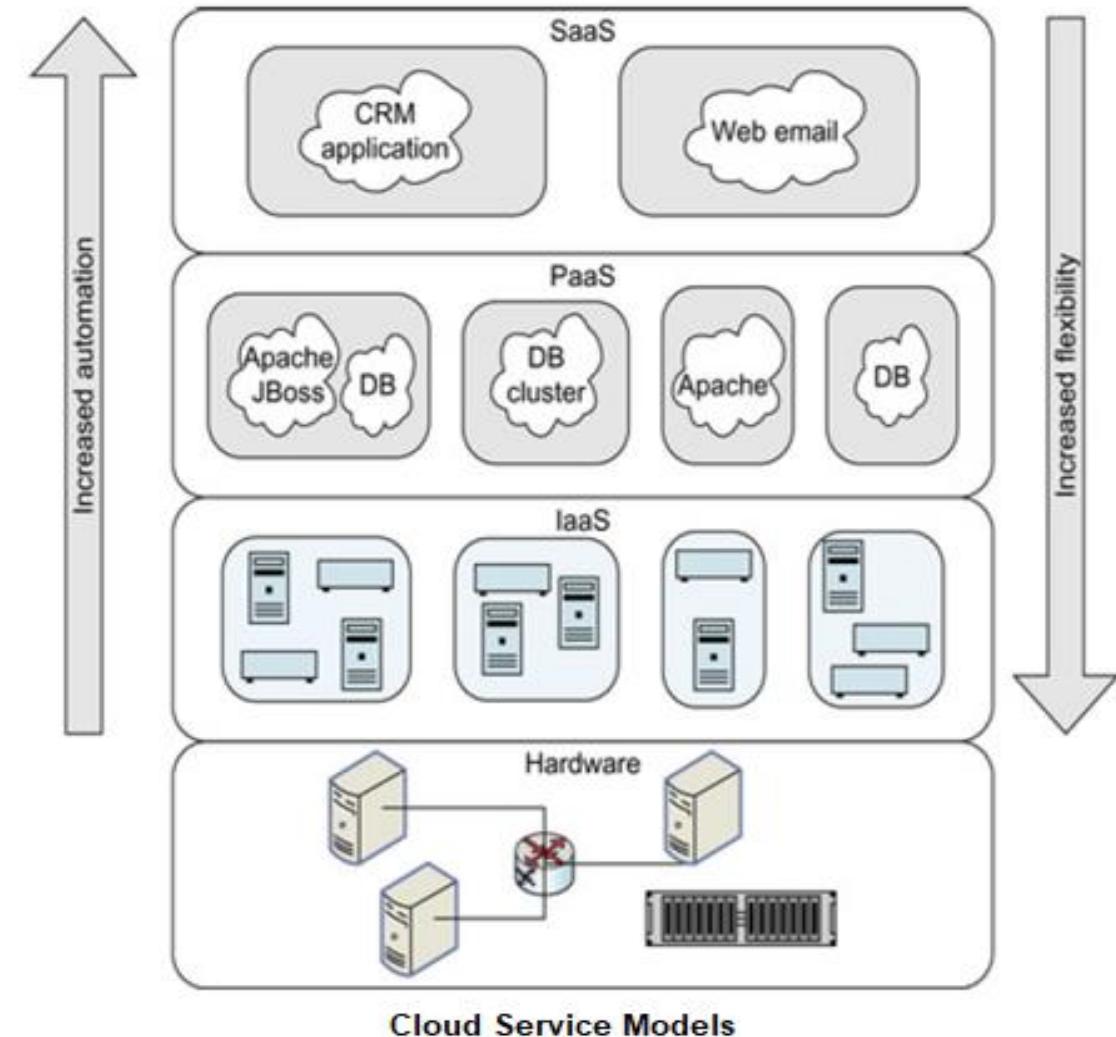
Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Cloud Service Models (Recap)

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers.

The services provided over the cloud have been generally categorized into three different service models:

- Infrastructure as a service
- Platform as a service
- Software as a service



A programming model is an execution model linked into an API or a particular pattern of code. In this, there are actually two execution models in play:

- the **execution model of the base programming language** and
- the **execution model of the programming model.**

A execution model, say of a language like C on a system would be execution of the program one instruction after the other in sequence. The C program statements which are expanded, compiled or converted to assembly, subsequently converted to object code, linked with different object codes to an executable set of instructions, are loaded code into memory and then executed instruction after instruction.

So in the case of a simple program say written in C running on a single system, the programming model would be to execute the code instruction by instruction and use the disk, memory and CPUs locally available on the system.

In cloud environments like IaaS or PaaS or SaaS, the language execution model does not change, but there is an independent execution model of the programming model.

The eco-system for a program execution in terms of the CPU, memory, the persistent memory for storing the programs and data, network ports and IP address which were available locally on your system is now not guaranteed and would be something which needs to be factored in.

Depending on the kind of model, the platforms or frameworks available cannot be assumed to a program as when working on a standalone local node like a physical server or a laptop.

The programming model will need to factor in these and ensure that the requisite environment is setup for each of the different service models of the cloud.

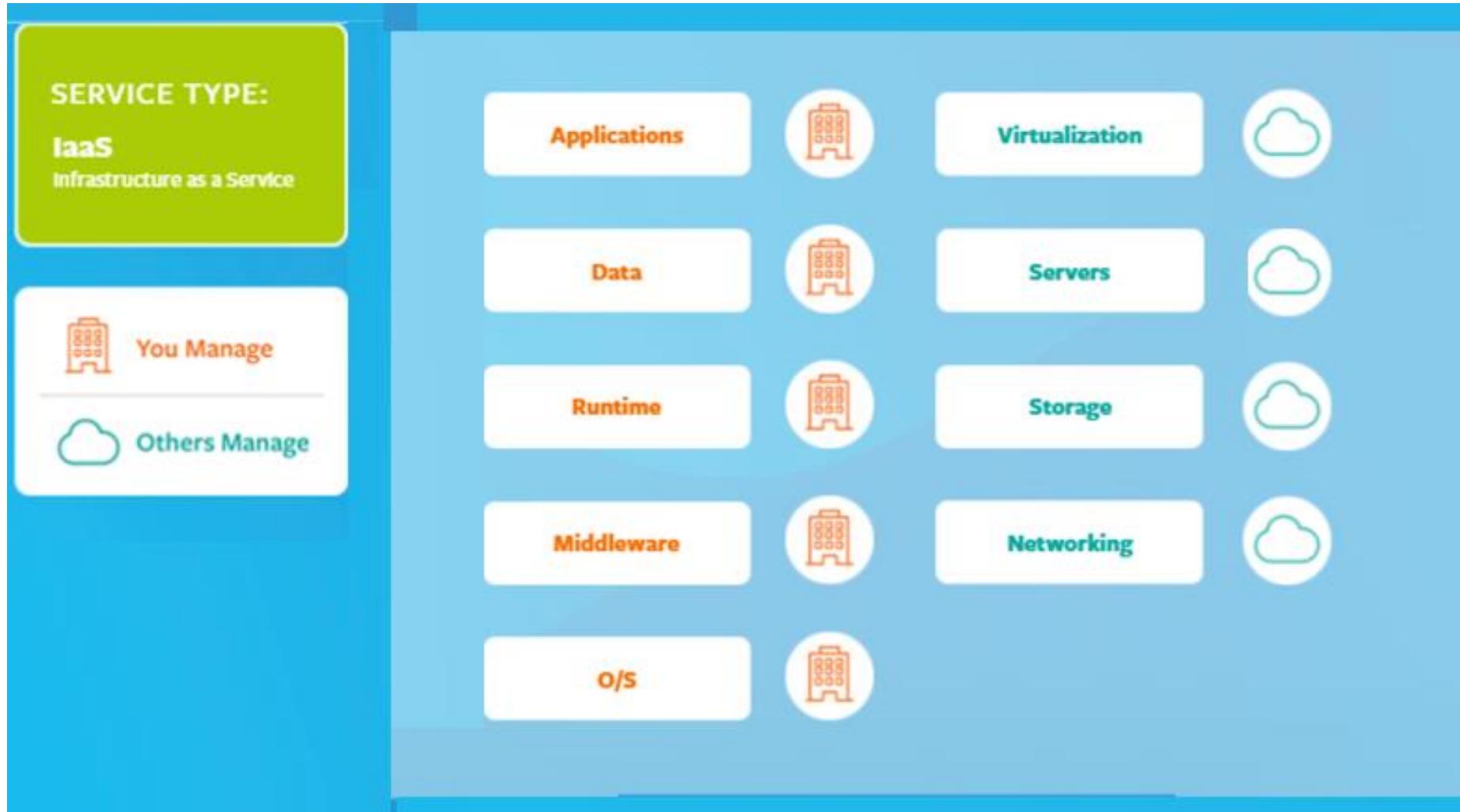
The IaaS model is about providing compute and storage resources as a service.

The capability provided to the consumer is to provision .. processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

The user of IaaS has single ownership of the hardware infrastructure allotted to him. The IaaS provider has control over the actual hardware and the cloud user can request allocation of virtual resources.

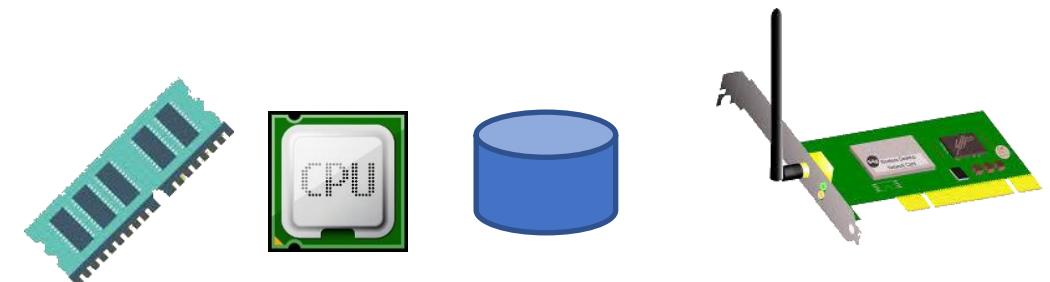
IaaS enables end users to scale and shrink resources on an as-needed basis, reducing the need for high, up-front capital expenditures or unnecessary “owned” infrastructure, especially in the case of “spiky” workloads.



- If you are writing a complex program
 - HelloWorld.exe
- Program needs resources to execute
 - CPU
 - Memory
 - Storage (where the program resides)
 - Possibly network
- Access to resources made available by the OS

HelloWorld.exe

Operating System



Programming model if this needs to run on a Virtual System

- If you are writing a complex program

- Helloworld.exe

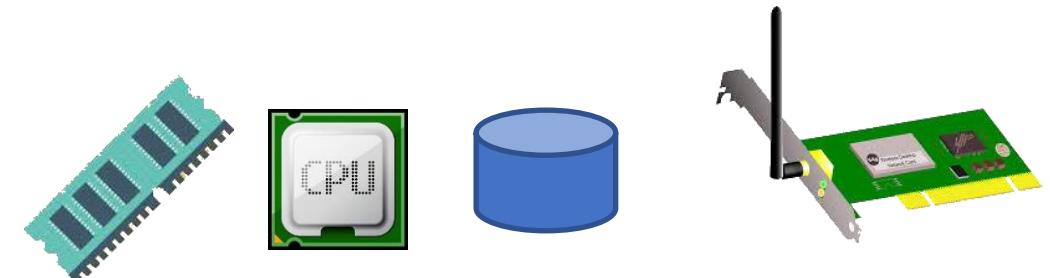
HelloWorld.exe

- Let's assume that this program needs to run on a virtual machine

- How does it get access to

- CPU
 - Memory
 - Storage (where the program resides)
 - Possibly network

???



- Given that your application is running now on a remote server, there will need to be something like a WebServer to accept the application requests

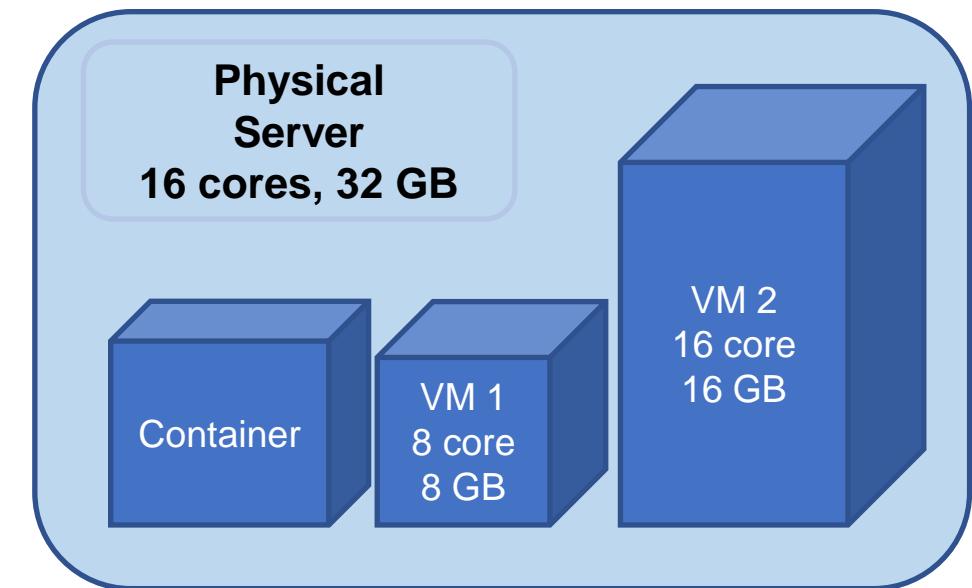
IaaS platform infrastructure - Processing

In IaaS, Infrastructure needs to be provisioned for the applications which will need to be run. IaaS is made up of a collection of physical and virtualized resources that provide consumers with the basic building blocks needed to run applications and workloads in the cloud.

Physical data centers: IaaS providers will manage large data centers, typically around the world, that contain the physical machines with various layers of abstraction on top of them and that are made available to end users over the web.

Compute: Providers provision your compute in the granularity of VMs/Containers on top of the physical Servers using Virtualization techniques which will study in the next unit.

- VMs
 - Each VM "looks like" an actual physical machine to the software
 - Can have its own OS and software
- Container
 - Uses physical server's OS
 - Actually a sandboxed process
 - More lightweight than a VM
- VMs or containers may belong to different users
- Each VM or container is isolated from other VMs
- For now, don't worry about how virtualisation works
 - We will study that in later classes
 - Assume it happens by magic
 - Achieved via the hypervisor



Storage: The three primary types of cloud storage are block storage, file storage, and object storage. An example of how Block storage is shared could be as below.

A way of sharing and pooling physical disks

Sharing:

Pooling: treat disk pool as single large storage say of disks of different capability

E.g., 256 GB virtual disk using 2x128 GB from different physical disks

Can virtual disk be bigger than physical disk?

Each virtual disk can be attached to a VM

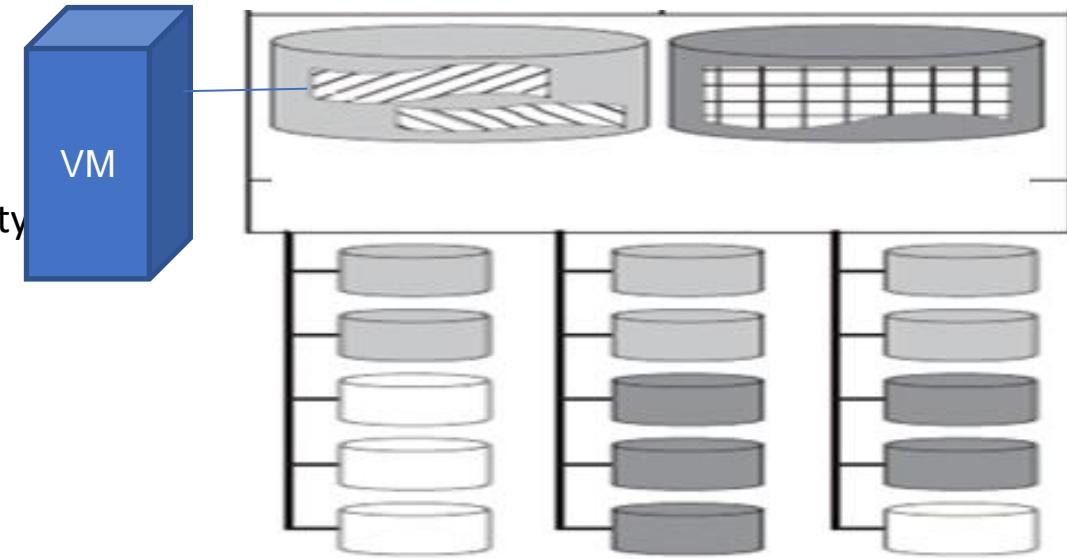
Each virtual disk "looks like" an actual physical disk to the software

Can create files on disk

Can use for paging or swapping

The virtual disks may belong to different users

Each virtual disk is isolated from other virtual disks



The Object storage has thus become the most common mode of storage in the cloud given that it is highly distributed (and thus resilient), it leverages commodity hardware, data can be accessed easily over HTTP, and scale is not only essentially limitless but performance scales linearly as the cluster grows.

We will discuss more on this later.

Network: Networking in the cloud is a form of Software Defined Networking in which traditional networking hardware, such as routers and switches, are made available programmatically, typically through APIs.

- Like physical network, contains virtual adapters and virtual switches
- Each VM has one or more *virtual network adapters* (V)

These have IP addresses, DNS names, just like a physical adapter

This is completely in software

VS is a *virtual network switch*

Completely in software

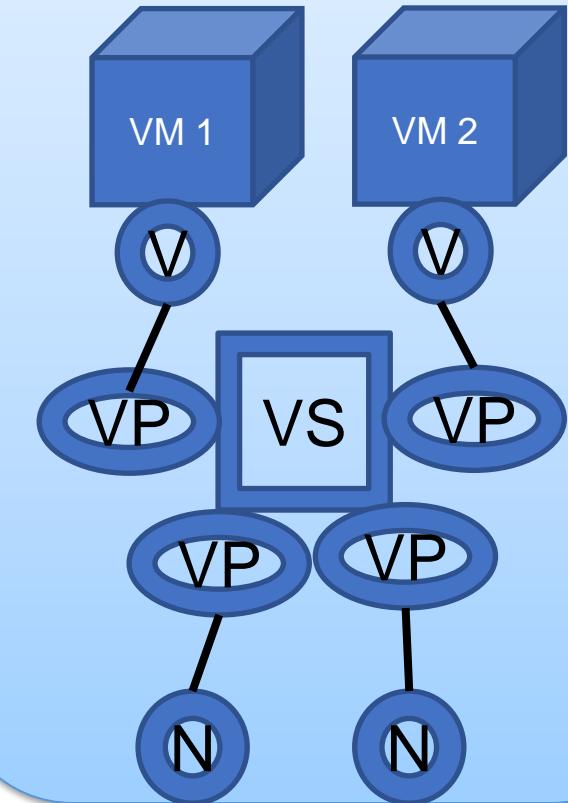
VP is a virtual port

Network adapter, invisible to network

No IP address, only MAC address

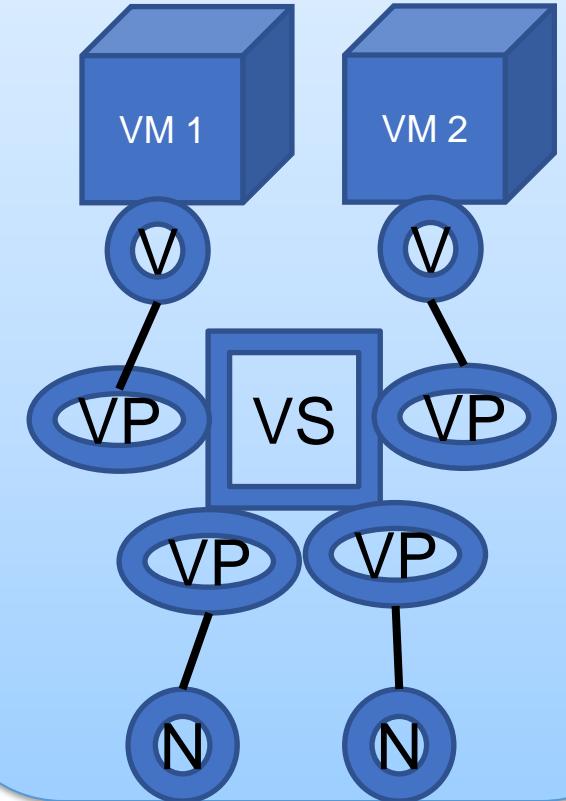
N is the network adapter of the physical server

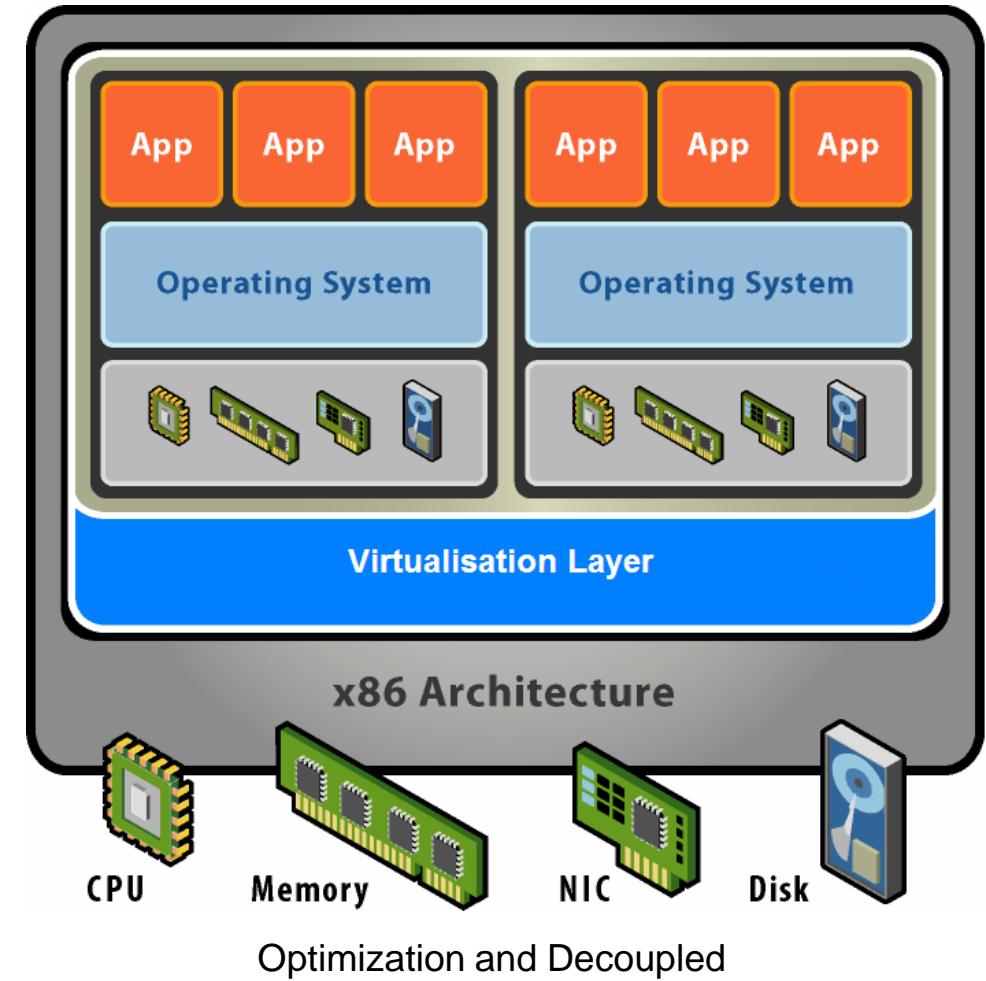
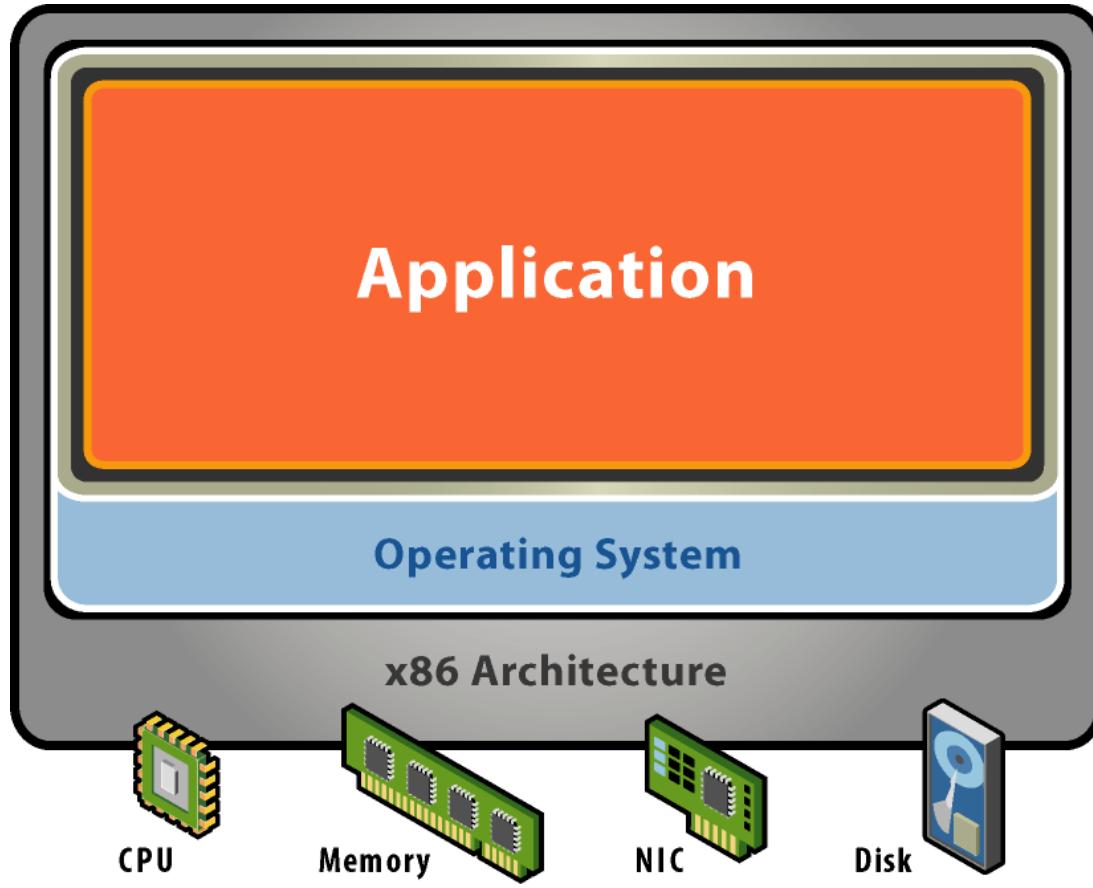
Physical Server



- User specifies
 - VM configuration (e.g., memory)
 - Software
- Cloud software
 - Create actual VM
 - Create virtual disk(s) needed
 - Copy OS and app binaries to boot disk
 - Boot OS
 - Set up virtual network

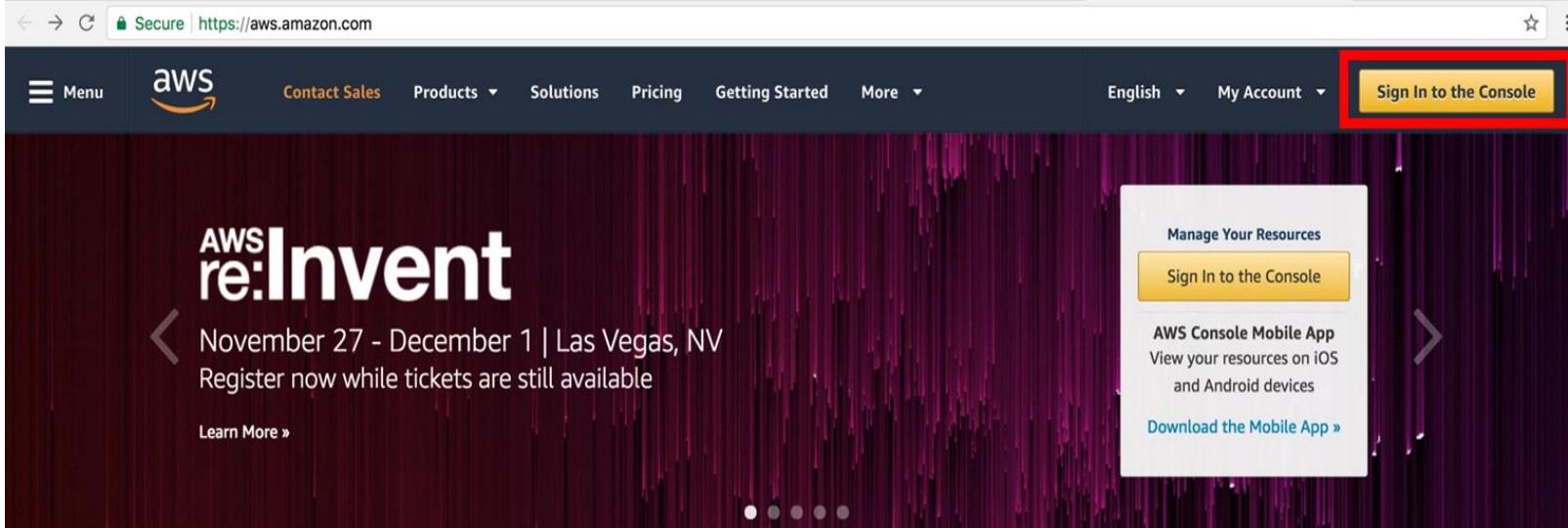
Physical Server





Demo: IaaS on AWS (Compute as a service)

- Step 1: Create an AWS Account and Sign into AWS.

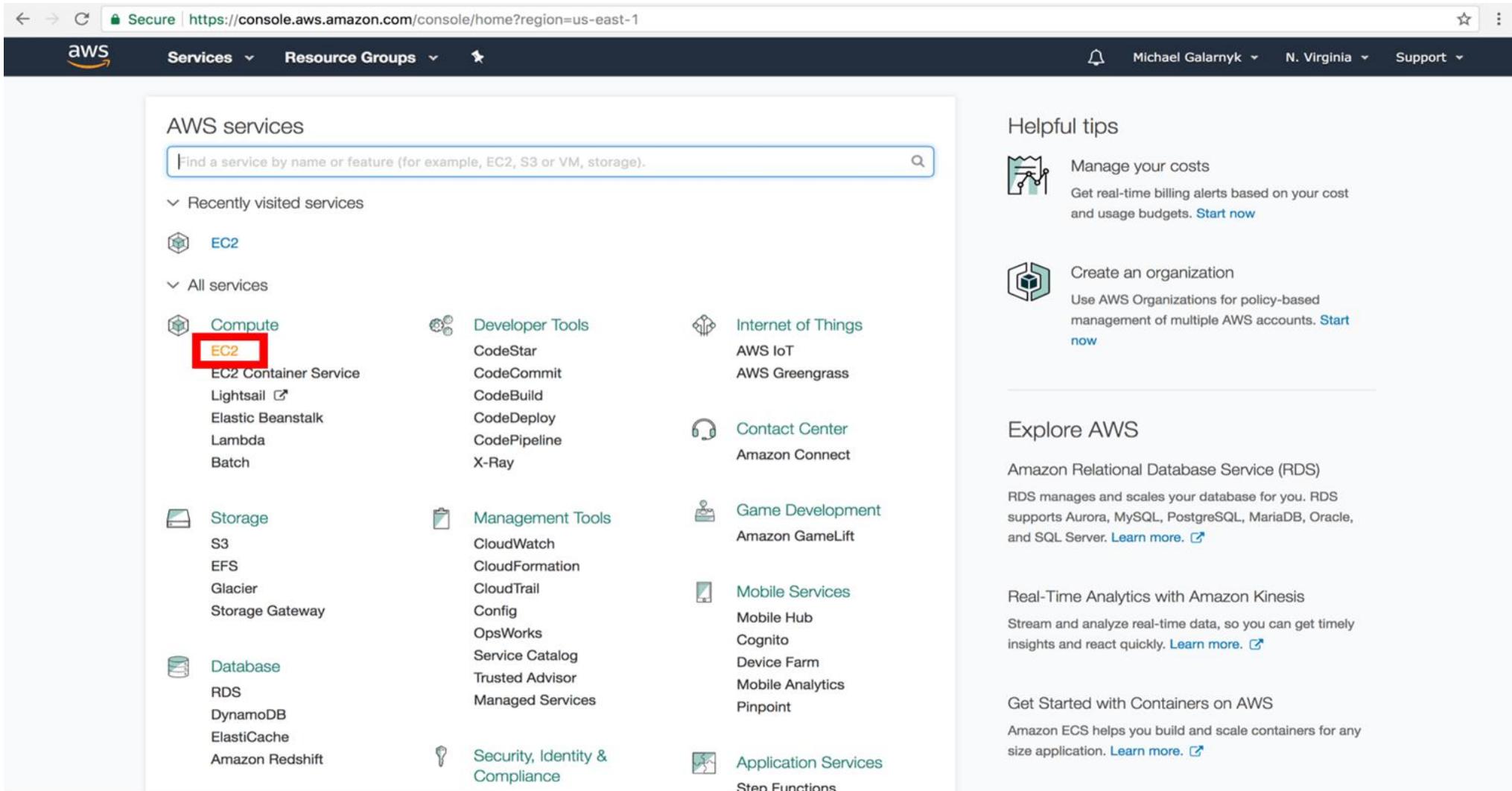


<https://www.datacamp.com/community/tutorials/aws-ec2-beginner-tutorial>

CLOUD COMPUTING

Demo: IaaS on AWS (Compute as a service)

- Step 2: On the EC2 Dashboard, click on EC2..



The screenshot shows the AWS Cloud Services Dashboard. The top navigation bar includes the AWS logo, a 'Services' dropdown, 'Resource Groups' dropdown, user profile 'Michael Galarnyk', location 'N. Virginia', and 'Support'. The main area is titled 'AWS services' with a search bar. Under 'Recently visited services', 'EC2' is listed. Under 'All services', 'Compute' is expanded, showing 'EC2' (which has a red box around it), 'EC2 Container Service', 'Lightsail', 'Elastic Beanstalk', 'Lambda', 'Batch', 'Developer Tools' (listing 'CodeStar', 'CodeCommit', 'CodeBuild', 'CodeDeploy', 'CodePipeline', 'X-Ray'), 'Internet of Things' (listing 'AWS IoT', 'AWS Greengrass'), 'Contact Center' (listing 'Amazon Connect'), 'Management Tools' (listing 'CloudWatch', 'CloudFormation', 'CloudTrail', 'Config', 'OpsWorks', 'Service Catalog', 'Trusted Advisor', 'Managed Services'), 'Game Development' (listing 'Amazon GameLift'), 'Mobile Services' (listing 'Mobile Hub', 'Cognito', 'Device Farm', 'Mobile Analytics', 'Pinpoint'), 'Security, Identity & Compliance', and 'Application Services' (listing 'Step Functions'). To the right, there's a 'Helpful tips' section with 'Manage your costs' (with a graph icon) and 'Create an organization' (with a cube icon). Below that is an 'Explore AWS' section with links to 'Amazon Relational Database Service (RDS)', 'Real-Time Analytics with Amazon Kinesis', and 'Get Started with Containers on AWS'.

AWS services

Find a service by name or feature (for example, EC2, S3 or VM, storage).

Recently visited services

EC2

All services

Compute

EC2

EC2 Container Service

Lightsail

Elastic Beanstalk

Lambda

Batch

Developer Tools

CodeStar

CodeCommit

CodeBuild

CodeDeploy

CodePipeline

X-Ray

Internet of Things

AWS IoT

AWS Greengrass

Contact Center

Amazon Connect

Management Tools

CloudWatch

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

Trusted Advisor

Managed Services

Game Development

Amazon GameLift

Mobile Services

Mobile Hub

Cognito

Device Farm

Mobile Analytics

Pinpoint

Security, Identity & Compliance

Application Services

Step Functions

Helpful tips

Manage your costs

Get real-time billing alerts based on your cost and usage budgets. [Start now](#)

Create an organization

Use AWS Organizations for policy-based management of multiple AWS accounts. [Start now](#)

Explore AWS

Amazon Relational Database Service (RDS)

RDS manages and scales your database for you. RDS supports Aurora, MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. [Learn more](#).

Real-Time Analytics with Amazon Kinesis

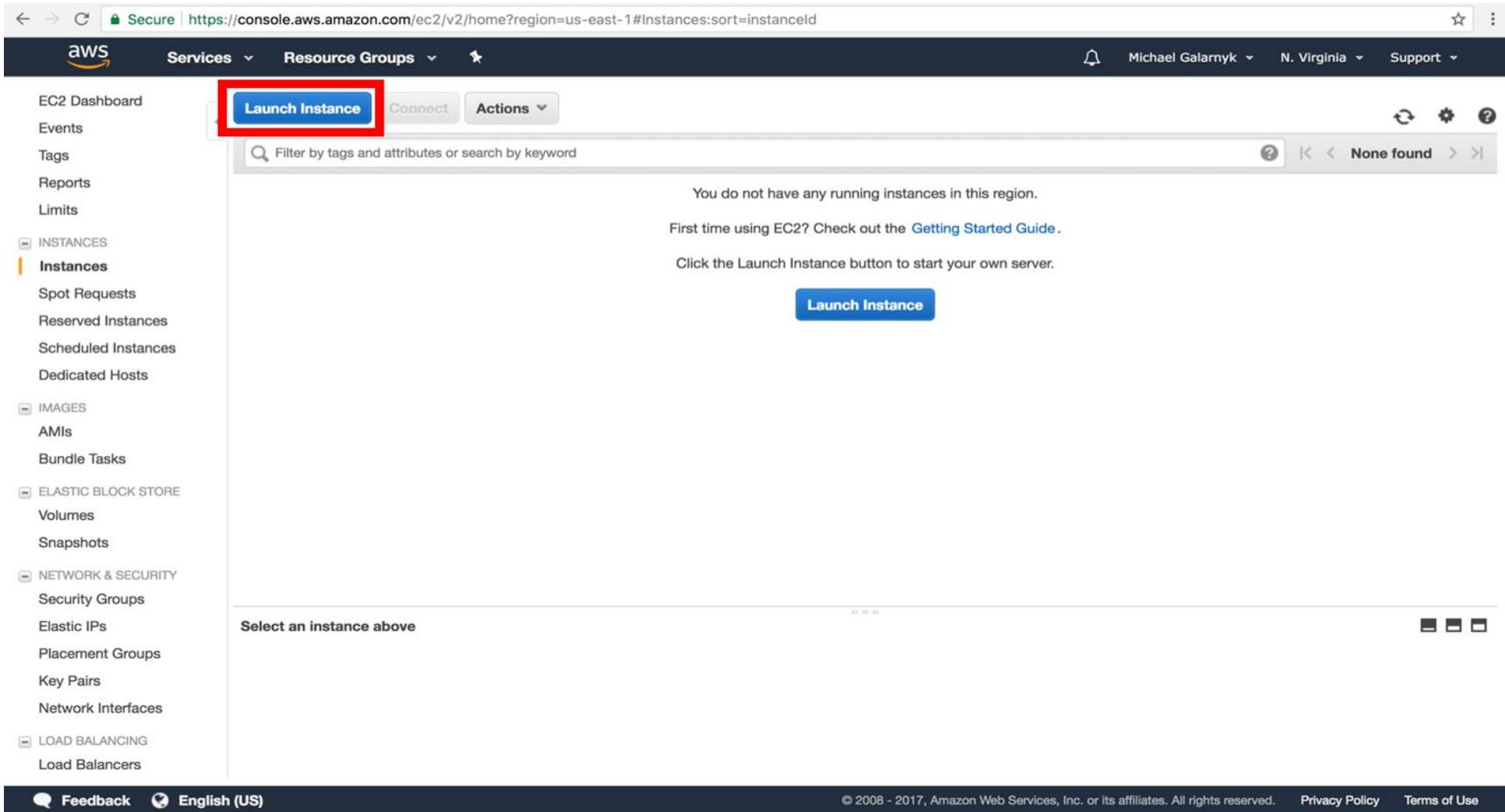
Stream and analyze real-time data, so you can get timely insights and react quickly. [Learn more](#).

Get Started with Containers on AWS

Amazon ECS helps you build and scale containers for any size application. [Learn more](#).

Demo: IaaS on AWS (Compute as a service)

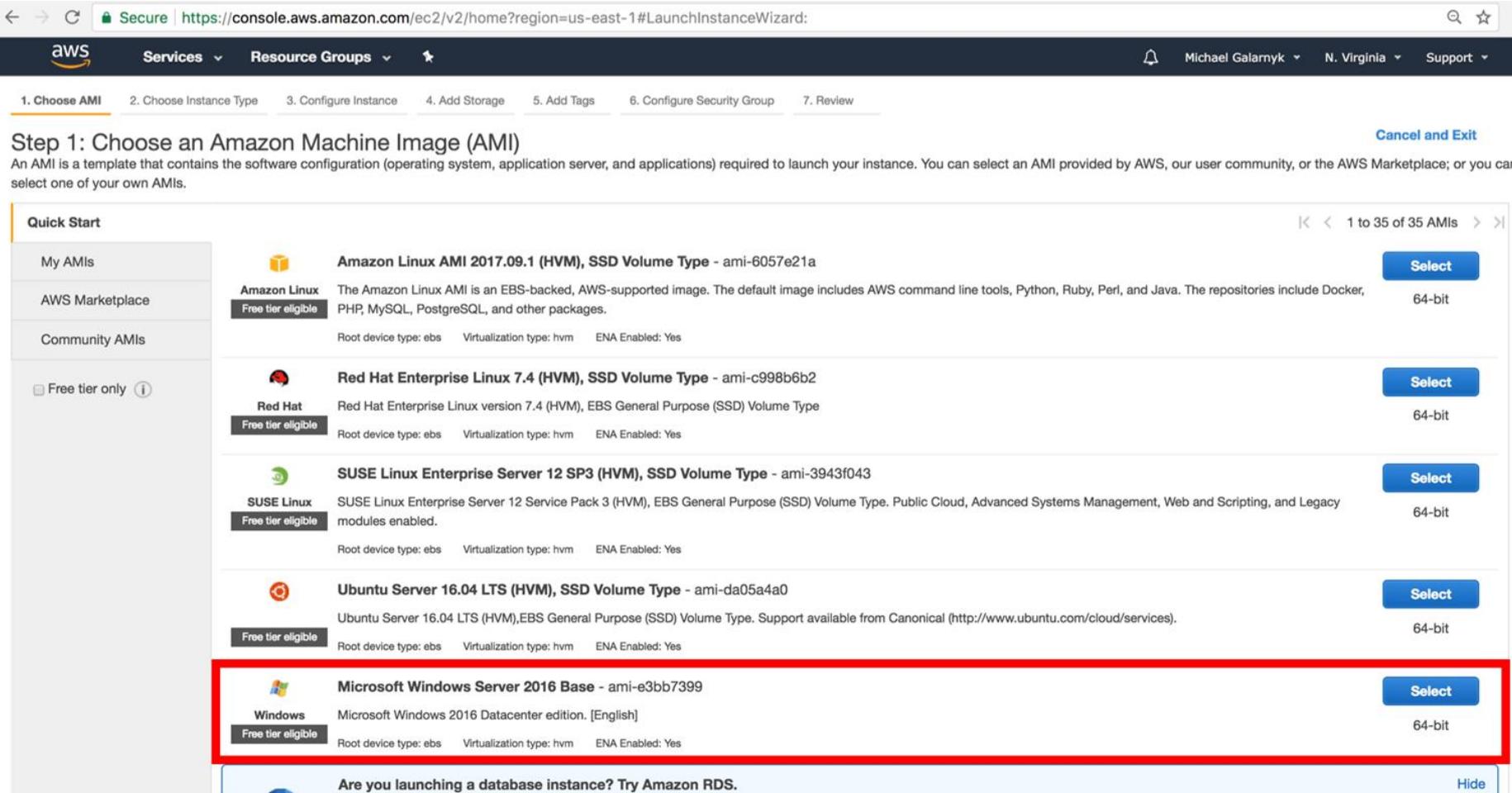
- Step 3: On the Amazon EC2 console, click on Launch Instance.



The screenshot shows the AWS EC2 Dashboard. The left sidebar menu is visible, with 'Instances' selected. The main content area displays a message: 'You do not have any running instances in this region.' It includes links for 'Getting Started Guide' and 'Click the Launch Instance button to start your own server.' A prominent blue 'Launch Instance' button is located in the center. The browser's address bar shows the URL: <https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#instances:sort=instanceId>.

Demo: IaaS on AWS (Compute as a service)

- Step 4: Select the required operating system image.



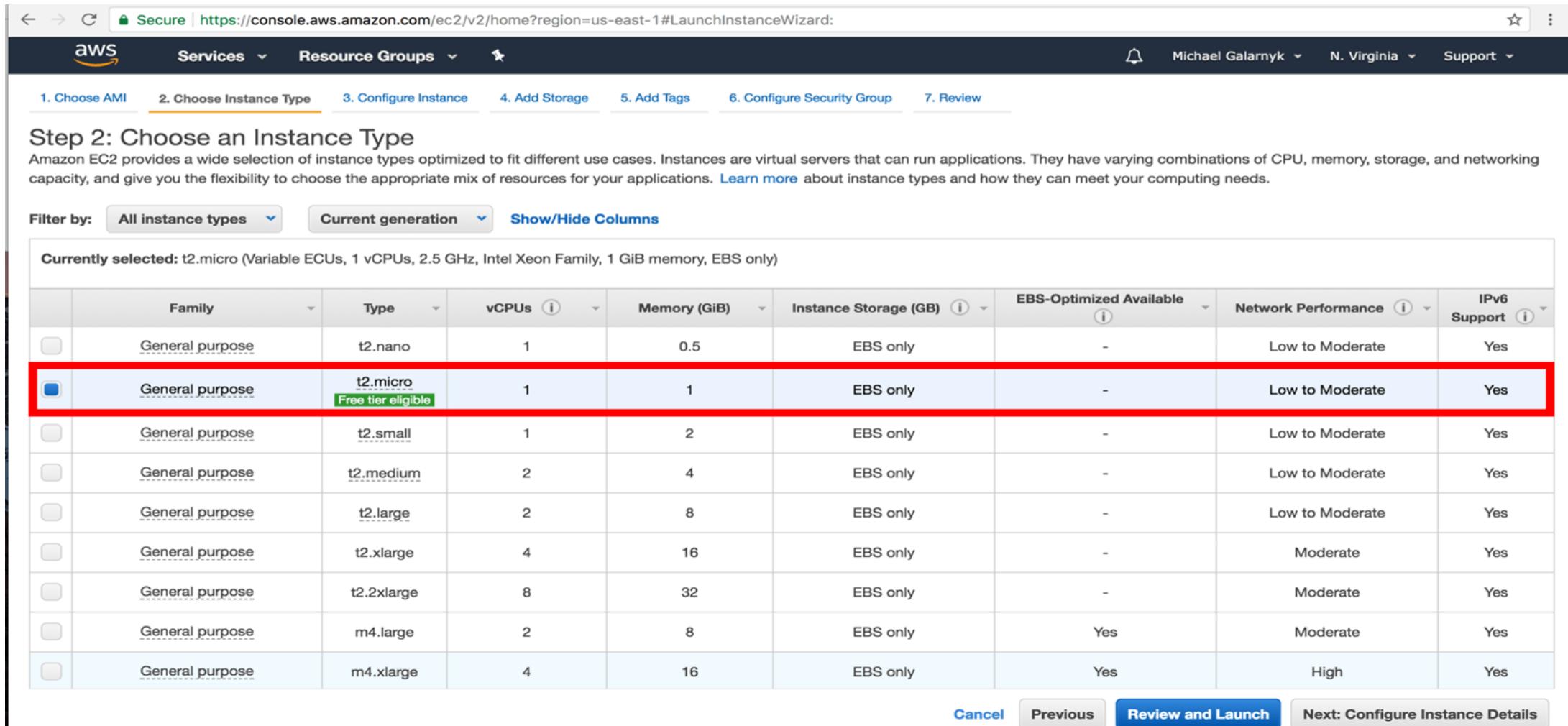
The screenshot shows the AWS Launch Instance Wizard Step 1: Choose AMI. The page title is "Step 1: Choose an Amazon Machine Image (AMI)". It explains that an AMI is a template containing software configuration required to launch an instance. The user can select an AWS-provided AMI or one from the Marketplace or their own AMIs. A sidebar on the left lists "Quick Start" options: My AMIs, AWS Marketplace, Community AMIs, and a "Free tier only" checkbox. The main content area displays a list of 35 AMIs:

Image	Name	Description	Select	Architecture
Amazon Linux icon	Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type - ami-6057e21a	The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.	Select	64-bit
Red Hat icon	Red Hat Enterprise Linux 7.4 (HVM), SSD Volume Type - ami-c998b6b2	Red Hat Enterprise Linux version 7.4 (HVM), EBS General Purpose (SSD) Volume Type	Select	64-bit
SUSE icon	SUSE Linux Enterprise Server 12 SP3 (HVM), SSD Volume Type - ami-3943f043	SUSE Linux Enterprise Server 12 Service Pack 3 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.	Select	64-bit
Ubuntu icon	Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-da05a4a0	Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).	Select	64-bit
Windows icon	Microsoft Windows Server 2016 Base - ami-e3bb7399	Microsoft Windows 2016 Datacenter edition. [English]	Select	64-bit

A red box highlights the Microsoft Windows Server 2016 Base AMI. A callout at the bottom suggests trying Amazon RDS for database instances.

Demo: IaaS on AWS (Compute as a service)

- Step 5: Select the required hardware configuration



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Michael Galarnyk N. Virginia Support

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

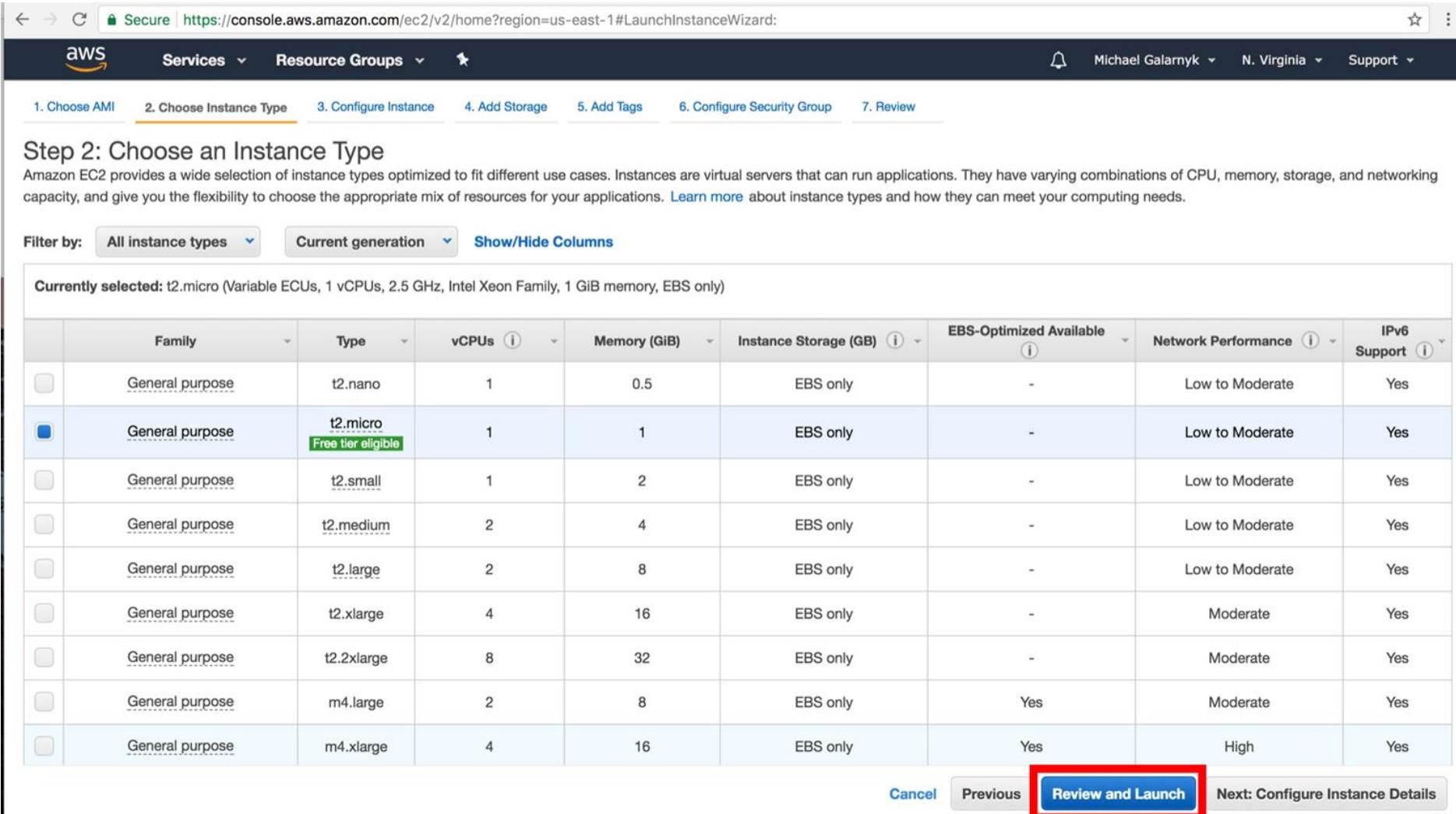
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
General purpose	m4.large	2	8	EBS only	Yes	Moderate	Yes
General purpose	m4.xlarge	4	16	EBS only	Yes	High	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Demo: IaaS on AWS (Compute as a service)

- Step 6: click on "Review and Launch"



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups Michael Galarnyk N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

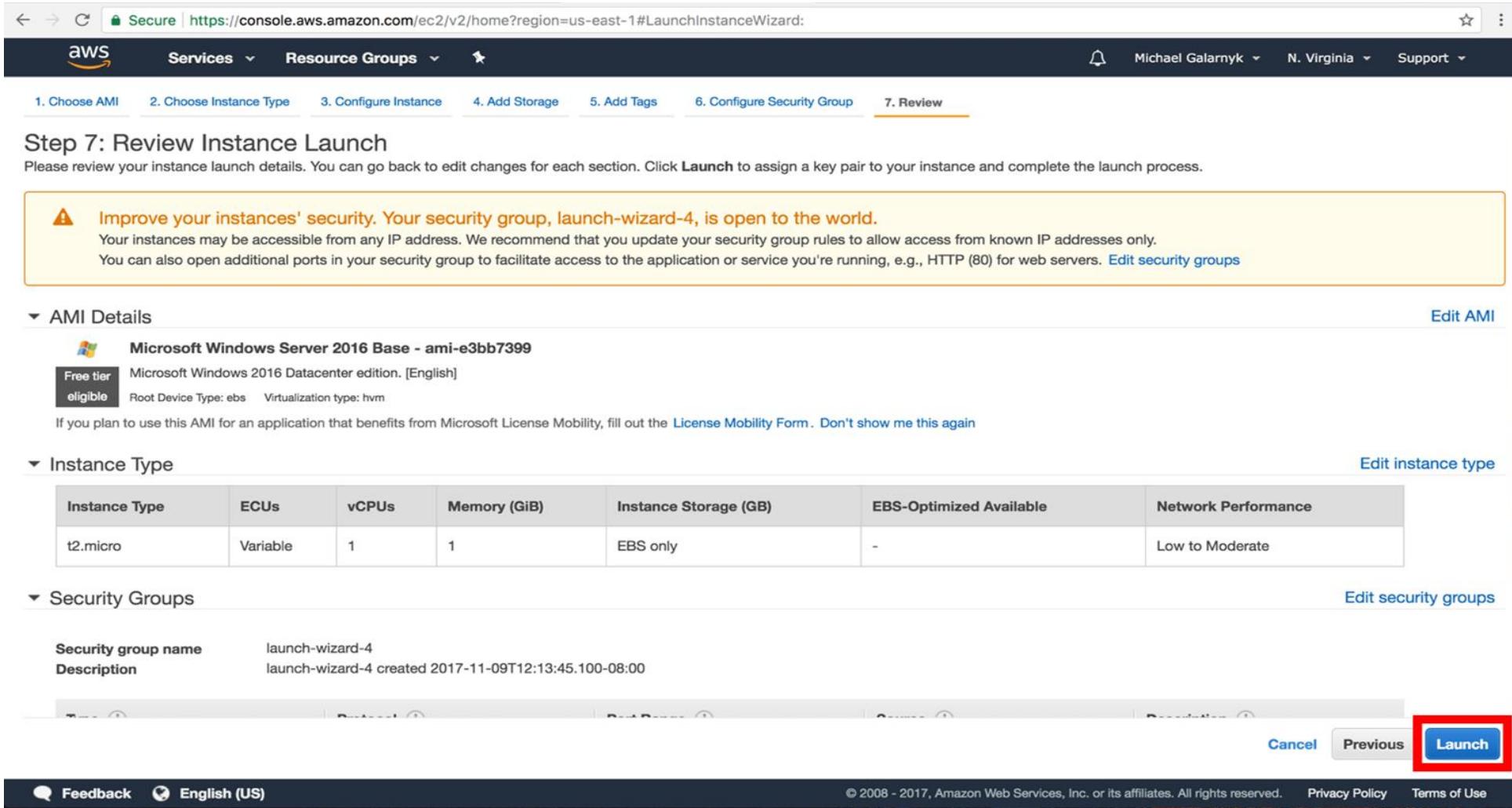
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
	General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
	General purpose	m4.large	2	8	EBS only	Yes	Moderate	Yes
	General purpose	m4.xlarge	4	16	EBS only	Yes	High	Yes

Cancel Previous **Review and Launch** Next: Configure Instance Details

Demo: IaaS on AWS (Compute as a service)

- Step 7: Click on Launch.



The screenshot shows the AWS Launch Instance Wizard Step 7: Review Instance Launch page. The URL is <https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard>. The top navigation bar includes the AWS logo, Services, Resource Groups, Michael Galarnyk, N. Virginia, and Support. The breadcrumb navigation shows 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review (the current step).

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

AMI Details

Microsoft Windows Server 2016 Base - ami-e3bb7399
Free tier eligible Microsoft Windows 2016 Datacenter edition. [English]
Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from Microsoft License Mobility, fill out the [License Mobility Form](#). Don't show me this again

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

Security Groups

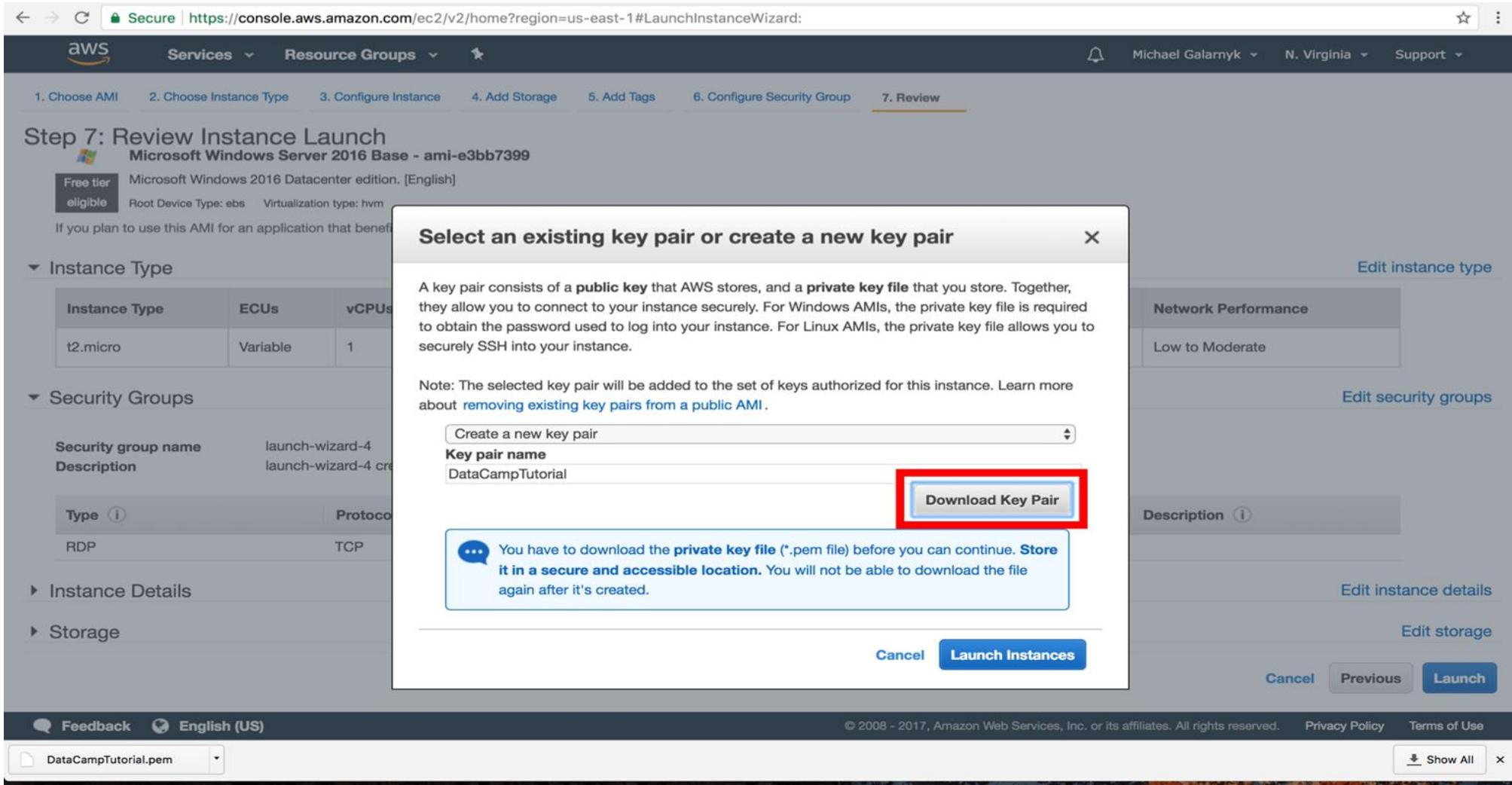
Security group name: launch-wizard-4
Description: launch-wizard-4 created 2017-11-09T12:13:45.100-08:00

At the bottom right, there are three buttons: Cancel, Previous, and Launch. The Launch button is highlighted with a red box.

CLOUD COMPUTING

Demo: IaaS on AWS (Compute as a service)

- Step 8: 7. Select "Create a new key pair". Click on "Download Key Pair". This will download the key. Keep it somewhere safe.



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

Michael Galarmy N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Microsoft Windows Server 2016 Base - ami-e3bb7399

Free tier eligible

Microsoft Windows 2016 Datacenter edition, [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from a free tier, you can use this instance type.

Instance Type

Instance Type	ECUs	vCPUs
t2.micro	Variable	1

Security Groups

Security group name: launch-wizard-4

Description: launch-wizard-4 created

Type: RDP Protocol: TCP

Instance Details

Storage

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name: DataCampTutorial

Download Key Pair

You have to download the private key file (*.pem file) before you can continue. Store it in a secure and accessible location. You will not be able to download the file again after it's created.

Cancel Launch Instances

Network Performance: Low to Moderate

Edit security groups

Description

Edit instance details

Edit storage

Cancel Previous Launch

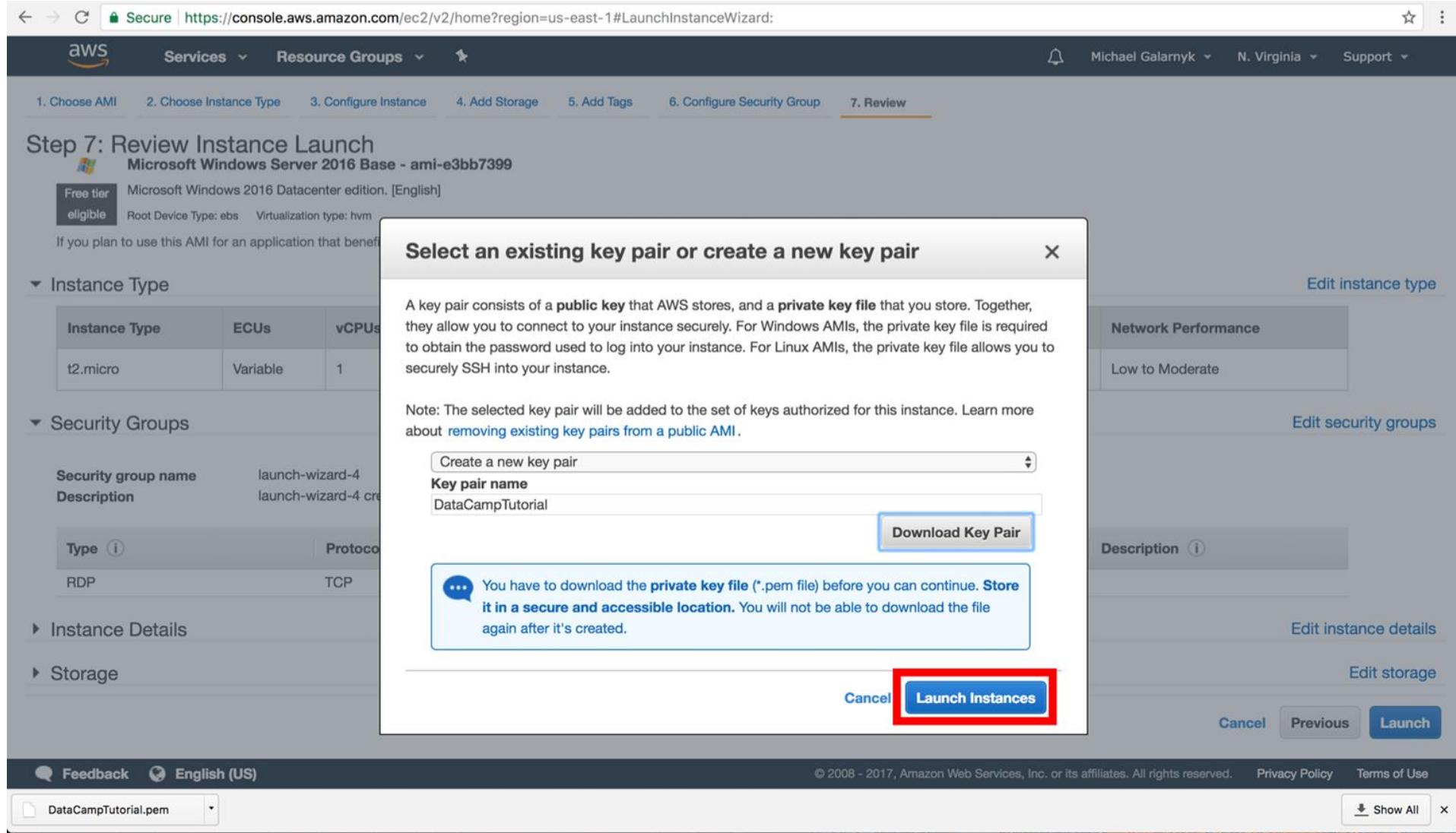
Feedback English (US)

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

DataCampTutorial.pem Show All

Demo: IaaS on AWS (Compute as a service)

- Step 9: Next, click on "Launch Instances".



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

Michael Galarnyk N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Microsoft Windows Server 2016 Base - ami-e3bb7399

Free tier eligible Microsoft Windows 2016 Datacenter edition, [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from enhanced security, consider using a key pair.

Instance Type

Instance Type	ECUs	vCPUs
t2.micro	Variable	1

Security Groups

Security group name	Description
launch-wizard-4	launch-wizard-4 created

Type RDP Protocol TCP

Instance Details

Storage

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name DataCampTutorial

Download Key Pair

You have to download the **private key file (*.pem file)** before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

Network Performance Low to Moderate

Edit instance type Edit security groups

Description

Edit instance details Edit storage

Cancel Previous Launch

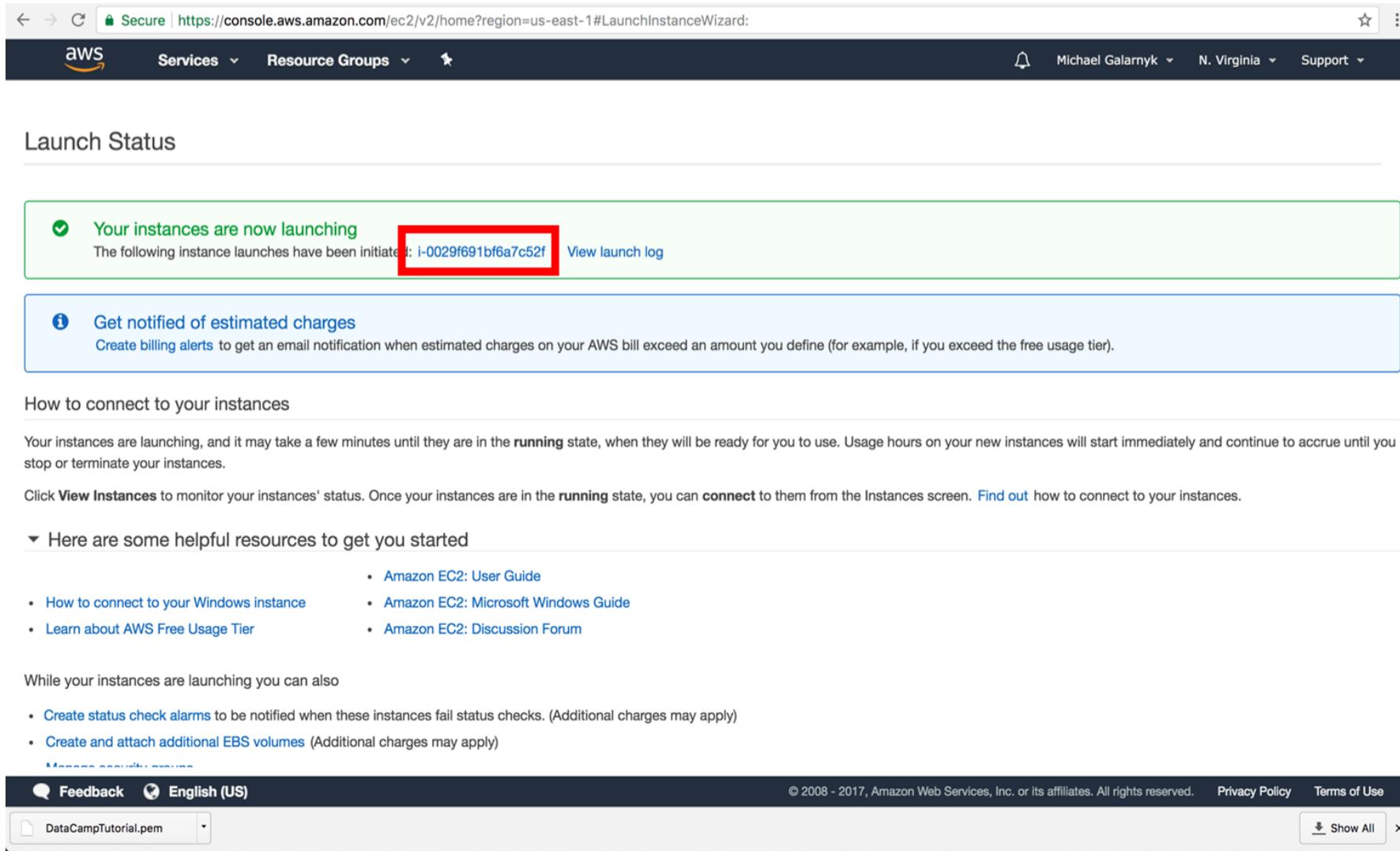
Feedback English (US)

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

DataCampTutorial.pem Show All

Demo: IaaS on AWS (Compute as a service)

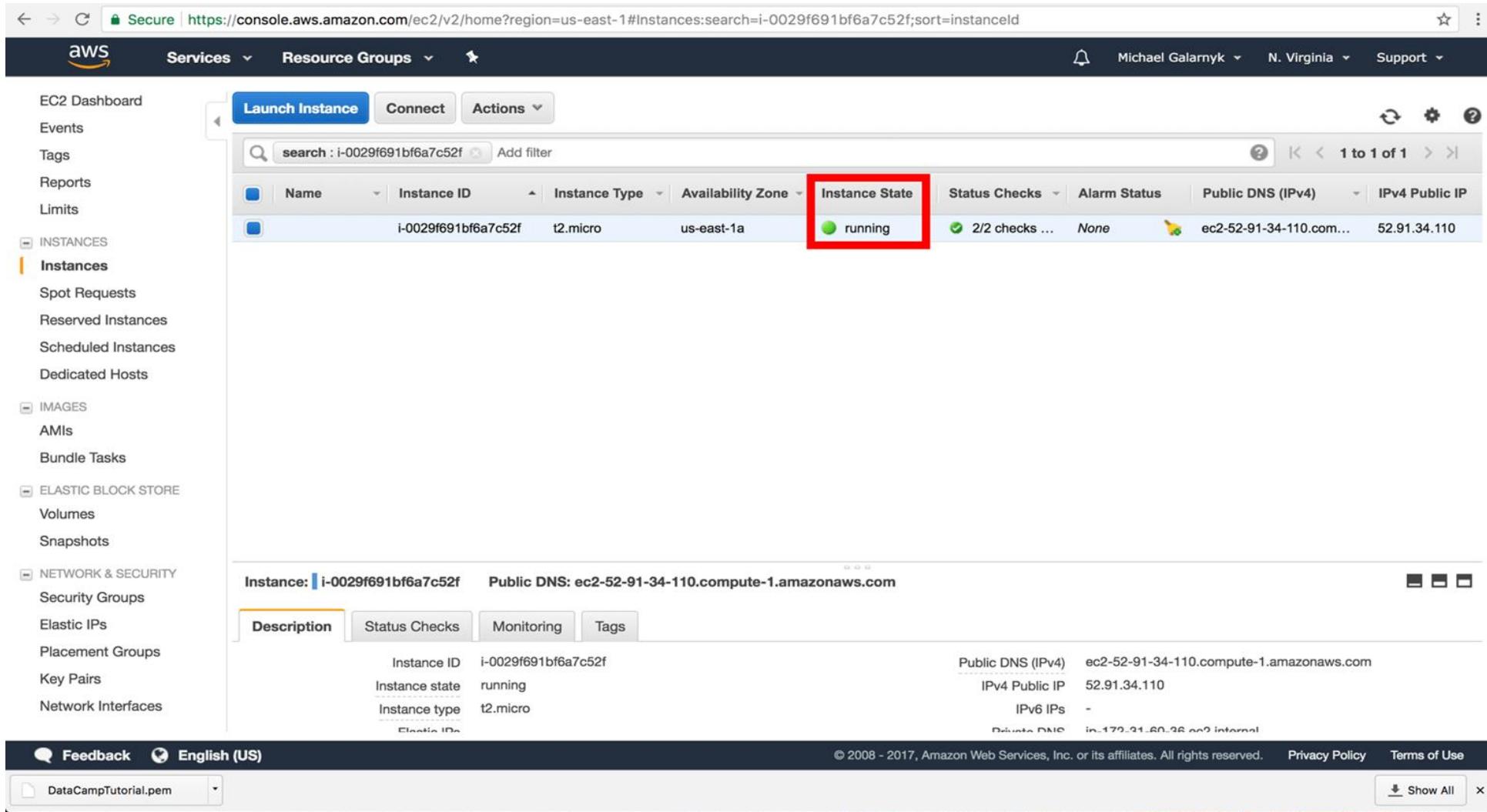
- Step 10: The instance is now launched. Go back to the Amazon EC2 console.



The screenshot shows the AWS EC2 Launch Status page. At the top, there's a navigation bar with links for Services, Resource Groups, and Support, along with user information for Michael Galarnyk and N. Virginia. Below the navigation, the main content area is titled "Launch Status". It displays a green box with a checkmark and the message "Your instances are now launching". Inside this box, it says "The following instance launches have been initiated: i-0029f691bf6a7c52f" and a "View launch log" link. A red rectangle highlights the instance ID "i-0029f691bf6a7c52f". Below this, there's a blue box with an info icon and the message "Get notified of estimated charges". It includes a link to "Create billing alerts" and a note about receiving email notifications when estimated charges exceed a defined amount. Further down, under "How to connect to your instances", it explains that instances are launching and may take a few minutes to reach the "running" state. It also suggests monitoring instances via the "View Instances" link and provides a "Find out" link for connecting. A section titled "Here are some helpful resources to get you started" lists several links, including "Amazon EC2: User Guide", "How to connect to your Windows instance", "Learn about AWS Free Usage Tier", "Amazon EC2: Microsoft Windows Guide", and "Amazon EC2: Discussion Forum". At the bottom, there are links for "Feedback", "English (US)", "DataCampTutorial.pem", and "Show All". The footer contains copyright information for Amazon Web Services (2008-2017) and links for "Privacy Policy" and "Terms of Use".

Demo: IaaS on AWS (Compute as a service)

- Step 11: Observe the instance is up and running



The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, Instances (which is selected and highlighted in orange), Spot Requests, Reserved Instances, Scheduled Instances, Dedicated Hosts, Images, AMIs, and Bundle Tasks. Below that are sections for Elastic Block Store (Volumes, Snapshots) and Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces). The main content area has tabs for Launch Instance, Connect, and Actions. A search bar at the top right says "search : i-0029f691bf6a7c52f". The main table has columns for Name, Instance ID, Instance Type, Availability Zone, and Instance State. The first row shows an instance named "i-0029f691bf6a7c52f" with type "t2.micro" in "us-east-1a". The "Instance State" column shows a green circle and the word "running", which is highlighted with a red box. Below the table, there's a detailed view for the instance "i-0029f691bf6a7c52f" with Public DNS "ec2-52-91-34-110.compute-1.amazonaws.com". The "Description" tab is selected, showing fields like Instance ID, Instance state (running), Instance type (t2.micro), and various IP addresses (Public DNS IPv4, IPv4 Public IP, IPv6 IPs, Private DNS). At the bottom, there are links for Feedback, English (US), Privacy Policy, Terms of Use, and a dropdown menu showing "DataCampTutorial.pem".

- **Flexible:** The most flexible cloud computing model.
- **Control:** Clients retain complete control of their infrastructure
- **Pay-as-you-Go:** Unlike traditional IT, IaaS does not require any upfront, capital expenditures, and end users are only billed for what they use.
- **Speed:** With IaaS, users can provision small or vast amounts of resources in a matter of minutes, testing new ideas quickly or scaling proven ones even quicker.
- **Availability:** Through things like multizone regions, the availability and resiliency of cloud applications can exceed traditional approaches.
- **Scale:** With seemingly limitless capacity and the ability to scale resources either automatically or with some supervision, it's simple to go from one instance of an application or workload to many.
- **Latency and performance:** Given the broad geographic footprint of most IaaS providers, it's easy to put apps and services closer to your users, reducing latency and improving performance.

- **Startups and small companies** may prefer IaaS to avoid spending time and money on purchasing and creating hardware and software.
- **Larger companies** may prefer to retain complete control over their applications and infrastructure, but they want to purchase only what they actually consume or need.
- **Companies experiencing rapid growth** like the scalability of IaaS, and they can change out specific hardware and software easily as their needs evolve.
- **Website hosting** :Running websites using IaaS can be less expensive than traditional web hosting.
- **Storage, backup and recovery** : Organizations avoid the capital outlay for storage and complexity of storage management, which typically requires a skilled staff to manage data. IaaS is useful for handling unpredictable demand and steadily growing storage needs. It can also simplify planning and management of backup and recovery systems.
- **High-performance computing.** High-performance computing (HPC) on supercomputers, computer grids or computer clusters helps solve complex problems involving millions of variables or calculations.
- **Big data analysis.** Mining data sets to locate or tease out hidden patterns requires a huge amount of processing power, which IaaS economically provides.

Anytime you are unsure of a new application's demands, IaaS offers plenty of flexibility and scalability.

- **Security:** While the customer is in control of the apps, data, middleware, and the OS platform, security threats can still be sourced from the host or other virtual machines (VMs). Insider threat or system vulnerabilities may expose data communication between the host infrastructure and VMs to unauthorized entities.
- **Multi-tenant security:** Since the hardware resources are dynamically allocated across users as made available, the vendor is required to ensure that other customers cannot access data deposited to storage assets by previous customers. Similarly, customers must rely on the vendor to ensure that VMs are adequately isolated within the multi tenant cloud architecture.
- **Internal resources and training:** Additional resources and training may be required for the workforce to learn how to effectively manage the infrastructure. Due to inadequate control into the infrastructure however, monitoring and management of the resources may be difficult without adequate training and resources available in house.

- Amazon Web Services : Amazon EC2.
- Microsoft Azure : Azure Virtual Machines.
- Google Cloud : Google compute engine.
- IBM Cloud : IBM Cloud Private.
- Digital Ocean : Digital Ocean Droplets.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

REST and Web Services

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Service oriented architecture (SOA)

SOA, or **service-oriented architecture**, defines a way to make software components reusable via service interfaces. These interfaces utilize common communication standards in such a way that they can be rapidly incorporated into new applications without having to perform deep integration each time.

Each service in an SOA embodies the code and data integrations required to execute a complete, discrete business function. The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the integration is implemented underneath. The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or JSON/HTTP—to send requests to read or change data. The services are published in a way that enables developers to quickly find them and reuse them to assemble new applications.

Two major service-oriented architecture styles :

1. **REST** (REpresentational State Transfer)
2. **SOAP based WS** (Web Services)

REST(Representational State Transfer)

- Cloud architecture involves number of distributed autonomous systems or components or applications frequently communicating or interacting between themselves for performing an application request.
- REST is an architectural (sometimes also called programming) style for providing a set of rules to be used for building computer systems on the web, and making it easier for systems to communicate with each other.
- REST helps in building a client-friendly distributed systems that are simple to understand and simple to scale or following the REST principles while designing your application (distributed system), you will end up with a system that exploits the Web's architecture to your benefit
- An API which adheres to these constraints are considered RESTful and helps getting benefits of a Client Server distributed architecture by reducing the complexities of distributed systems (often called RESTful systems) and allows the benefits to be achieved more easily.
- REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web APIs. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.

CLOUD COMPUTING

REST(REpresentational State Transfer) - Design Principles

The following are some of the mandatory constraints (principles) for designing a RESTful system.

1. Client Server Constraint (Separation of concerns)

This sets the constraint on the system to be built in such a fashion where the client can change and evolve without having to change anything on the server, and also on the contrary different aspects of the Server should also be able to be changed without breaking clients. Or

Its about how the client sends the server a message, and how the server rejects or responds to the client

2. Stateless Constraint

The communication between the client and the server must remain stateless between requests. Each request the client makes should contain all information needed for the server to answer successfully. All of the state information should be transferred back to the client as part of the response and cannot take advantage of any stored context on the server.

Session state is therefore kept entirely on the client. This is the “S” and “T” in REST; we are always giving the state back to the client. Not keeping the state in the Server impacts performance which are being addressed by the other concerns.

3. Cache Constraint

In order to improve network efficiency, cache constraints are added require that the data within a Server's response to a request, be implicitly or explicitly labeled/mark as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

CLOUD COMPUTING

REST(Representational State Transfer)

Constraints to designing a RESTful system (Cont.)

4. Uniform Interface Constraint

A uniform interface makes it generic and improves the overall visibility of interactions on how the client and server exchange requests and responses. Implementations are decoupled from the services they provide. This could impact performance for non web based data interactions. REST is defined by 4 interface constraints

- a. Resource and Resource Identification
- b. Manipulation of Resources through Representations
- c. Self-descriptive messages
- d. Hypermedia as the engine of application state

These will be discussed further.

5. Layered System Constraint

Messages between a client and server can go through a hierarchy of intermediate components (could be load balancers, proxy servers, firewalls, or other types of devices) where these intermediate components should not be able to “see” the next layer. Thus the interaction between the client and server should not be affected by these devices and thus ensures independence and bounding of complexity. All communication should remain consistent, even if one of the layers is added or removed

REST(Representational State Transfer) Functioning

- The REST architectural style is designed for network-based applications, specifically client-server applications.
- Clients can only access resources using **URIs**.
- Client requests a resource using a URI and the server responds with a **representation** of the resource.
- To ensure responses can be interpreted by the widest possible number of client **applications** a representation of the resource is sent in **hypermedia** format.
- Thus, a resource is manipulated through hypermedia representations transferred in **messages** between the clients and servers.



CLOUD COMPUTING

REST Architectural Elements – Resources and Resource Identification through URIs (T2)

Resource and Resource Identification: The key abstraction of information in REST is a resource. A resource is a “Thing”. Any information that can be named can be a resource, such as a document or image or a temporal service.

Eg. Light on board in seminar hall. If you need to control it, you name it

/pesu/b-block/groundfloor/seminarhall/board/light/1

The RESTful web service exposes a set of resources which identify targets of interaction with its clients. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability.

URI – Uniform Resource Identifier	URL – Uniform Resource Locator
<ul style="list-style-type: none">▪ The name of the object on the web▪ Identifies a resource by<ul style="list-style-type: none">▫ Name▫ Location▫ Or both	<ul style="list-style-type: none">▪ Subset of an URI▪ Specifies where to find a resource – the location▪ How to retrieve the resource

Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) Interfaces which supports operations as below :

- GET — retrieve a specific resource or a collection of resources
- POST — Create or partial update of a resource
- PUT — Create or update resource by replacement.
- DELETE — remove a resource

So typically in your application clients say for a book lending library something like

- GetListOfBooks()
- AddBookToShoppingCart()

Need to define these operations. No standard style exists. These operations are implemented using the Interfaces described earlier.

- REST Operations

GET

- Retrieve representation of a resource

PUT

- Create or update resource by replacement.

POST

- Create or partial update of a resource

DELETE

- Remove a resource

- Usage

GET

- <https://bblock/seminarhall/board/light1.xml>

- To check if light is ON

POST to

- <https://bblock/seminarhall/board/light1.xml>

- To turn on/off the light

- Put ON in the body of the message

CLOUD COMPUTING : Operations

- When we treat resources as things to manipulate via a URL and we restrict ourselves to a small set of operations, there is less need to extensively describe a service.
- It's not about the 50 things you can do with a particular service (which is how SOAP developers view the world). It's about using a standard set of things you can do (PUT, GET, etc.) with a set of resources (robots, spaceships, flightpaths, etc.)
- So how do you FlyShip (`id="ship-123"`, `destination=Planets.Mars`) ?
 - PUT to `/ships/ship-123/flightpath` and send "destination=mars" in the body of the message
 - Now you can poll that URI using GET for flight updates.
 - or
 - POST to `/ships/ship-123/flightpaths` and send "/planets/mars" in the body of the message. The server might return something like `/ships/ship-123/flightpaths/456` to indicate that the ship is now flying to Mars and that a new flightpath resource was created to track it. You can poll that URI using GET for flight updates.
- Instead of defining a new operation such as `StopFlight (id)` to stop the flight, do this:
 - DELETE `/ships/ship-123/flightpaths/456`
- How does a web browser know what to do with a URL? All resources support the same operations, and GET is one of them. The browser GETs until everything has been gotten!

CLOUD COMPUTING : Characteristics of these Operations: Safe - Idempotent

- Safe
 - Does not modify the resources
 - Example
 - GET
- Idempotent
 - Idempotent has no additional effect if it is called more than once with same input parameters
 - Can repeatedly perform operations.
 - No effect on servers
 - How would you like to pay for a seat multiple times?

CLOUD COMPUTING : Operations with characteristics

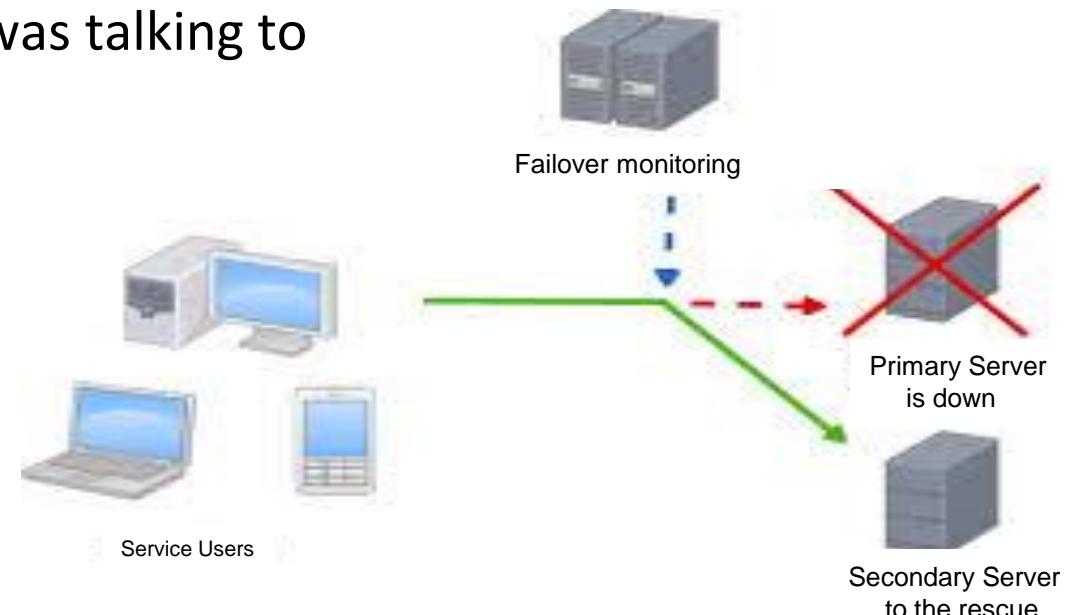
Operation	Safe	Idempotent	When to use
GET	Yes	Yes	Mostly for retrieving resources. Can call multiple times.
PUT	No	Yes	Modifies a resource but no additional impact if called multiple times
POST	No	No	Modifies resources, multiple calls will cause additional effect if called with same parameter
DELETE	No	Yes	Removing a resource.

Stateless Interactions: Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa. In this way, both the server and the client can understand any message received, even without seeing previous messages. Each client request is treated independently.

Statelessness benefits :

1. Clients isolated against changes on server
2. Promotes redundancy - unlocks performance:
 - a. don't really need to know which server client was talking to
 - b. No synchronization overhead

No state saved on server,
so even if server fails, the
client connects to another
server and continue



CLOUD COMPUTING

REST Principles - Representations

Representation : A representation is a snapshot in time of the state of a given resource. It's a sequence of bytes made up of the data, plus representation metadata to describe those bytes.

- This captures the current or intended state of that resource and helps transferring that representation between the interacting components.
- The metadata could be in the form of binary or textual key-value pairs.
- The data format of a representation is known as a media type and this type identifies a specification that defines how a representation is to be processed.
- The message type is ***hypermedia***, which refers to any content that contains links to other forms of media such as images, movies, and text. hypermedia links in the API response contents. It allows the client to dynamically navigate to the appropriate resources by traversing the hypermedia links.
- Navigating hypermedia links is conceptually the same as browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.
- All the response representation need to be self-descriptive

A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents.

In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.) i.e. the resources can have **multiple representations**. RESTful systems empowers client to ask for data in a form they understand.

Example:

```
GET /articles/23 HTTP/1.1
Accept: text/html, application/xhtml
```

Client request

```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

Server Response

Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

1. Rest is not a standard :

It is a design and architectural style for large scale distributed systems

2. Rest is not same as http:

Http can also be used in non-RESTful way. Example : SOAP services that use http to transport data.

Web Service: a software system designed to support interoperable machine-to-machine interaction over a network.

The term “web service” is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.

Once a web service is deployed, other applications and other web services can discover and invoke the deployed service. Other systems interact with the web service in a manner prescribed by its description.

A web service is one of the most common instances of an SOA implementation.

The two prominent ways of implementing Web Services. This could be using the approach of :

1. Simple Object Access Protocol (SOAP)
2. REST (we have discussed this at length)

Simple Object Access Protocol (SOAP)

SOAP is a protocol which was designed before REST came into the picture. The main idea behind creating SOAP was to ensure that programs built on different platforms and programming languages could securely exchange data.

SOAP provides a structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP.

A SOAP message consists of element called **envelope**, which is used to encapsulate all of the data in the SOAP message which contains a **Header** element that contains header information such as **authentication credentials** which can be used by the calling application, and a **body** element that carries the payload of the message.

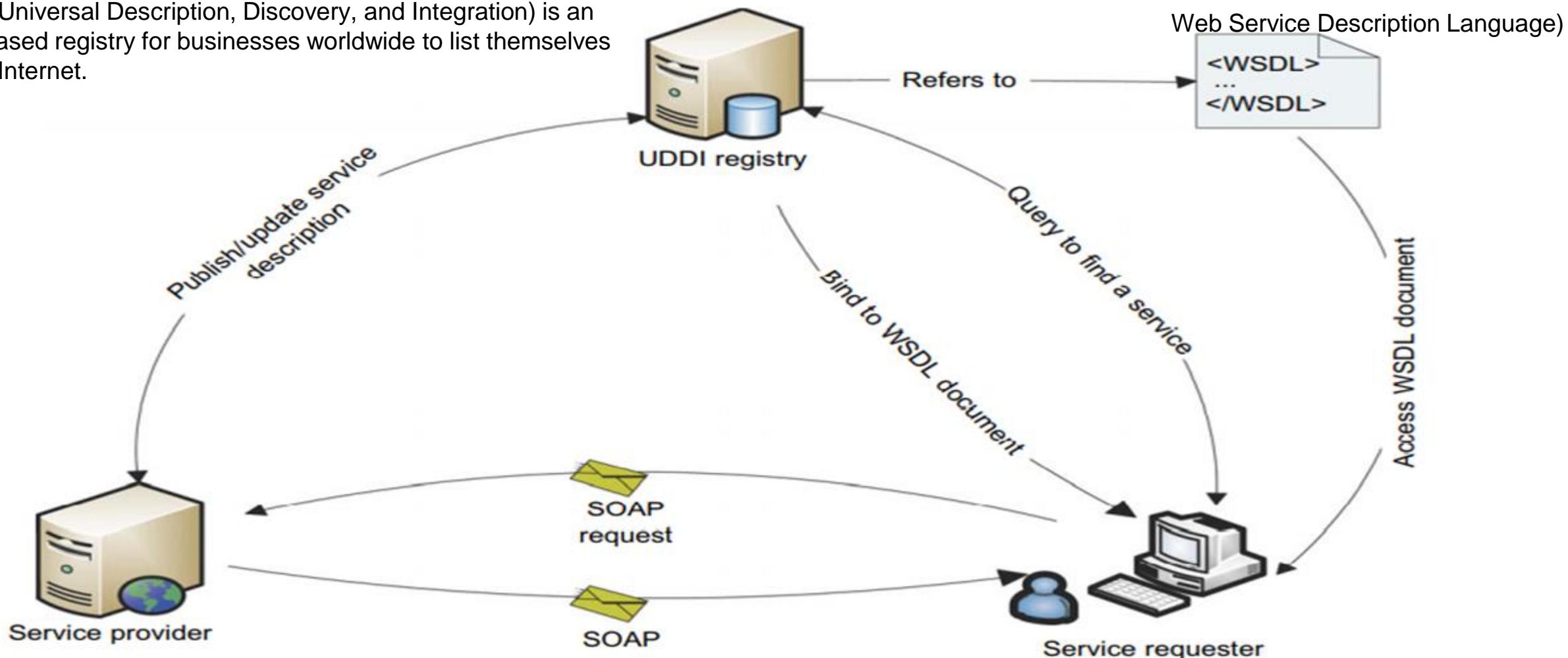
The content of the payload will be marshaled by the sender's SOAP engine and un-marshaled at the receiver side.

WSDL or web service description language describes the functionality of the SOAP based webservice

Web Services in action say using SOAP

(T2)

UDDI (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet.



A simple web service interaction among provider, user, and the UDDI registry.

Refer to Figure 5.2(previous slide):

Step 1 : The service provider publishes its location and description to the UDDI registry.

Step 2: The service requester queries the UDDI registry using names, identifiers, or the specification. The UDDI registry finds the corresponding service and returns the WSDL document of the service.

Step 3: The service requester reads the WSDL document and forms a SOAP message binding to all constraints in the WSDL document.

Step 4: This SOAP message is sent to the web service as the body of an HTTP/SMTP/FTP request.

Step 5: The web service unpacks the SOAP request and converts it into a command that the application can understand and executes the command.

Step 6: The web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP/SMTP/FTP request

Step 7: The client program unpacks the SOAP message to obtain the results



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Platform as a Service(PaaS) Programming Model

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

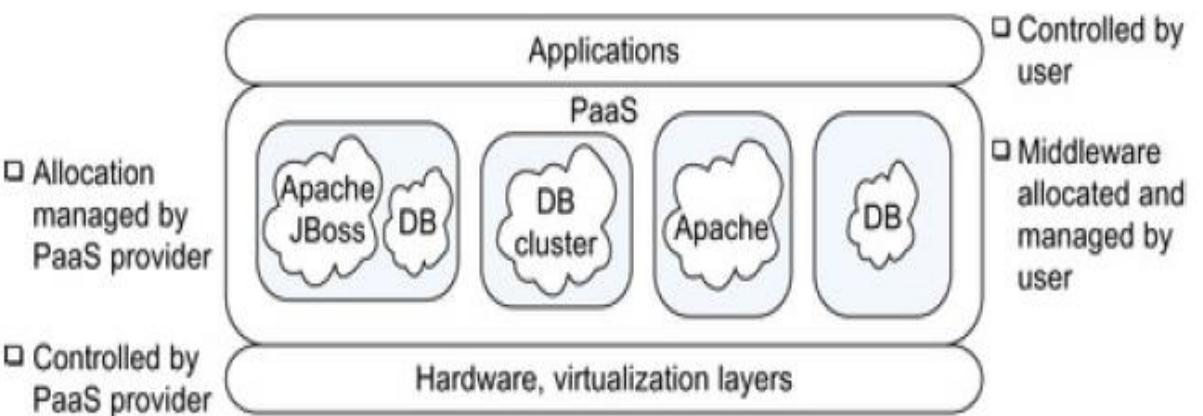
Acknowledgements:

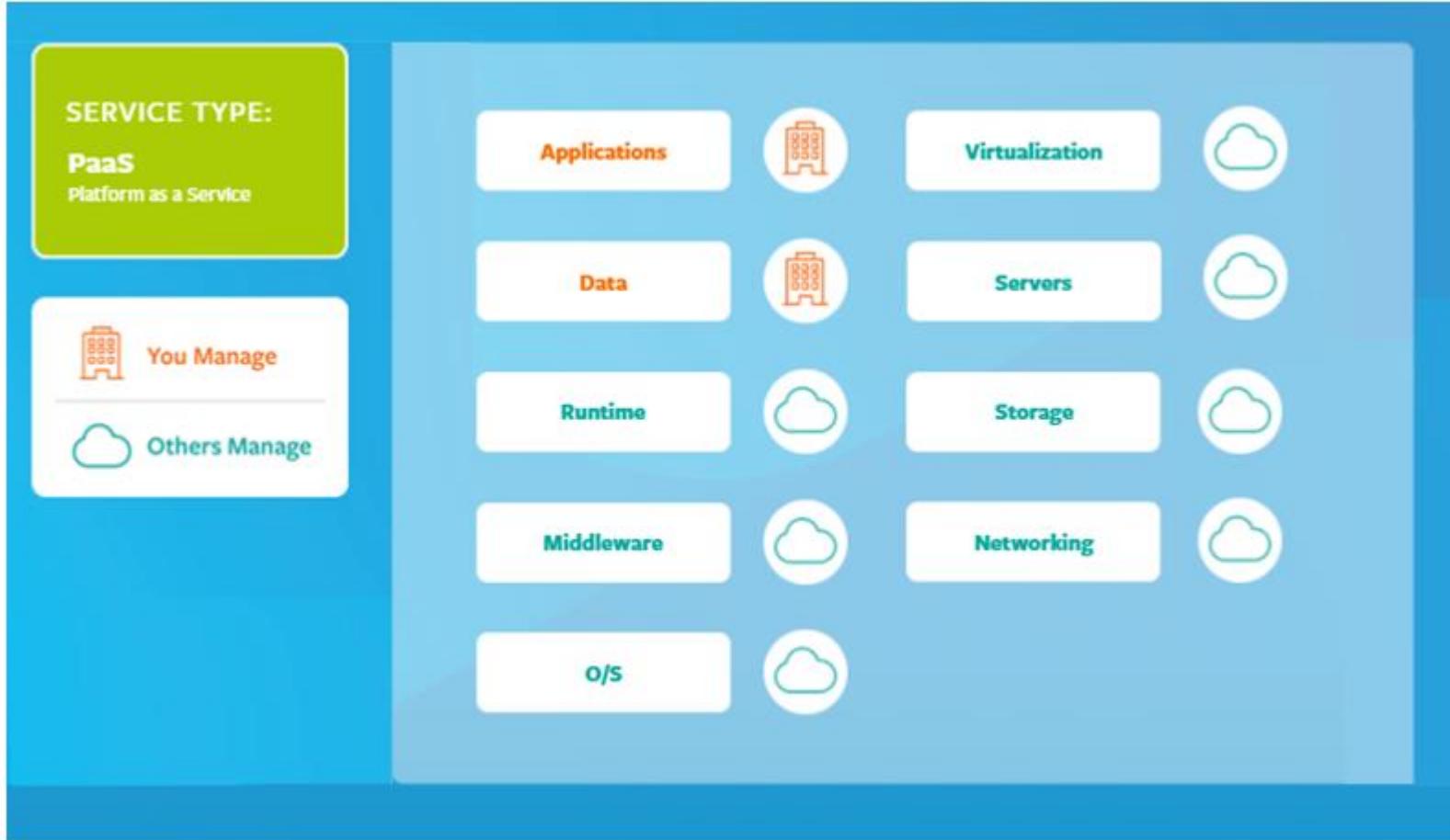
Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Platform as a Service(PaaS) Programming Model (Recap)

Platform as a service (PaaS)

- It's a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications.
- You purchase the resources you need from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection.
- Like IaaS, PaaS includes not only the abstracted infrastructure—servers, storage and networking—but also middleware, development tools, business intelligence (BI) services, database management systems and more. PaaS is designed to support a Cloud application like a web application across the complete lifecycle: building, testing, deploying, managing and updating.
- This platform is delivered via the web, giving developers the freedom to concentrate on building the software without having to worry about operating systems, software updates, storage, or infrastructure.
- The hardware, as well as any mapping of hardware to virtual resources, such as virtual servers, is controlled by the PaaS provider. The cloud user can configure and build on top of this middleware, like defining a new database table in a database





Ref - <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/#ref3>

PaaS Advantages - Reiteration

- 1. Faster time to market:** With PaaS, there's no need to purchase and install the hardware and software you'll use to build and maintain your application development platform and no need for development teams to wait while you do this. You simply tap into the cloud service provider's PaaS resources and begin developing immediately.
- 2. Faster, easier, less-risky adoption of a wider range of resources:** PaaS platforms typically include access to a greater variety of choices up and down the application development stack—operating systems, middleware, and databases, and tools such as code libraries and app components—than you can affordably or practically maintain on-premises. It also lets you test new operating systems, languages, and tools without risk—that is, without having to invest in the infrastructure required to run them.
- 3. Develop for multiple platforms—including mobile—more easily.** Some service providers give you development options for multiple platforms, such as computers, mobile devices and browsers making cross-platform apps quicker and easier to develop.
- 4. Easy, cost-effective scalability:** If an application developed and hosted on-premises starts getting more traffic, you'll need to purchase more computing, storage, and even network hardware to meet the demand, which you may not be able to do quickly enough and can be wasteful (since you typically purchase more than you need). With PaaS, you can scale on-demand by purchasing just the amount of additional capacity you need.

PaaS Advantages - Reiteration (Cont.)

5. **Support geographically distributed development teams:** Because the development environment is accessed over the Internet, development teams can work together on projects even when team members are in remote locations.
6. **Efficiently manage the application lifecycle :** PaaS provides all of the capabilities that you need to support the complete web application lifecycle: building, testing, deploying, managing and updating within the same integrated environment.
7. **Lower costs:** Because there's no infrastructure to build, your upfront costs are lower. Costs are also lower and more predictable because most PaaS providers charge customers based on usage.
8. **Development framework:** PaaS provides a framework that developers can build upon to develop or customize cloud-based applications. PaaS lets developers create applications using built-in software components. Cloud features such as scalability, high-availability and multi-tenant capability are included, reducing the amount of coding that developers must do.
9. **Analytics or business intelligence:** Tools provided as a service with PaaS allow organizations to analyze and mine their data, finding insights and patterns and predicting outcomes to improve forecasting, product design decisions, investment returns and other business decisions.

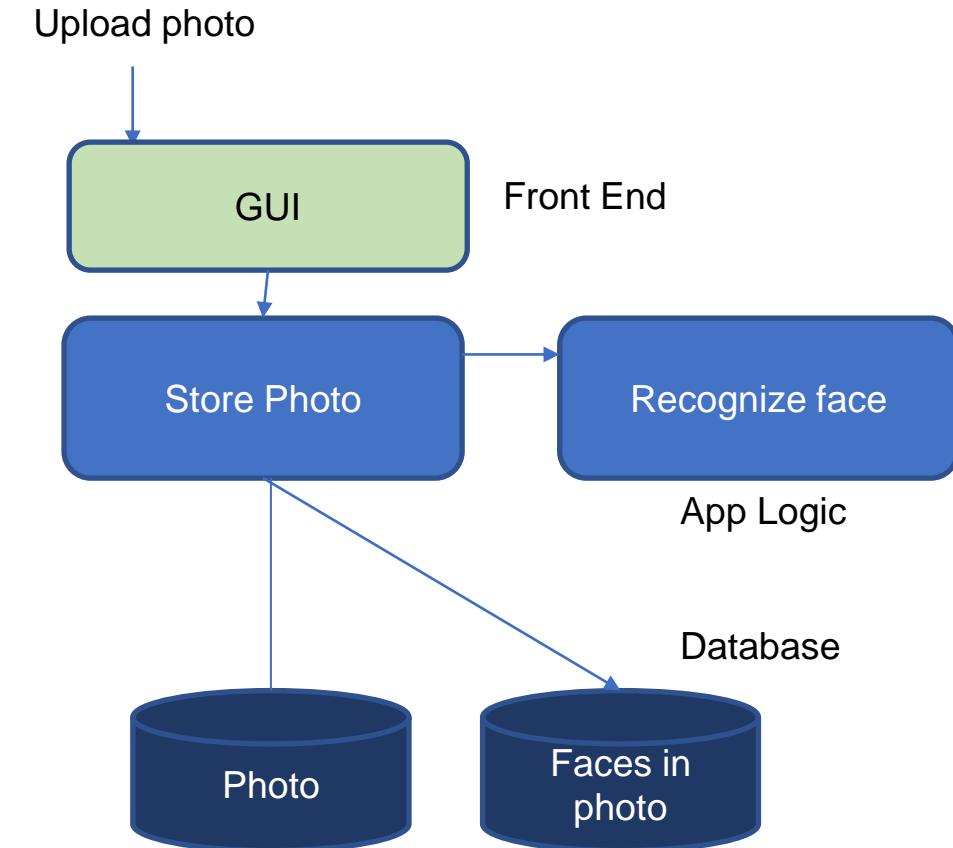
Consider you are building a photo sharing app

- User uploads Photo
- App recognizes faces in photo
- App Stores photos + faces in photo into database

If we are looking at a 3 tier application model, the layers would have something like shown.

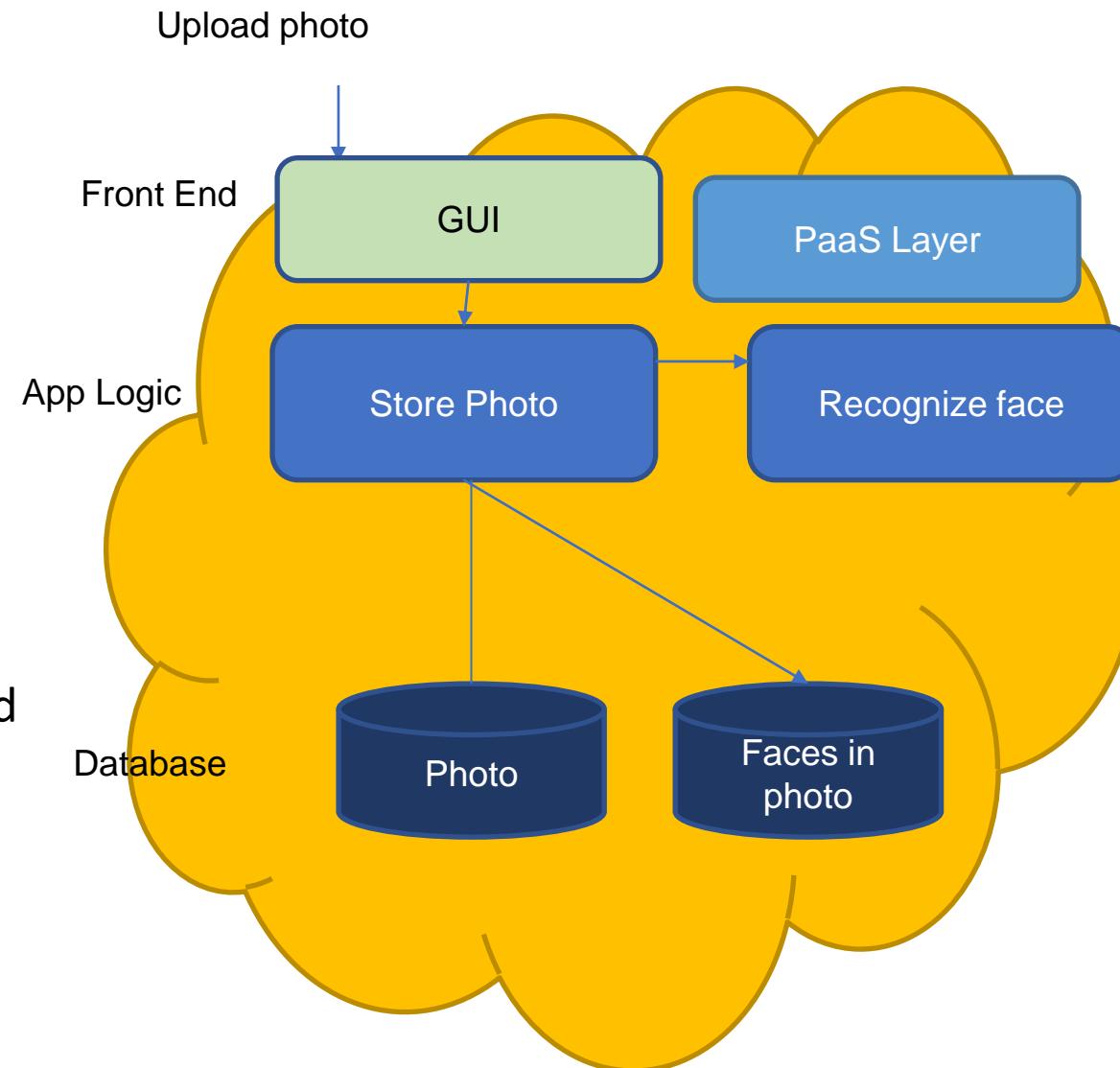
The activities in an IaaS programming model would be

- Start a VM
- Connect a block storage device
- Install Web Server
- Install App Server
- Install Database
- Upload application
- Run Application



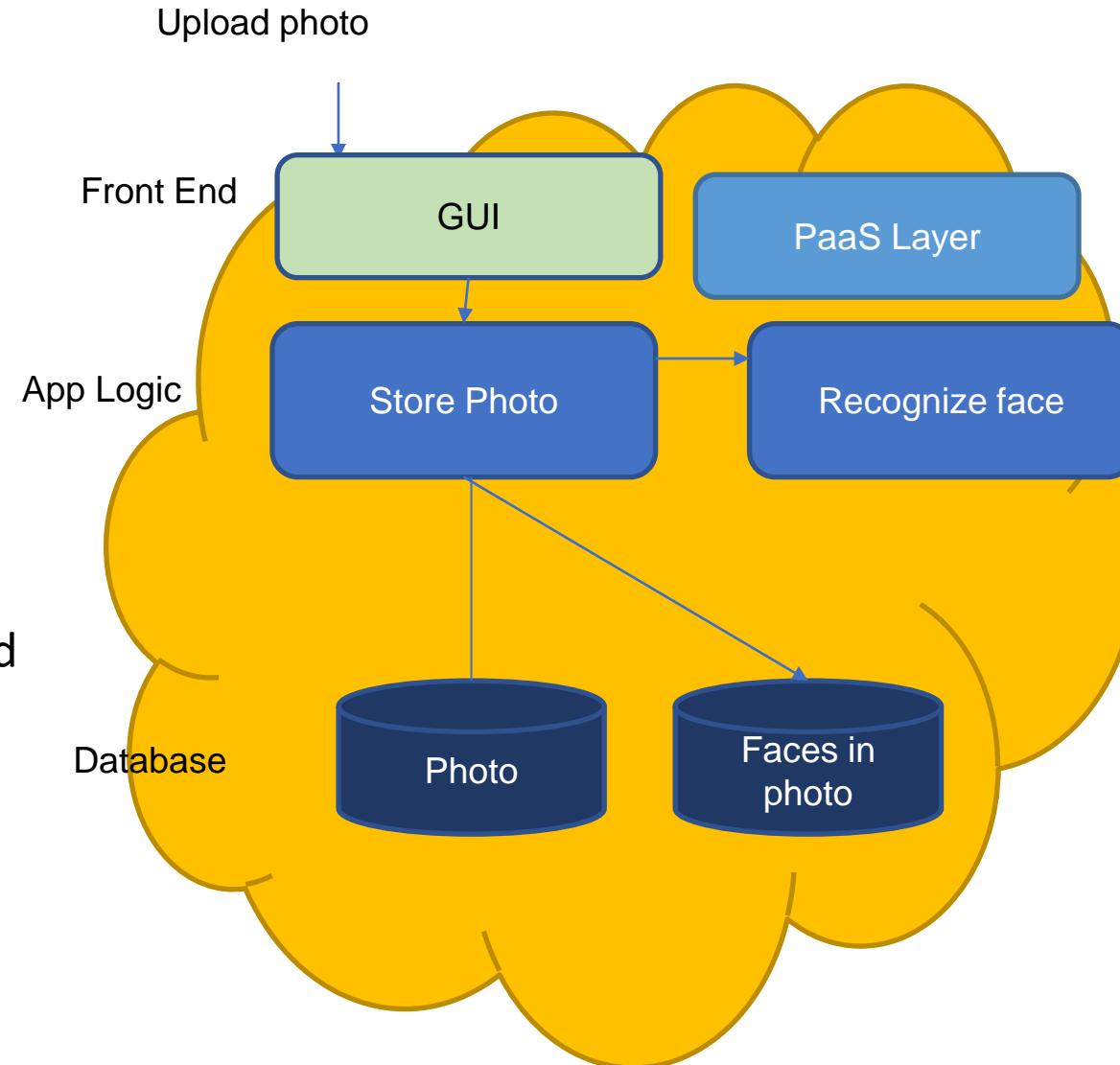
In the PaaS programming model, this would be

- Application is uploaded
 - This will be made up of components/sections
 - Each component specifies environment (e.g., part of code/actions to run under web server or it could be part of the code to run under app server)
 - Specifies initial number of components, scale up / scale down policy
- In Windows Azure
 - Web roles – front end type tasks
 - Worker roles – those that run tasks in the background (Different colours on the figure)
- PaaS Layer
 - Deploy
 - Allocate resources
 - Number of instances



In the PaaS programming model, this would be

- Application
 - Made up of components
 - Each component specifies environment (e.g., run under web server or app server)
 - Specifies initial number of components, scale up / scale down policy
- In Windows Azure
 - Web roles – front end type tasks
 - Worker roles – those that run tasks in the background (Different colours on the figure)
- PaaS Layer
 - Deploy
 - Allocate resources
 - Number of instances

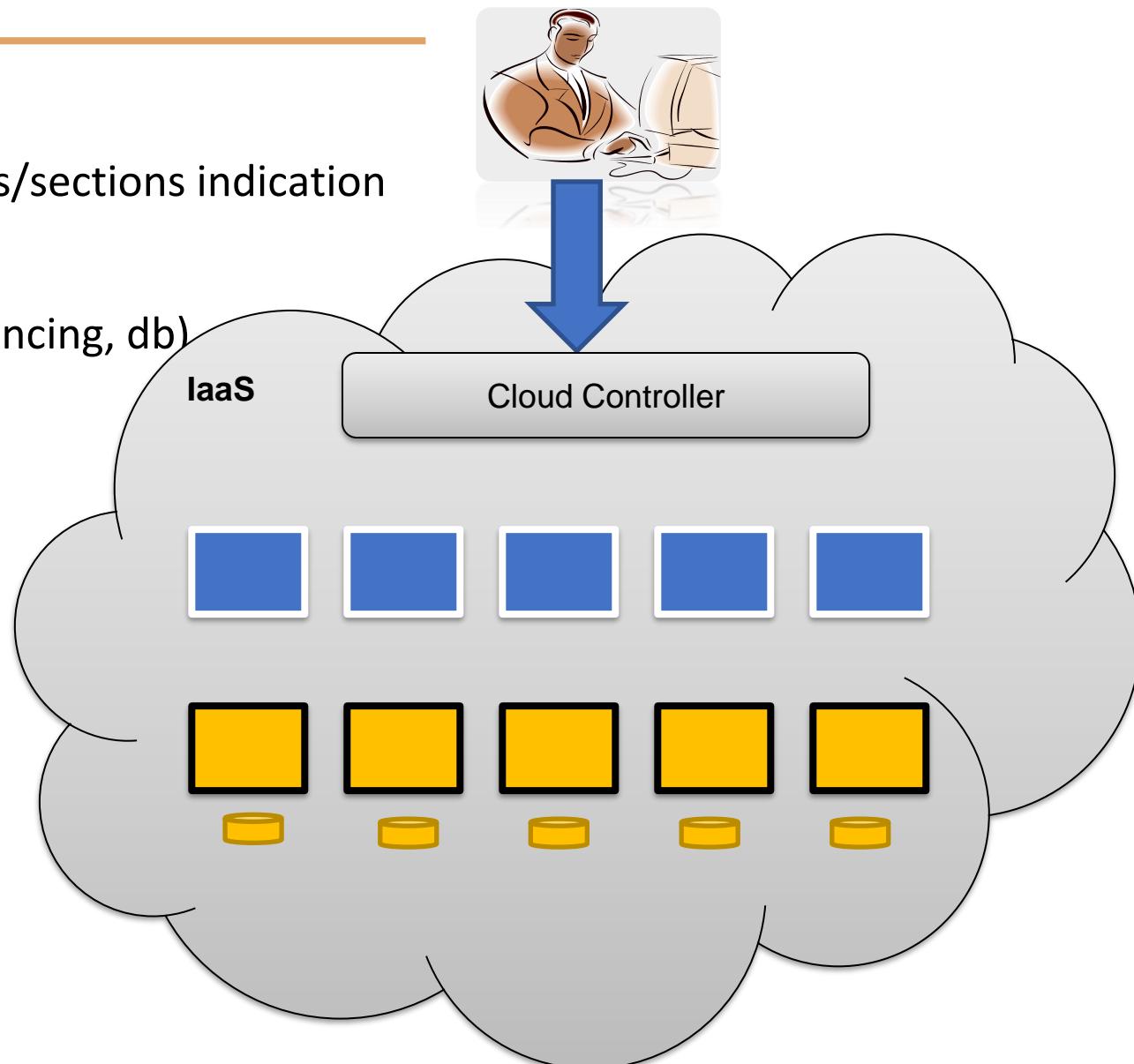


How is this built with PaaS as the Service Model

- What do we want to do?
 - User pushes into cloud
 - Application (with the different components/sections indication where it needs to done)
 - Specification of S/W (e.g., OS, Web server)
 - Services needed from cloud (e.g., load balancing, db)

How do we do this?

- Cloud controller
 - Automatically creates VMs and containers
 - Deploys applications
 - Connects to services
 - Automatically scales up or down
- Storage Services
 - Object
 - NoSQL
 - Relational
 - Block storage
- Applications store state in Storage Services
 - Simpler to scale applications
 - Easier recovery from failure



When to Use PaaS

PaaS can streamline workflows when multiple developers are working on the same development project. If other vendors must be included, PaaS can provide great speed and flexibility to the entire process. PaaS is particularly beneficial if you need to create customized applications.

API development and management: You can use PaaS to develop, run, manage, and secure application programming interfaces (APIs) and microservices.

Internet of Things (IoT): PaaS can support the broad range of application environments, programming languages, and tools used for IoT deployments.

PaaS Limitations & Concerns (Recap)

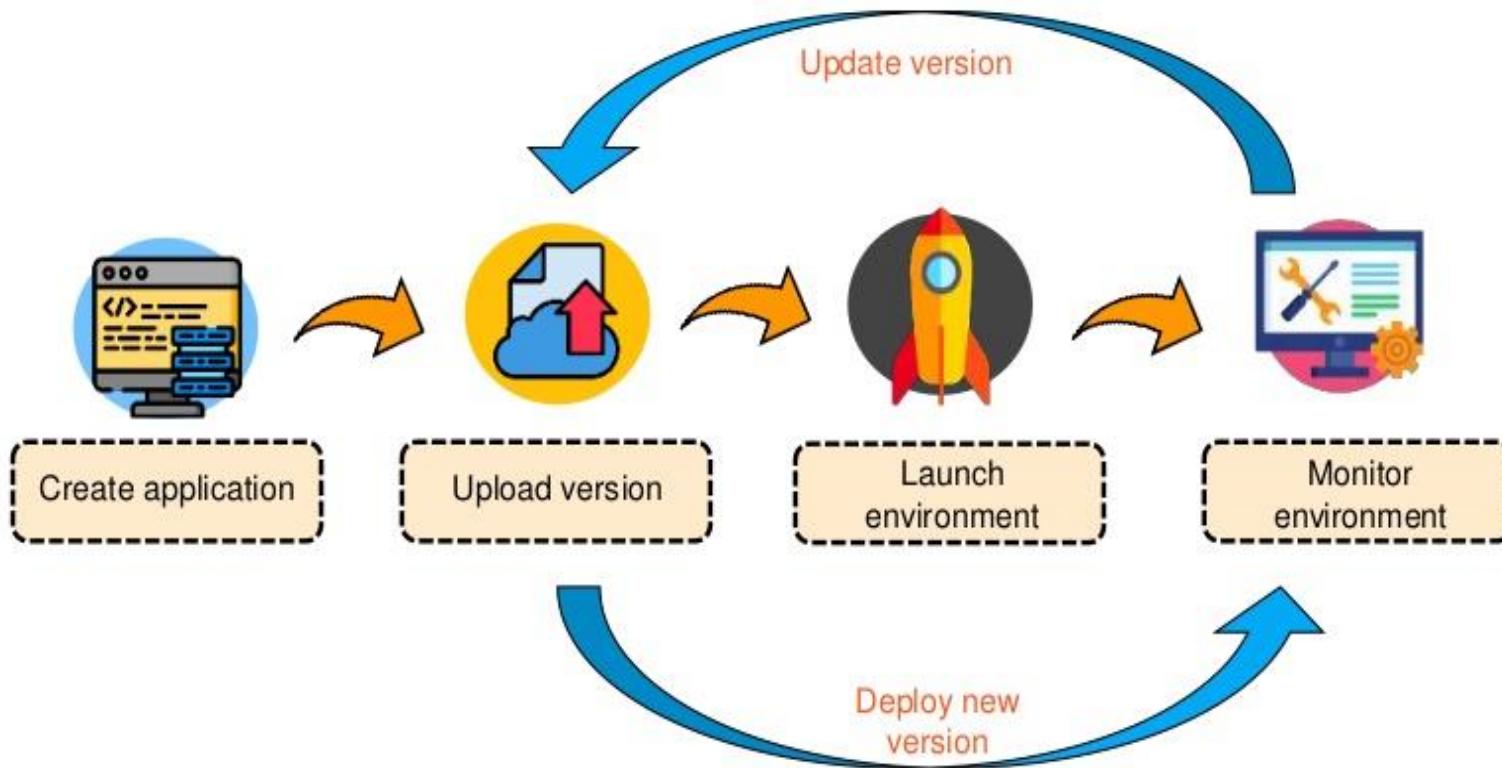
1. **Operational limitation(Lack of control):** Customized cloud operations with management automation workflows may not apply to PaaS solutions, as platform tends to limit operational capabilities for end users. Although this is intended to reduce the operational burden on end users, the loss of operational control may affect how PaaS solutions are managed, provisioned, and operated.
2. **Vendor lock-in:** Business and technical requirements that drive decisions for a specific PaaS solution may not apply in the future. If the vendor has not provisioned convenient migration policies, switching to alternative PaaS options may not be possible without affecting the business.
3. **Runtime issues:** In addition to limitations associated with specific apps and services, PaaS solutions may not be optimized for the language and frameworks of your choice. Specific framework versions may not be available or perform optimally with the PaaS service. Customers may not be able to develop custom dependencies with the platform.
4. **Data security:** Organizations can run their own apps and services using PaaS solutions, but the data residing in third-party, vendor-controlled cloud servers poses security risks and concerns. Your security options may be limited as customers may not be able to deploy services with specific hosting policies.
5. **Integrations:** The complexity of connecting the data stored within an onsite data center or off-premise cloud is increased, which may affect which apps and services can be adopted with the PaaS offering. Particularly when not every component of a legacy IT system is built for the cloud, integration with existing services and infrastructure may be a challenge.
6. **Customization of legacy systems:** PaaS may not be a plug-and-play solution for existing legacy apps and services. Instead, several customizations and configuration changes may be necessary for legacy systems to work with the PaaS service. The resulting customization can result in a complex IT system that may limit the value of the PaaS investment altogether.

- Amazon Web Services : Elastic Beanstalk.
- Microsoft Azure : Azure DevOps.
- Google Cloud : Google App Engine.

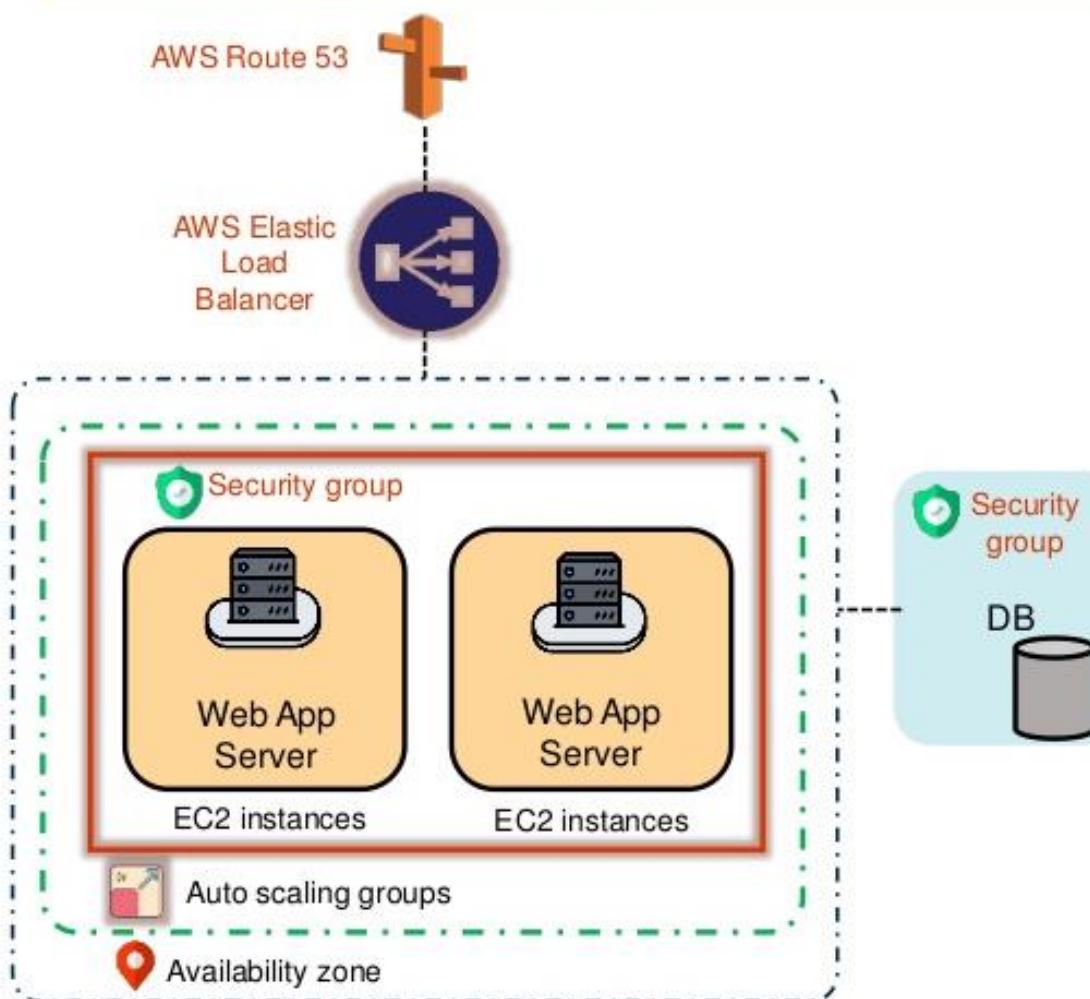
Table 4.2 Five Public Cloud Offerings of PaaS [10,18]

Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

How does Elastic Beanstalk in AWS work?



PaaS Demo : Architecture of AWS Elastic Beanstalk

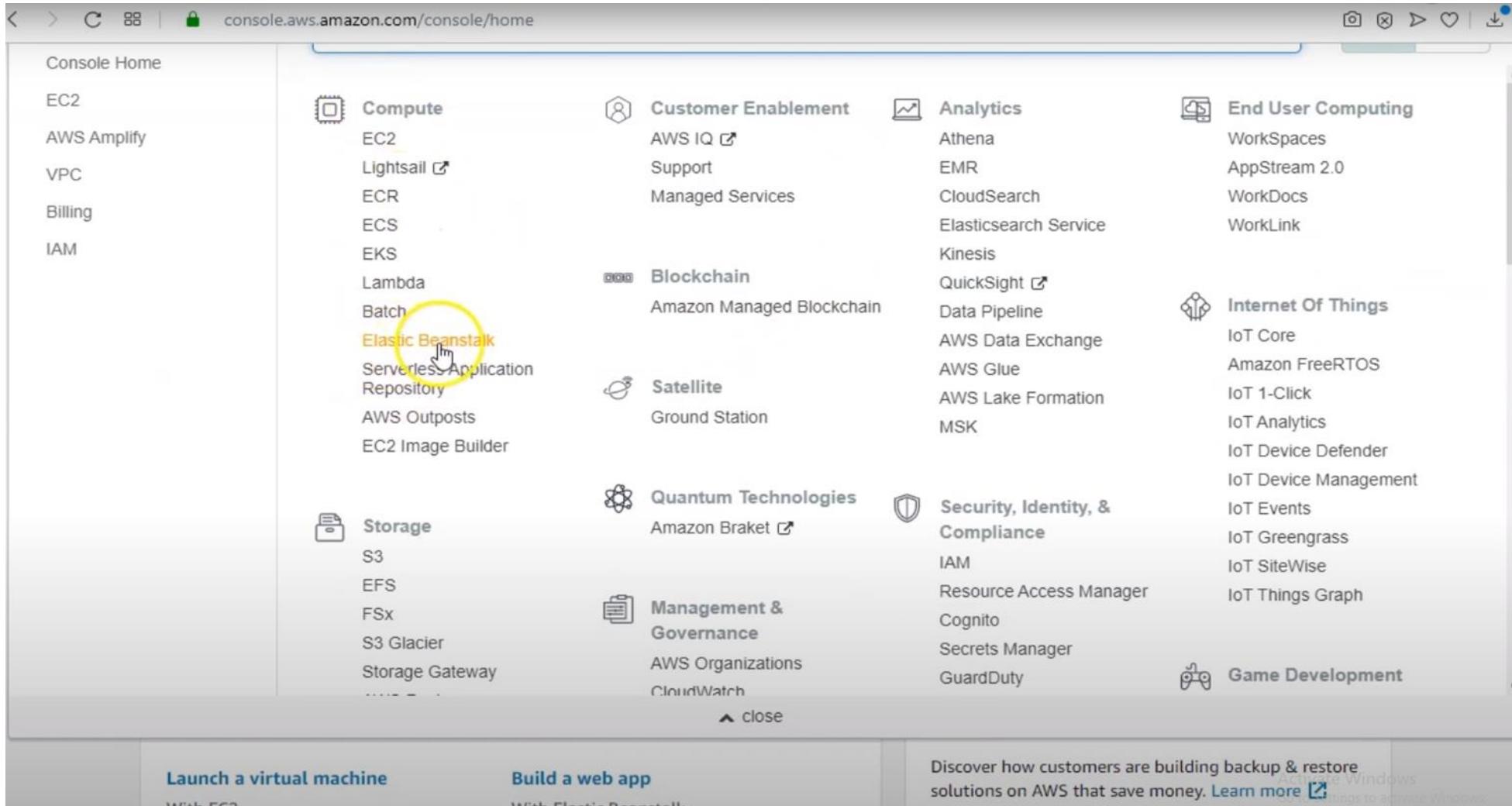


Web Server environment

- If the application receives client requests, Amazon Route53 sends these requests to the AWS Elastic Load Balancer.
- Later, AWS load balancer shares the requests among EC2 instances
- Every EC2 instances have a security group
- The load balancer is connected to Amazon EC2 instances, which are part of an Auto Scaling group

PaaS Demo : AWS Elastic Beanstalk

Step 1: Log into AWS, go to services and select Elastic Beanstalk



The screenshot shows the AWS Console Home page. On the left, there's a sidebar with links to EC2, AWS Amplify, VPC, Billing, and IAM. The main area is a grid of service icons. In the first row, 'Compute' is expanded, showing EC2, Lightsail, ECR, ECS, EKS, Lambda, Batch, and Elastic Beanstalk. 'Elastic Beanstalk' is highlighted with a yellow circle and a cursor icon pointing at it. Other services in this row include Customer Enablement (AWS IQ, Support, Managed Services), Analytics (Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis), and End User Computing (WorkSpaces, AppStream 2.0, WorkDocs, WorkLink). The second row includes Blockchain (Amazon Managed Blockchain), QuickSight, Data Pipeline, AWS Data Exchange, AWS Glue, AWS Lake Formation, and MSK. The third row includes Satellite (Ground Station), Internet Of Things (IoT Core, Amazon FreeRTOS, IoT 1-Click, IoT Analytics, IoT Device Defender, IoT Device Management, IoT Events, IoT Greengrass, IoT SiteWise, IoT Things Graph). The fourth row includes Quantum Technologies (Amazon Braket), Security, Identity, & Compliance (IAM, Resource Access Manager, Cognito, Secrets Manager, GuardDuty), and Game Development.

Console Home

EC2

AWS Amplify

VPC

Billing

IAM

Compute

Customer Enablement

Analytics

End User Computing

Blockchain

Satellite

Quantum Technologies

Management & Governance

Security, Identity, & Compliance

Game Development

Elastic Beanstalk

Serverless Application Repository

AWS Outposts

EC2 Image Builder

Storage

S3

EFS

FSx

S3 Glacier

Storage Gateway

Launch a virtual machine With EC2

Build a web app With Elastic Beanstalk

Discover how customers are building backup & restore solutions on AWS that save money. Learn more

PaaS Demo : AWS Elastic Beanstalk

Step 2: Click on Get started and then create new application

Welcome to AWS Elastic Beanstalk

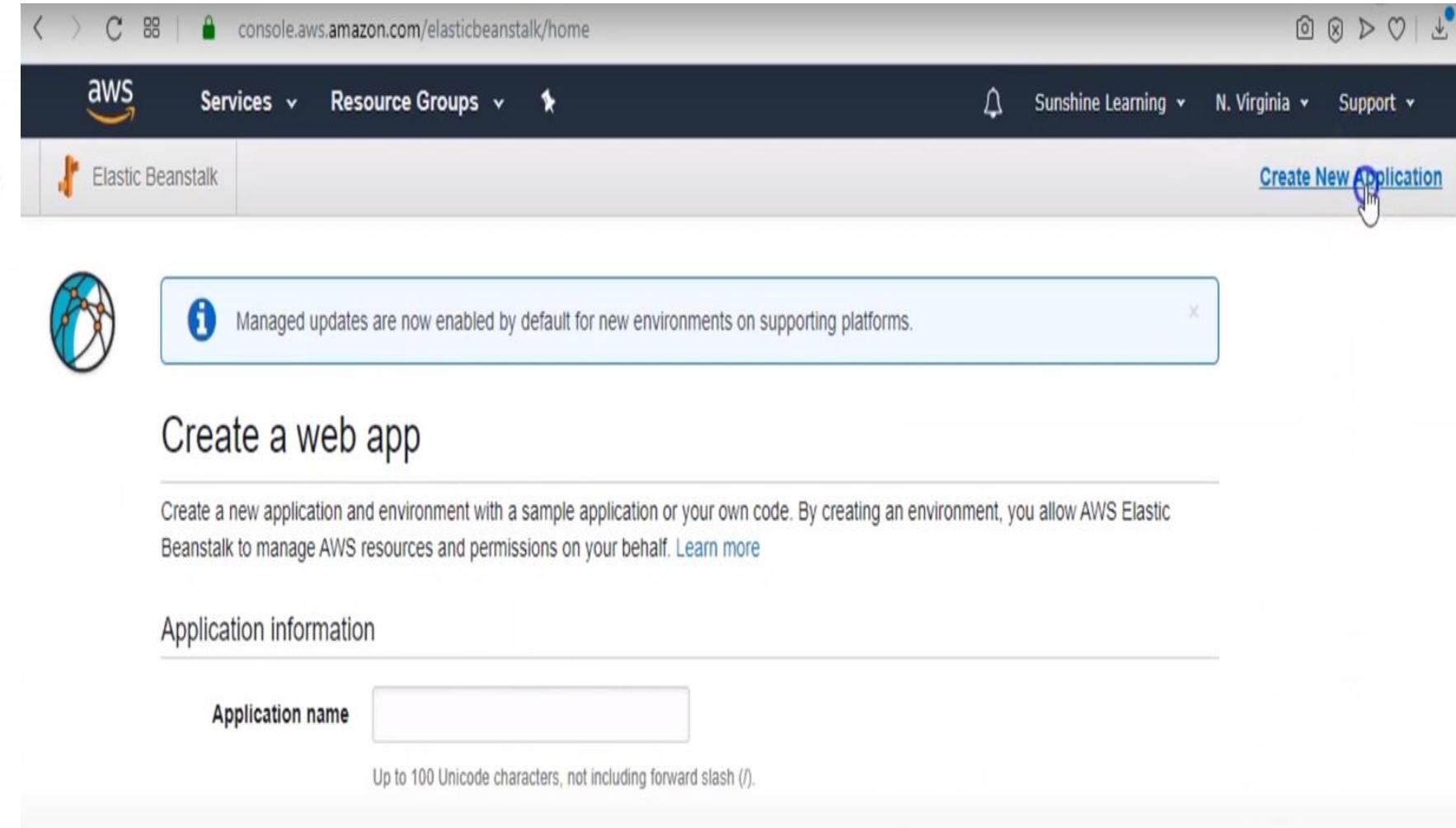
With Elastic Beanstalk, you can **deploy**, **monitor**, and **scale** an application quickly and easily. Let us do the heavy lifting so you can focus on your business.

To deploy your **existing web application**, create an **application source bundle** and then **create a new application**. If you're using **Git** and would prefer to use it with our command line tool, please see [Getting Started with the EB CLI](#).

To deploy a **sample application**, click **Get started**, choose a name, select a platform and click **Create app**.

By launching the sample application, you allow AWS Elastic Beanstalk to administer AWS resources and necessary permissions on your behalf. [Learn more](#)

Get started



console.aws.amazon.com/elasticbeanstalk/home

aws Services Resource Groups Sunshine Learning N. Virginia Support

Elastic Beanstalk Create New Application

Managed updates are now enabled by default for new environments on supporting platforms.

Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Application information

Application name

Up to 100 Unicode characters, not including forward slash (/).

PaaS Demo : AWS Elastic Beanstalk

Step 3: Enter application name, description and click on create

Create New Application

Application Name XYZ
Maximum length of 100 characters, not including forward slash (/).

Description Demo App
Maximum length of 200 characters.

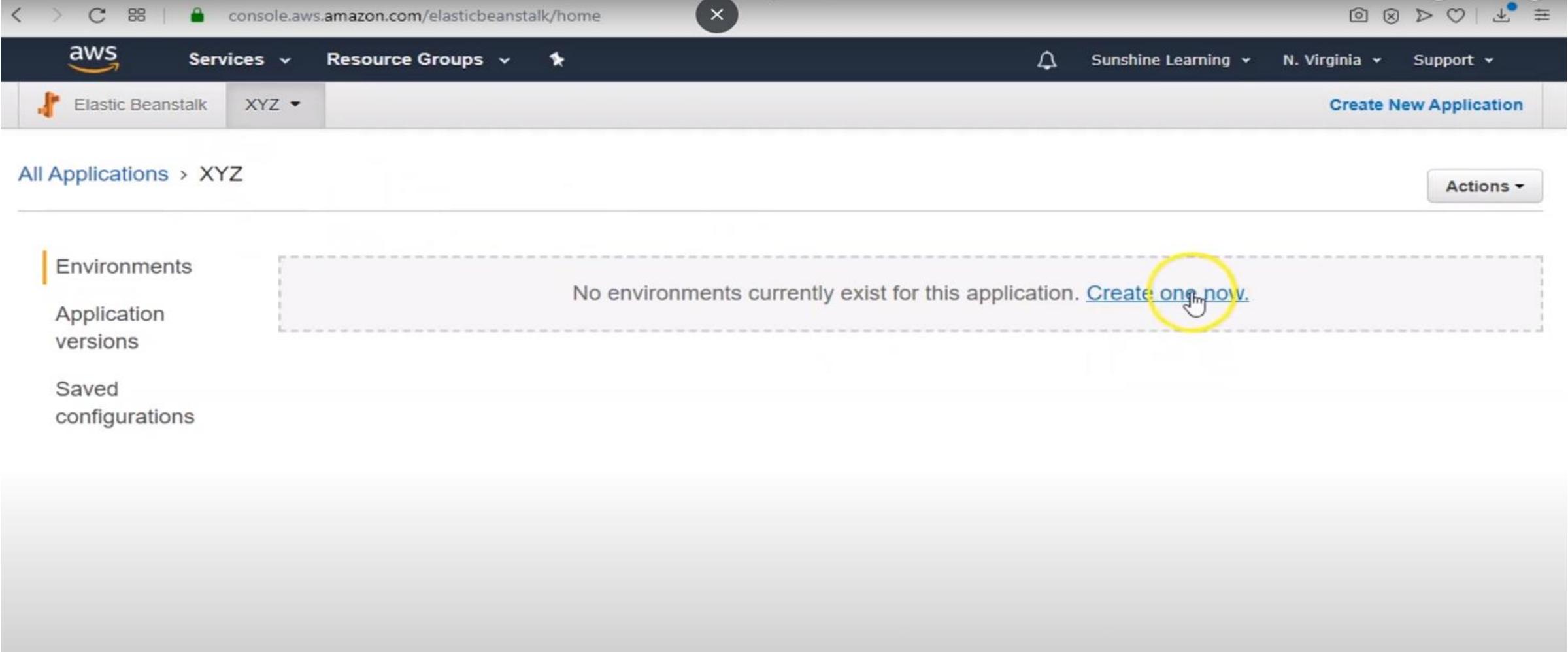
Tags
Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive.
[Learn more](#)

Key (127 characters maximum)	Value (255 characters maximum)
<input type="text"/>	<input type="text"/>

50 remaining

PaaS Demo : AWS Elastic Beanstalk

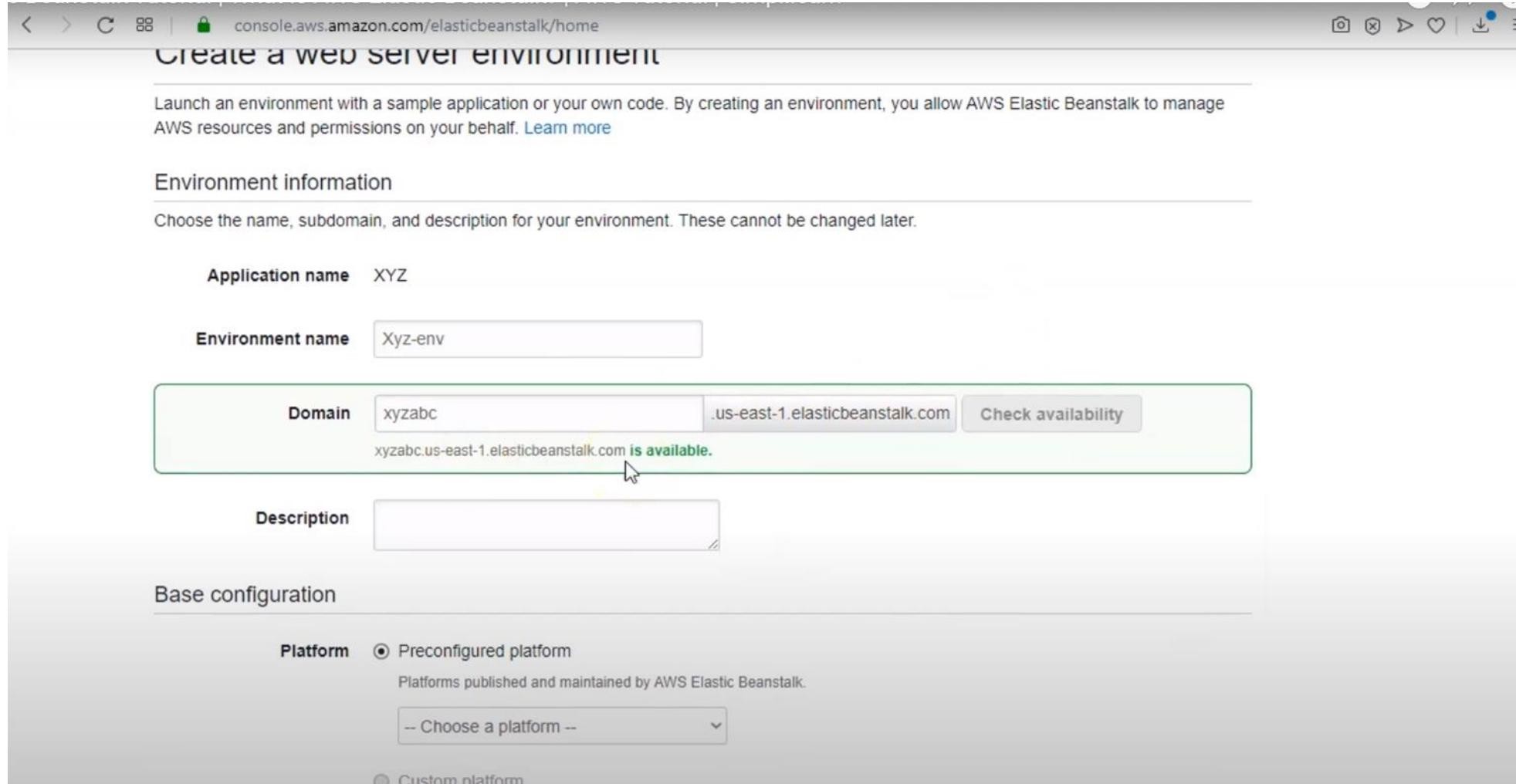
Step 4: Observer we don't have an environment for application, click on create one now.



The screenshot shows the AWS Elastic Beanstalk console interface. At the top, the URL is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, Sunshine Learning, N. Virginia, and Support. Below the navigation bar, there are tabs for 'Elastic Beanstalk' and 'XYZ'. On the right, there is a 'Create New Application' button. The main content area shows the 'XYZ' application under 'All Applications'. On the left, there is a sidebar with three options: 'Environments' (which is selected and highlighted in orange), 'Application versions', and 'Saved configurations'. The main content area displays a message: 'No environments currently exist for this application. [Create one now.](#)' A yellow circle with a cursor icon is drawn around the 'Create one now.' link.

PaaS Demo : AWS Elastic Beanstalk

Step 5: Enter the domain name for the application



console.aws.amazon.com/elasticbeanstalk/home

Create a web server environment

Launch an environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name XYZ

Environment name Xyz-env

Domain xyzabc.us-east-1.elasticbeanstalk.com [Check availability](#)

xyzabc.us-east-1.elasticbeanstalk.com **is available.**

Description

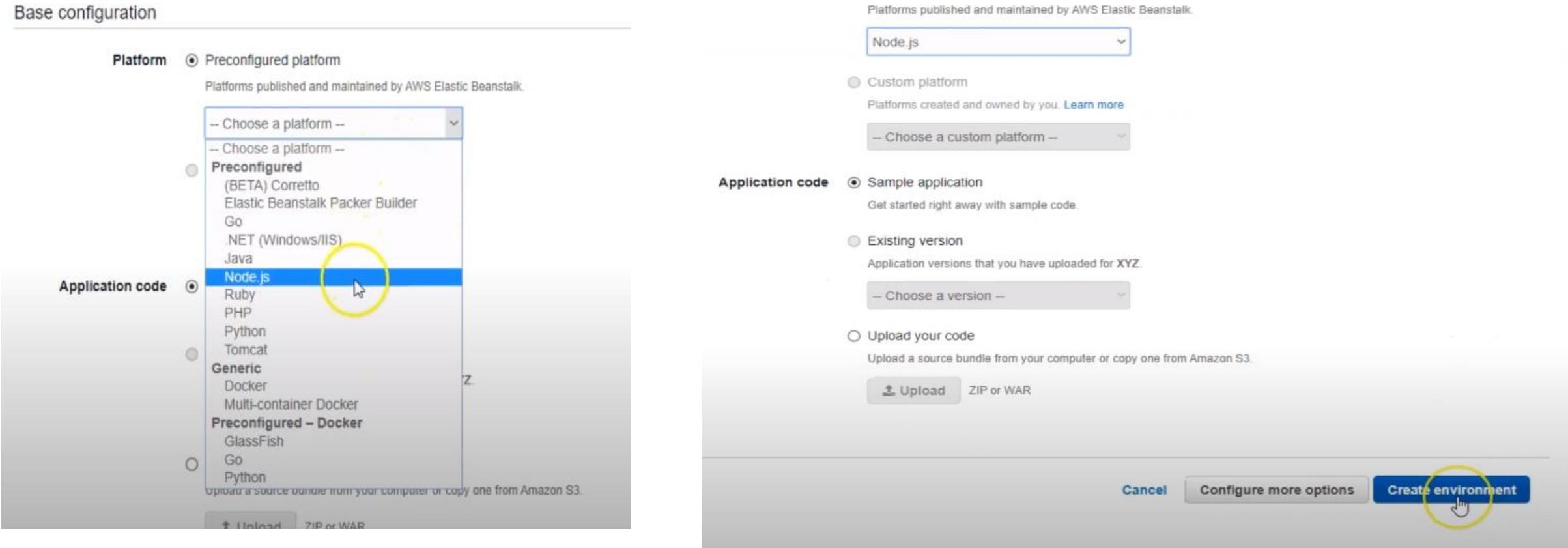
Base configuration

Platform Preconfigured platform
Platforms published and maintained by AWS Elastic Beanstalk.

Custom platform

PaaS Demo : AWS Elastic Beanstalk

Step 6: Enter the platform(Node js in this case) and click on create environment



The screenshot shows the 'Create New Environment' wizard in the AWS Elastic Beanstalk console. The first step, 'Base configuration', is displayed.

Platform: Preconfigured platform (selected). A dropdown menu shows 'Node.js' highlighted with a yellow circle and a cursor icon.

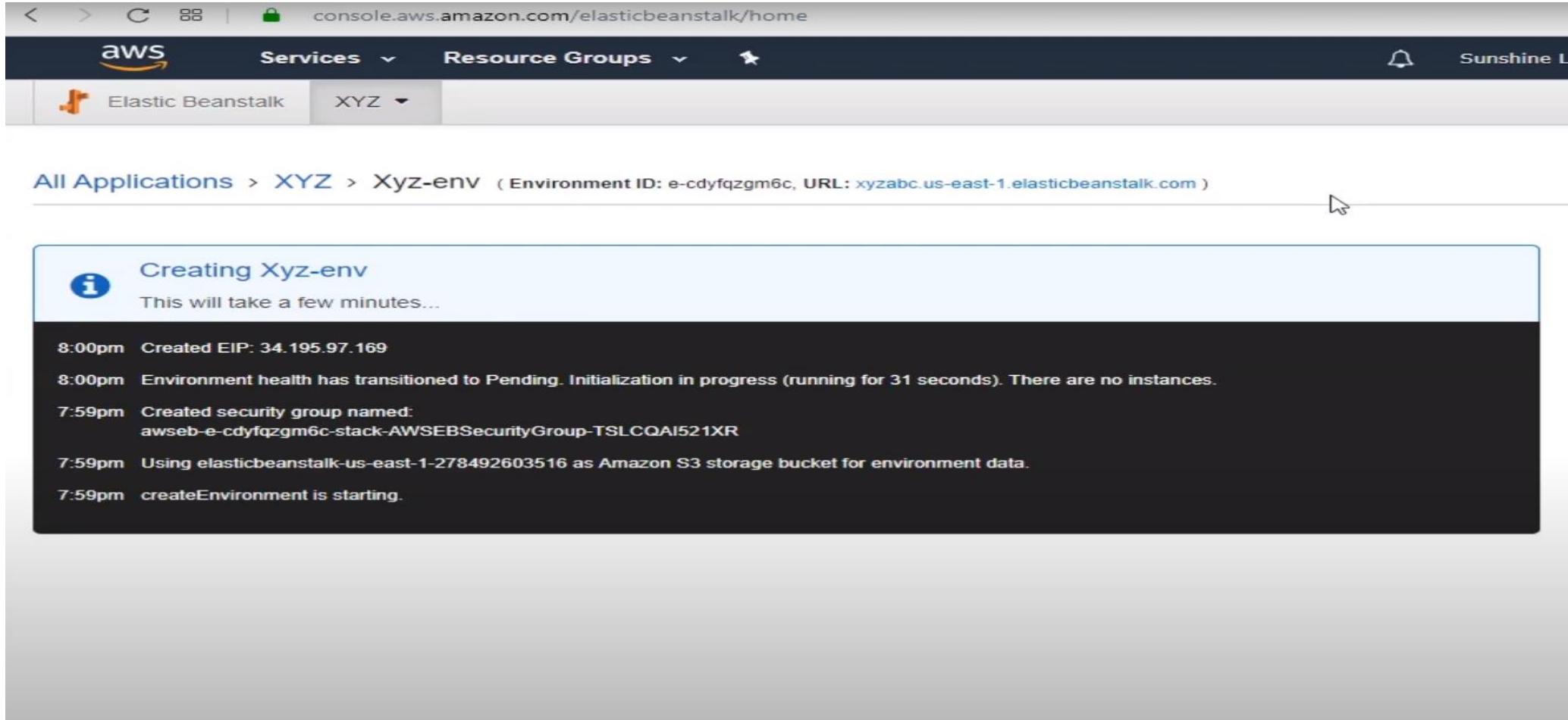
Application code: Node.js (selected). Other options include Ruby, PHP, Python, Tomcat, Generic, Docker, Multi-container Docker, and Preconfigured – Docker. A 'Upload' button is visible at the bottom left.

Platform selection: A dropdown menu titled 'Platforms published and maintained by AWS Elastic Beanstalk' shows 'Node.js' selected. Other options include 'Custom platform' (selected) and 'Choose a custom platform'.

Application code selection: A dropdown menu titled 'Application code' shows 'Sample application' selected. Other options include 'Existing version' and 'Upload your code'. A 'Create environment' button is highlighted with a yellow circle and a cursor icon.

PaaS Demo : AWS Elastic Beanstalk

Step 7: Elastic Beanstalk will create the environment and we can observe the logs on the dashboard.
Click on the application name(XYZ in this case).



The screenshot shows the AWS Elastic Beanstalk console interface. At the top, the URL is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, and a Sunshine icon. Below the navigation bar, there are tabs for **Elastic Beanstalk** and **XYZ**. The main content area displays the following information:

All Applications > XYZ > Xyz-env (Environment ID: e-cdyfqzgm6c, URL: xyzabc.us-east-1.elasticbeanstalk.com)

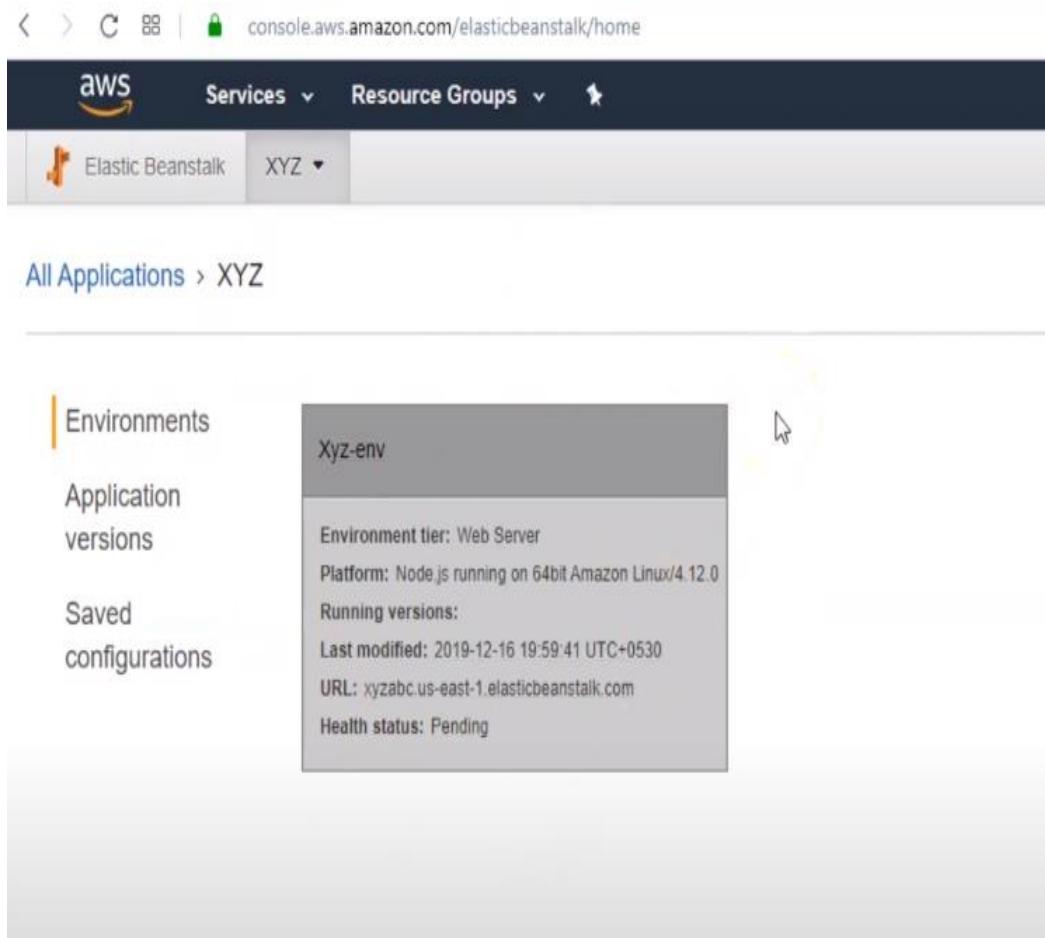
Creating Xyz-env
This will take a few minutes...

Logs (Timeline):

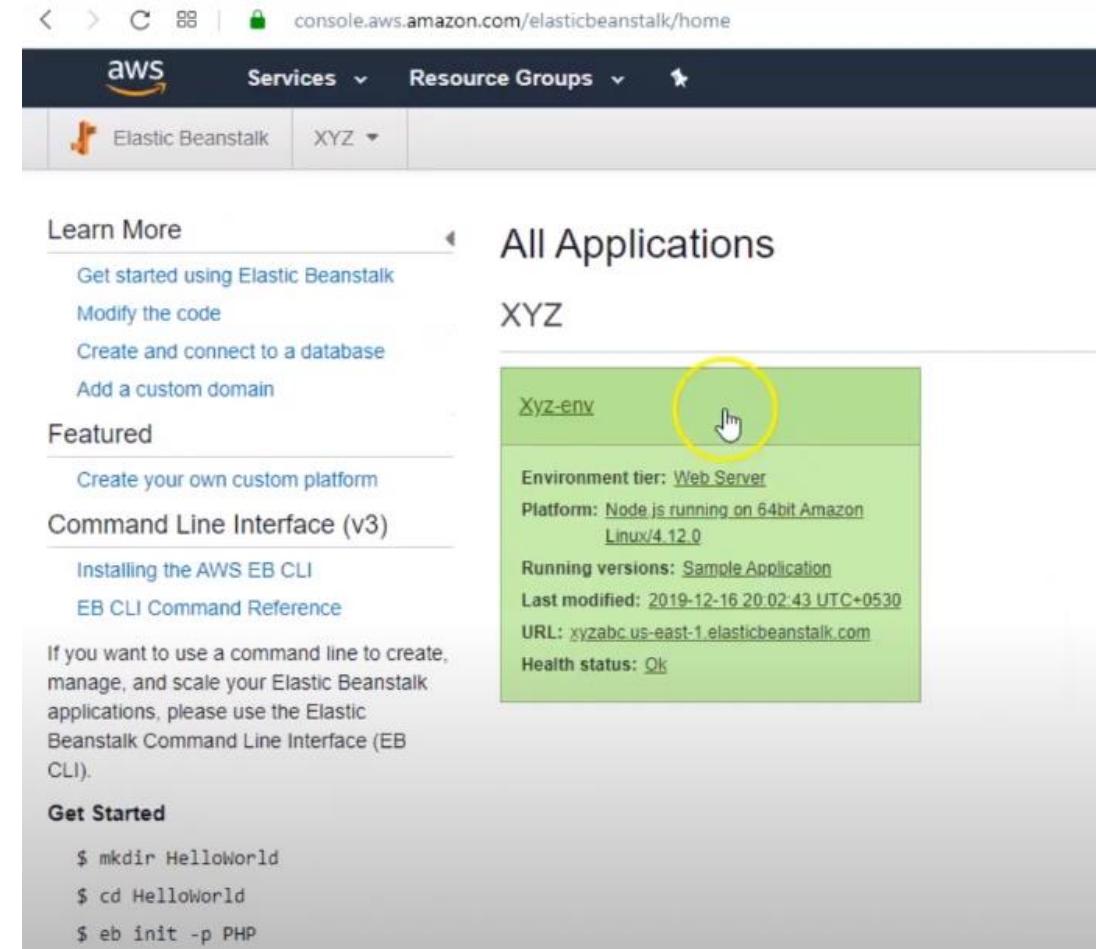
- 8:00pm Created EIP: 34.195.97.169
- 8:00pm Environment health has transitioned to Pending. Initialization in progress (running for 31 seconds). There are no instances.
- 7:59pm Created security group named:
awseb-e-cdyfqzgm6c-stack-AWSEBSecurityGroup-TSLCQAI521XR
- 7:59pm Using elasticbeanstalk-us-east-1-278492603516 as Amazon S3 storage bucket for environment data.
- 7:59pm createEnvironment is starting.

PaaS Demo : AWS Elastic Beanstalk

Step 8: Observe the colour of the box depicting the environment. Grey colour indicates the build is in progress. The green colour indicates the environment is up and running. Once the environment is ready click on it.



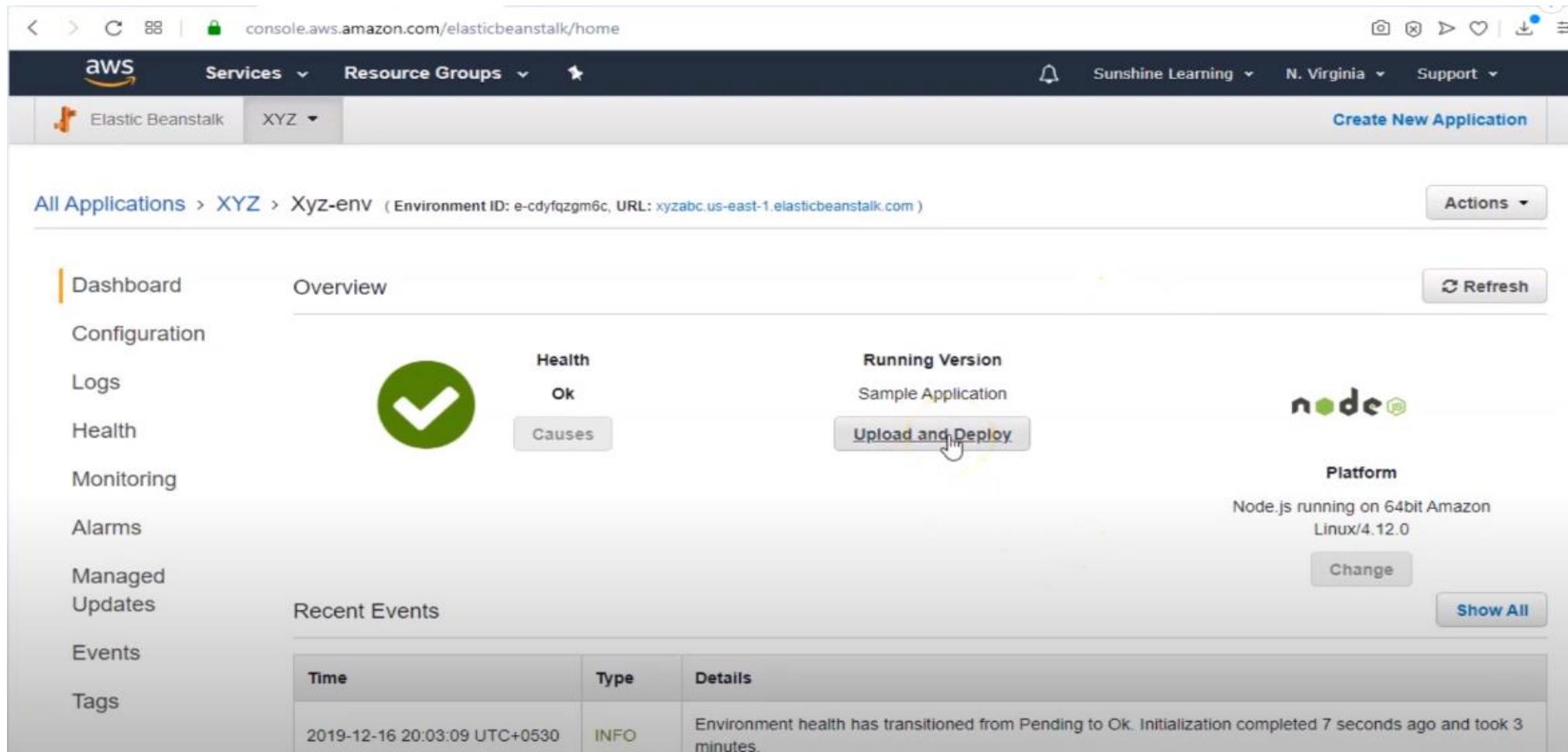
The screenshot shows the AWS Elastic Beanstalk console. The URL in the browser is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, and Resource Groups dropdown. Under the Services dropdown, 'Elastic Beanstalk' is selected. A dropdown menu shows 'XYZ' is currently selected. The main content area displays the 'All Applications > XYZ' section. On the left, there's a sidebar with 'Environments', 'Application versions', and 'Saved configurations'. The 'Environments' section lists 'Xyz-env' which is highlighted. The details for 'Xyz-env' are shown in a box: 'Environment tier: Web Server', 'Platform: Node.js running on 64bit Amazon Linux/4.12.0', 'Running versions:', 'Last modified: 2019-12-16 19:59:41 UTC+0530', 'URL: xyzabc.us-east-1.elasticbeanstalk.com', and 'Health status: Pending'. A cursor arrow is pointing towards the 'Xyz-env' box.



The screenshot shows the AWS Elastic Beanstalk console. The URL in the browser is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, and Resource Groups dropdown. Under the Services dropdown, 'Elastic Beanstalk' is selected. A dropdown menu shows 'XYZ' is currently selected. The main content area displays the 'All Applications > XYZ' section. On the right, there's a detailed view of the 'Xyz-env' environment. The environment status is listed as 'Ok'. Other details include 'Environment tier: Web Server', 'Platform: Node.js running on 64bit Amazon Linux/4.12.0', 'Running versions: Sample Application', 'Last modified: 2019-12-16 20:02:43 UTC+0530', 'URL: xyzabc.us-east-1.elasticbeanstalk.com', and 'Health status: Ok'. A yellow circle highlights the 'Xyz-env' environment name. The sidebar on the left is identical to the one in the first screenshot.

PaaS Demo : AWS Elastic Beanstalk

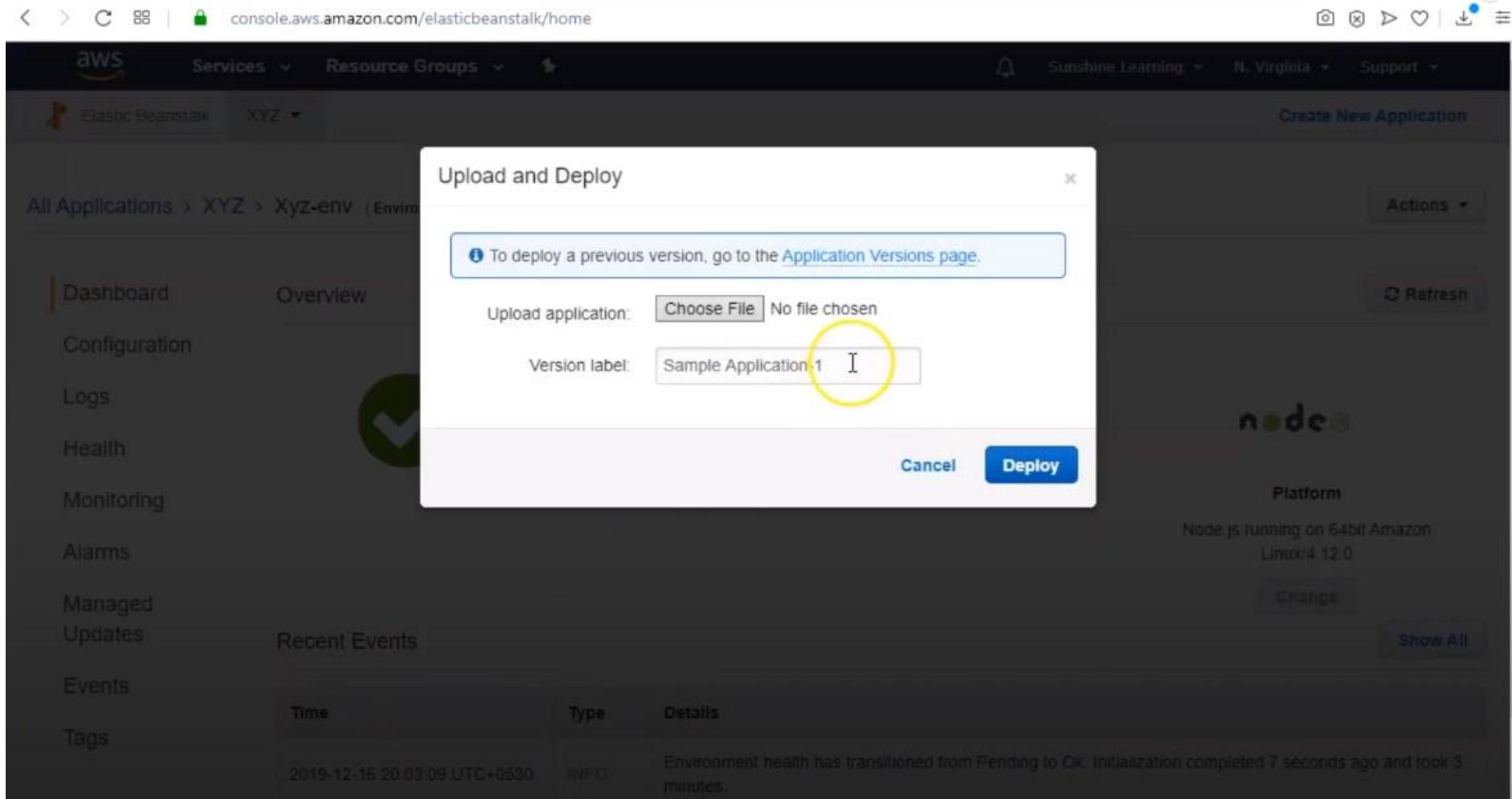
Step 9: We can see a dashboard giving an overview of the environment. Observe the health status and the selected platform. Click on Upload and Deploy



The screenshot shows the AWS Elastic Beanstalk Overview page for the 'XYZ' application environment 'Xyz-env'. The left sidebar lists navigation options: Dashboard (selected), Configuration, Logs, Health, Monitoring, Alarms, Managed Updates, Events, and Tags. The main content area displays the 'Overview' tab. It features a large green circle with a white checkmark, labeled 'Health' and 'Ok'. Below it is a 'Causes' link. To the right, there's a section for the 'Running Version' labeled 'Sample Application' with a 'node.js' logo. A large, prominent blue button labeled 'Upload and Deploy' is centered below the health status. To the right of the application details, a 'Platform' section indicates 'Node.js running on 64bit Amazon Linux/4.12.0' with a 'Change' link. At the bottom, a 'Recent Events' table shows one entry: '2019-12-16 20:03:09 UTC+0530' (INFO) - 'Environment health has transitioned from Pending to Ok. Initialization completed 7 seconds ago and took 3 minutes.'

PaaS Demo : AWS Elastic Beanstalk

Step 10: Upload your application file and deploy.



The screenshot shows the AWS Elastic Beanstalk console interface. In the center, a modal window titled "Upload and Deploy" is open. It contains a message: "To deploy a previous version, go to the [Application Versions page](#)". Below this, there is a "Upload application:" section with a "Choose File" button and a message "No file chosen". Underneath is a "Version label:" section with an input field containing "Sample Application 1", which is circled in yellow. At the bottom of the modal are "Cancel" and "Deploy" buttons. The background shows the main Elastic Beanstalk dashboard for the "XYZ" application environment, featuring sections for Dashboard, Configuration, Logs, Health, Monitoring, Alarms, Managed Updates, and Events. The "Recent Events" table shows one entry: "Environment health has transitioned from Pending to Ok. Initialization completed 7 seconds ago and took 3 minutes." The platform listed is "Node.js running on 64bit Amazon Linux/4.12.0".



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

COMMUNICATION USING MESSAGE QUEUES

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

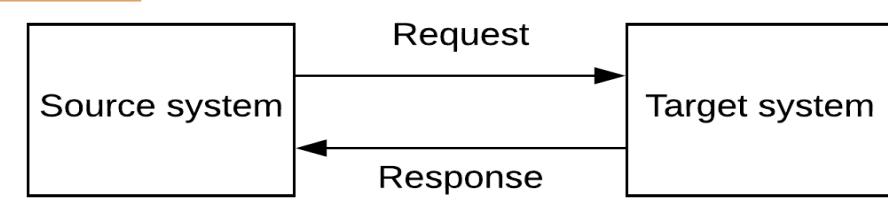
- In a distributed computing environment like the Cloud, where the hardware infrastructure typically configured as a cluster with different system architectures like a client-server or master-slave or say a P2P architecture, there exists a significant interactions between the components or machines involving data flows from one system to another which could be through messages or events.
- These data flows could happen using different protocols such as HTTP, AMQP (Advanced Message Queuing Protocol, or a binary protocol like TCP and with different data formats.
- There are also different styles or natures through which these communication may happen like **synchronous** or **asynchronous** or based on the **request** and **number of processors** of the service
- If the application is a monolith with multiple processes, simple intra-system IPC mechanism may work to communicate with processes which need to interact, but when built as a set of say microservices maybe running on different systems, communication mechanisms which can support Inter-service communication would be essential

CLOUD COMPUTING

Interaction styles

1. Synchronous (request-response) :

- Synchronous messaging means that the system which is sending the message expects an immediate response from the target system.
- Source system sends a message (request) to the target system and waits (blocks) until it receives a response from the target system.
- Target system may process the message within itself or send the message to another system and generate a response within a short time period.
- To make this communication successful and make sure source system does not waste its resources in case of a target system failure, there is a timeout configured at the source side so that it will stop waiting for the response if the timeout is elapsed without receiving a response from the target system.
- There are variants of this like with call-back and full-duplex .. which you could go through



Source system waits(blocks) for the response from target system

2. Asynchronous :

- The client doesn't block, and the response, if any, isn't necessarily sent immediately
- This supports high rates of data flow.
- The expectations of these type of messages are
 - Guaranteed delivery
 - Extensive processing
 - Correlation and time series analysis
 - Decoupling of source and target systems
- These could be of the types
 - Publish Subscribe (based on topics)
 - Message Queues
 - *Event based real time processing (not planned for discussion)*
 - *Batch Processing*
 - *Store and Forward*

Asynchronous messaging : Advantages

- **Reduced coupling:** The message sender does not need to know about the consumer.
- **Multiple subscribers:** Using a pub/sub model, multiple consumers can subscribe to receive events.
- **Failure isolation:**
 - If the consumer fails, the sender can still send messages.
 - The messages will be picked up when the consumer recovers.
 - Asynchronous messaging can handle intermittent downtime. Synchronous APIs, on the other hand, require the downstream service to be available or the operation fails.
- **Load leveling:** A queue can act as a buffer to level the workload, so that receivers can process messages at their own rate.

Asynchronous messaging : Disadvantages

Coupling with the messaging infrastructure: Using a particular messaging infrastructure may cause tight coupling with that infrastructure. It will be difficult to switch to another messaging infrastructure later.

- **Latency:** End-to-end latency for an operation may become high if the message queues fill up.
- **Complexity:** Handling asynchronous messaging is not a trivial task. For example, handling of duplicated messages, or correlating request and response messages using a separate response queue.
- **Throughput:** If message queues are used, each message requires at least one queue operation and one dequeue operation. Moreover, queue semantics generally require some kind of locking inside the messaging infrastructure.

Interaction styles (Synchronous or Asynchronous)

	one-to-one	one-to-many
Synchronous	Request/response	—
Asynchronous	Asynchronous request/response One-way notifications	Publish/subscribe Publish/async responses

One-to-one interaction : Each client request is processed by exactly one service

The following are the different types of one-to-one interactions

1. ***Synchronous Request/response*** — A service client makes a request to a service and waits for a response. The client expects the response to arrive in a timely fashion. It might even block while waiting. This is an interaction style that generally results in services being tightly coupled.
2. ***Asynchronous Request/response*** — A service client sends a request to a service, which replies asynchronously. The client doesn't block while waiting, because the service might not send the response for a long time.
3. ***One-way notifications*** — A service client sends a request to a service, but no reply is expected or sent.

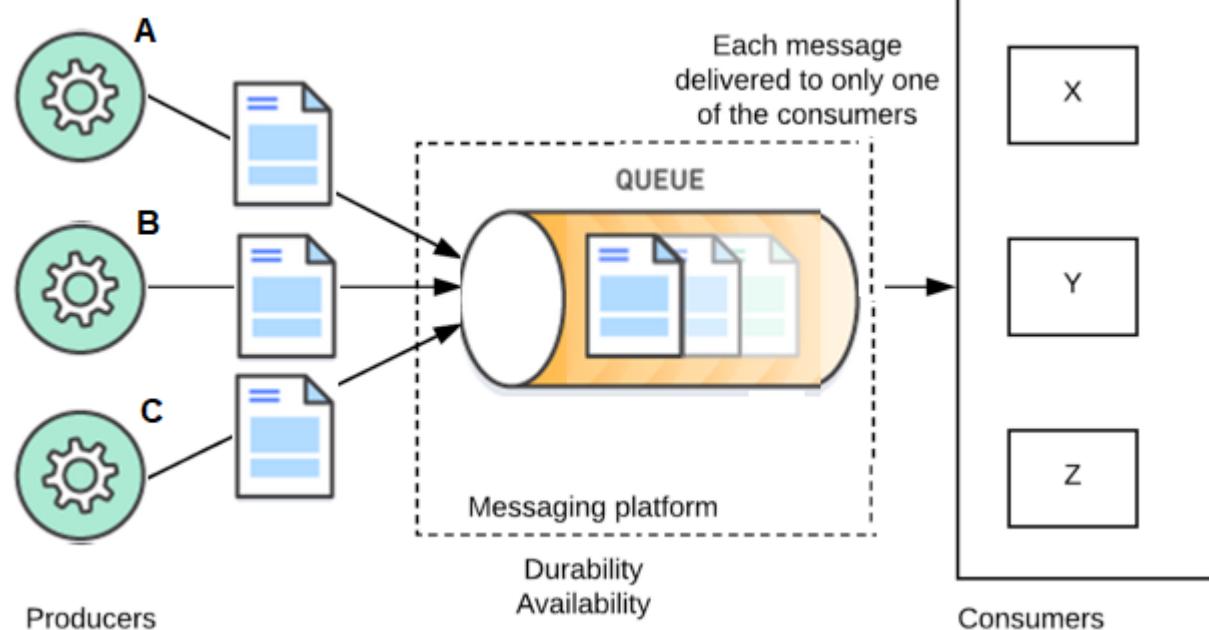
One-to-many interaction : Each request is processed by multiple services

The following are the different types of one-to-one interactions

1. ***Publish/subscribe***— A client publishes a notification message, which is consumed by zero or more interested services.
2. ***Publish/async responses***— A client publishes a request message and then waits for a certain amount of time for responses from interested services.

- A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures.
- Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer. Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.

Examples - Apache ActiveMQ, RabbitMQ



- A message queue provides a lightweight buffer which temporarily stores messages, and endpoints that allow software components to connect to the queue in order to send and receive messages.
- The messages are usually small, and can be things like requests, replies, error messages, or just plain information.
- To send a message, a component called a producer adds a message to the queue. The message is stored on the queue until another component called a consumer retrieves the message and does something with it.
- Many producers and consumers can use the queue, but each message is processed only once, by a single consumer. For this reason, this messaging pattern is often called one-to-one, or point-to-point, communications.
- When a message needs to be processed by more than one consumer, message queues can be combined with Pub/Sub messaging in a fanout design pattern.

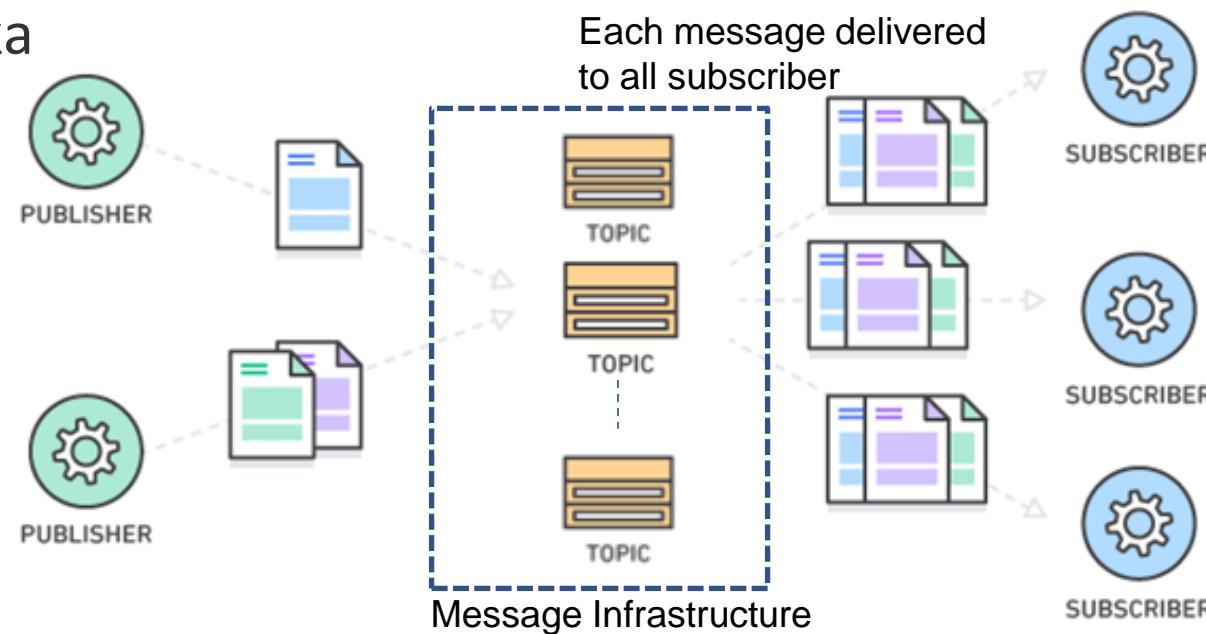
Asynchronous Communication : Publish-Subscribe

Publish/subscribe messaging is another asynchronous service-to-service communication mechanism

In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic.

Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability.

Eg. Apache Kafka



Four core concepts make up the pub/sub model:

- 1. Topic** – An intermediary channel that maintains a list of subscribers to relay messages to that are received from publishers

There can be different topic channels & different subscribers can look at their interested topics

- 1. Message** – Serialized messages sent to a topic by a publisher, has no knowledge of the subscribers
- 2. Publisher** – The application that publishes a message to a topic
- 3. Subscriber** – An application that registers itself with the desired topic in order to receive the appropriate messages

Advantages

1. Loosing coupling

Publishers are never aware of the existence of subscribers so that both systems can operate independently of each other. This methodology removes service dependencies that are present in traditional coupling.

For example, a client generally cannot send a message to a server if the server process is not running. With pub/sub, the client is no longer concerned whether or not processes are running on the server.

2. Scalability

Pub/sub messaging can scale to volumes beyond the capability of a single traditional data center. This level of scalability is primarily due to parallel operations, message caching, tree-based routing, and multiple other features built into the pub/sub model.

Advantages (Cont.)

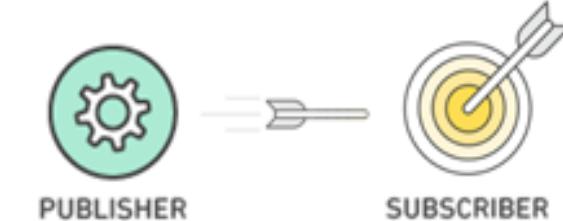
3. Eliminate Polling:

- Message topics allow instantaneous, push-based delivery, eliminating the need for message consumers to periodically check or “poll” for new information and updates.
- This promotes faster response time and reduces the delivery latency that can be particularly problematic in systems where delays cannot be tolerated.



4. Dynamic Targeting

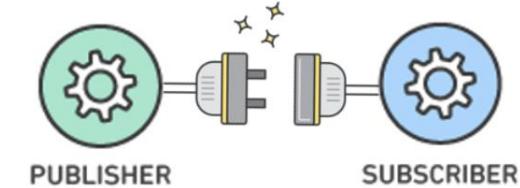
- Instead of maintaining a roster of peers that an application can send messages to, a publisher will simply post messages to a topic.
- Then, any interested party will subscribe its endpoint to the topic, and start receiving these messages. Subscribers can change, upgrade, multiply or disappear and the system dynamically adjusts.



Advantages (Cont.)

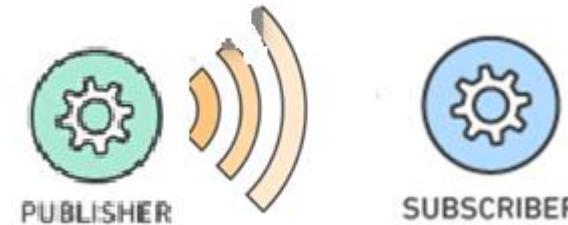
5. Decouple and Scale Independently:

- Publishers and subscribers are decoupled and work independently from each other, which allows you to develop and scale them independently.
- Adding or changing functionality won't send ripple effects across the system, because Pub/Sub allows you to flex how everything uses everything else.



6. Simplify Communication

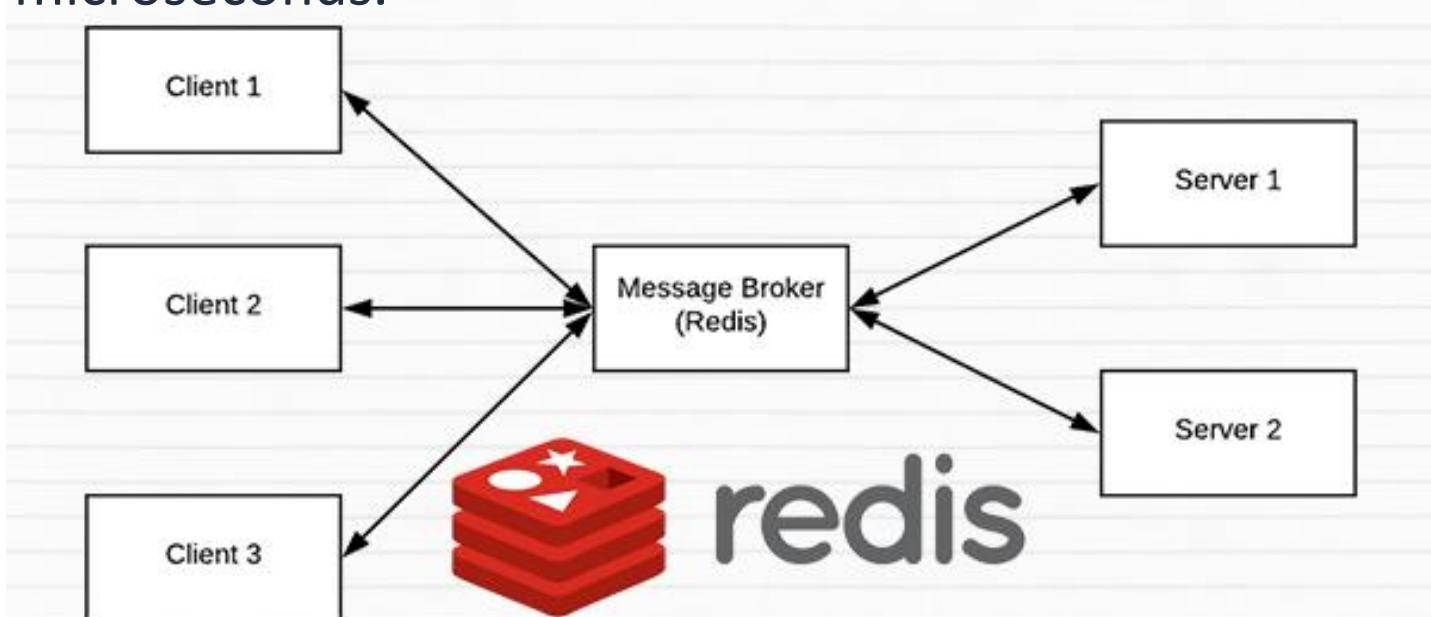
- Communications and integration code is some of the hardest code to write. The Publish Subscribe model reduces complexity by removing all the point-to-point connections with a single connection to a message topic, which will manage subscriptions to decide what messages should be delivered to which endpoints. Fewer callbacks results in looser coupling and code that's easier to maintain and extend.



Redis, which stands for **Remote Dictionary Server**, is a fast, open-source, in-memory key-value data store for use as a database, cache, **message broker**, and **queue**.

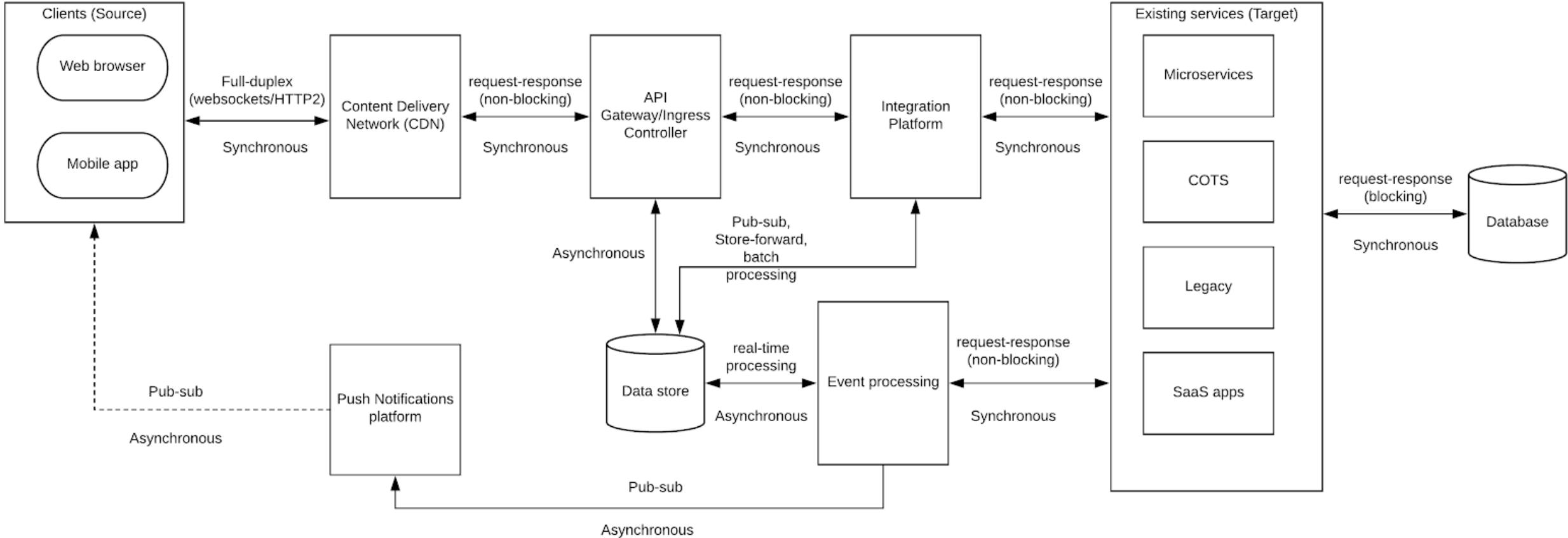
Why use Redis?

- All Redis data resides in-memory, in contrast to databases that store data on disk or SSDs. By eliminating the need to access disks, in-memory data stores such as Redis avoid seek time delays and can access data in microseconds.



CLOUD COMPUTING

A Distributed System with Messaging at Focus





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

SaaS Programming Model

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

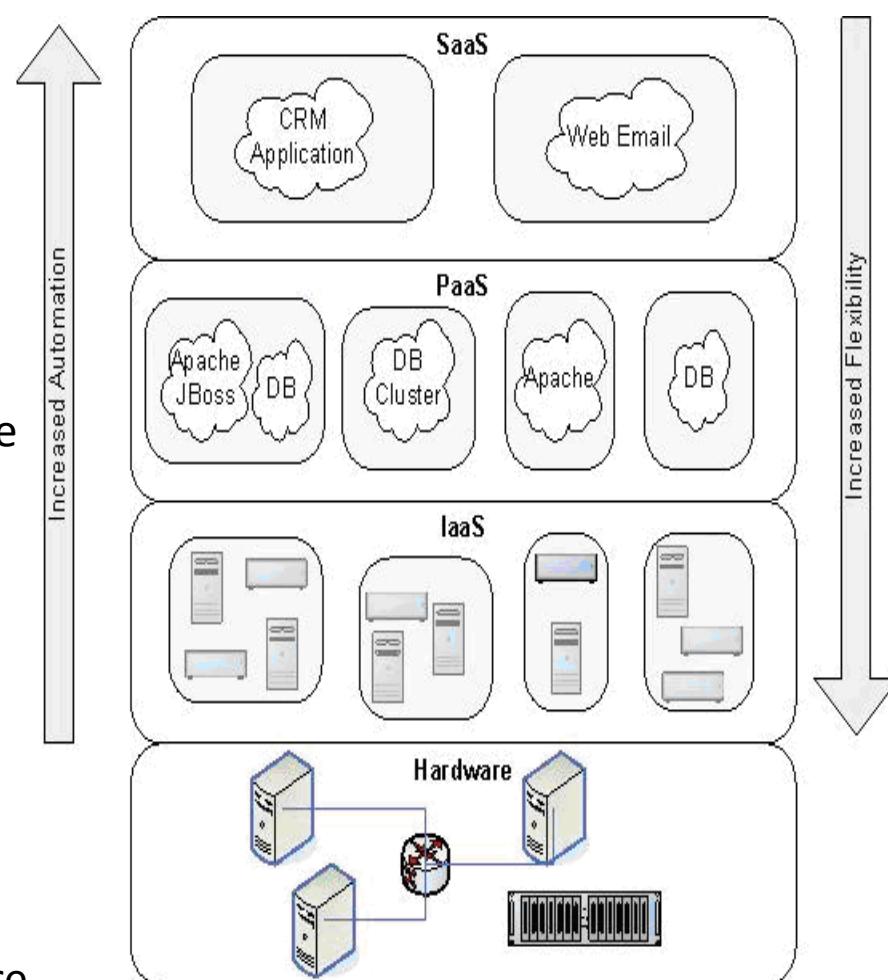
Software as a Service (Recap)

What is SaaS?

Software as a service (or SaaS) is a way of delivering applications as a service over the Internet. Instead of installing and maintaining software, you simply access it via the Internet from any thin client, freeing yourself from complex software and hardware management.

SaaS functioning

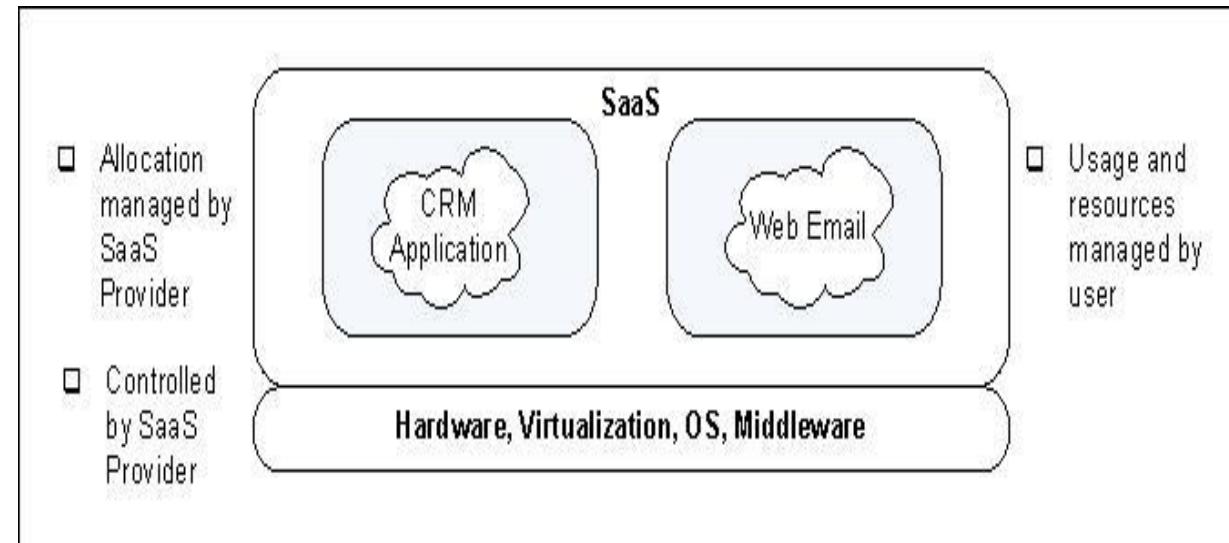
- Typically, a cloud service provider (like AWS, Azure, or IBM Cloud) manages the cloud environment on which the software is hosted.
- Users interact with the software through a web browser on their computer or mobile devices or they may use application programming interfaces (APIs) like REST to connect the software to other functions.
- SaaS applications take advantage of multitenant architecture to make use of pooled resources.
- Software updates, bug fixes, and other general app maintenance are taken care of by the SaaS provider.



Nearly every software that runs in your browser and is targeted towards the end user can be categorized as a Software as a Service product.

Popular SaaS Applications

1. Office apps - Google Docs, Sheets, Office 365
2. Email client - Gmail, Outlook
3. File Storage - DropBox, OneDrive
4. Social Media - Facebook, Snapchat
5. Customer Relationship Management - Salesforce
6. Customer support - Zendesk



1. Multi-tenant Architecture

- A multi-tenant architecture, in which all users and applications share a single, common infrastructure and code base that is centrally maintained.
- SaaS providers can make upgrades more often, with less customer risk and much lower adoption cost.
- The nature of SaaS makes it easier for providers to roll out new versions of software and features to their customers and reduce time previously spent on maintaining numerous versions of outdated code.

2. Easy Customization

- The ability for each user to easily customize applications to fit their business processes without affecting the common infrastructure.
- These customizations are unique to each company or user and are always preserved through upgrades. That means. Most SaaS applications are preconfigured plug-and-play products where the SaaS provider manages everything behind the app, including:

Lower up-front costs

Eliminate the need for additional hardware and middleware.
Reduce installation and implementation costs.
Validate and correct errors before making updates to your master data.

Predictable ongoing costs

Eliminate unpredictable costs of managing, patching, and updating software and hardware.
Turn capital expenses into operational expenses.
Reduce risk with experts managing software and overseeing cloud security.

Rapid deployment

Get up and running in hours instead of months.
Turn on and use the latest innovations and updates.
Automated software patching.

On-demand scalability.

Scale instantly to meet growing data or transactional demands.
Reduce disruptions while maintaining service levels.

Subscription

- Pay only monthly rental

Tiered Pricing

- Basic, Moderate and Superuser

Usage based pricing

- Pay for how much you are using, #API calls made

Per (Active) user pricing

- Pay for #users (or active users)

Per Feature pricing

- Each feature is priced separately

Ad-based revenue (pay per click)

- Every time there is a click, you pay

1. Security

Data is stored in the cloud, so security may be an issue for some users. There are number of approaches taken by Service providers to address this, and it could be as secure or more than in-house deployment.

2. Latency issue

Since data and applications are stored in the cloud at a variable distance from the end-user, there is a possibility that there may be greater latency when interacting with the application compared to local deployment.

Therefore, the SaaS model may not be suitable for applications whose demand response time is in milliseconds.

3. Dependency on Internet

Without an internet connection, most SaaS applications are not usable.

4. Switching between SaaS vendors is difficult

Switching SaaS vendors involves the difficult and slow task of transferring the very large data files over the internet and then converting and importing them into another SaaS.

In a SaaS Architecture, typically you will see

- Front-end or GUI for SaaS application is a browser
- Business Logic runs on the backend cloud infrastructure
- Business Logic frequently implemented as *microservices*

Examples: Google Docs, Gmail, Salesforce.com

Example of creating a SaaS application : Consider building an application say for a Book Mart which lets you browse, search, select and order, pay and maybe request for books.

Activities for the same would be

1. This SaaS application will need an say web-browser based GUI for the application (needs to factor in how the GUI screen would looks like, what technology to use ..)
2. Identify the features which would need to built into the system (authentication, listing of books, search, shopping cart, payment, refresh inventory ..)
3. Identify the workflows, customization of order of display of books .. which could happen in Backend
4. Identify and list say the features which could be built as microservices as a RESTful API for this application (Login, List books, Search, Shopping cart, purchase ..)
5. Identify those non-functional requirements (like availability, scaling, multi-tenancy, security ...) needed

Application Architectures - Monolithic

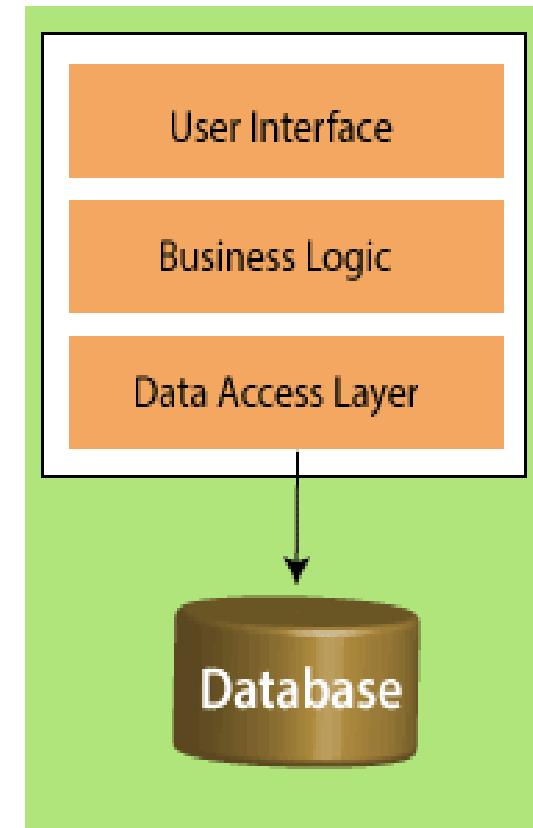
An application architecture describes the approach or patterns and techniques which are used to design and build an application.

Monolithic Architecture

- Monolith Architecture is the approach of building the application wherein all the features of the system are put on a single codebase and are typically with a single database
- In a monolith, all the components share the same resources and memory space
- It is considered to be a traditional way of building applications.
- A monolithic application is built as a single and indivisible unit. This could be visualized as in the figure with a client-side UI, Business logic and data access layer together with a database. If distributed you could have Server side application and a database access there.
- They can be characterized by being large, having long release cycles and having large teams

Typical Challenges

- Changes are not easily integrated (whole system may need to be rebuilt)
- Scalability challenges
- New Technology Adoption
- Difficult to adapt to newer practices, devices
- Reliability can be a question



Monolithic Architecture

Strengths of Monolithic Architecture

1. Development is quite simple.
2. Testing is very simple - Just launch the application and start end-to-end testing. We can also do test automation using Selenium without any difficulty.
3. Deploying the monolithic application is straightforward - Just copy the packaged application to the server.
4. Scalability is simple - We only need to have a new instance of the monolithic application and ask the load balancer to distribute load to the new instance as well. However, as the monolithic application grows in size, scalability becomes a serious issue.

Monolithic architecture worked successfully for many decades. In fact, many of the most successful and largest applications were initially developed and deployed as a monolith.

But serious issues like flexibility, reliability and scalability have led to the emergence of microservices architecture.

What are Microservices

Microservices are small services that work together but are independent, and are components or processes of an application. They have benefits of dynamic scalability, Reliability etc. as discussed later.

What is an Microservice Architecture

" The microservice is an architectural approach of developing a single application as a suite of small collaborating services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable by fully automated deployment machinery.

As an architectural framework, microservices are distributed and loosely coupled, so one team's changes won't break the entire app.

There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. "

- Martin Fowler

Benefits of Microservices Architecture

Benefits

1. Flexibility:

Microservices architecture is quite flexible. Different microservices can be developed in different technologies. Since a microservice is smaller, the code base is quite less, so it's not that difficult to upgrade the technology stack versions. Also, we can incrementally adopt a newer technology without much difficulty.

2. Reliability:

Microservices architecture can be very reliable. If one feature goes down, the entire application doesn't go down. We can fix the issue in the corresponding microservice and immediately deploy it. This makes the application to be more resilient.

3. Development speed:

Development is pretty fast in microservices architecture. Since the volume of code is much less for a microservice, it's not difficult for new team members to understand and modify the code. They become productive right from the start. Code quality is maintained well. The IDE is much faster. A microservice takes much less time to start up. All these factors considerably increase developers' productivity.

Benefits of Microservices Architecture (contd.)

4. Building complex applications:

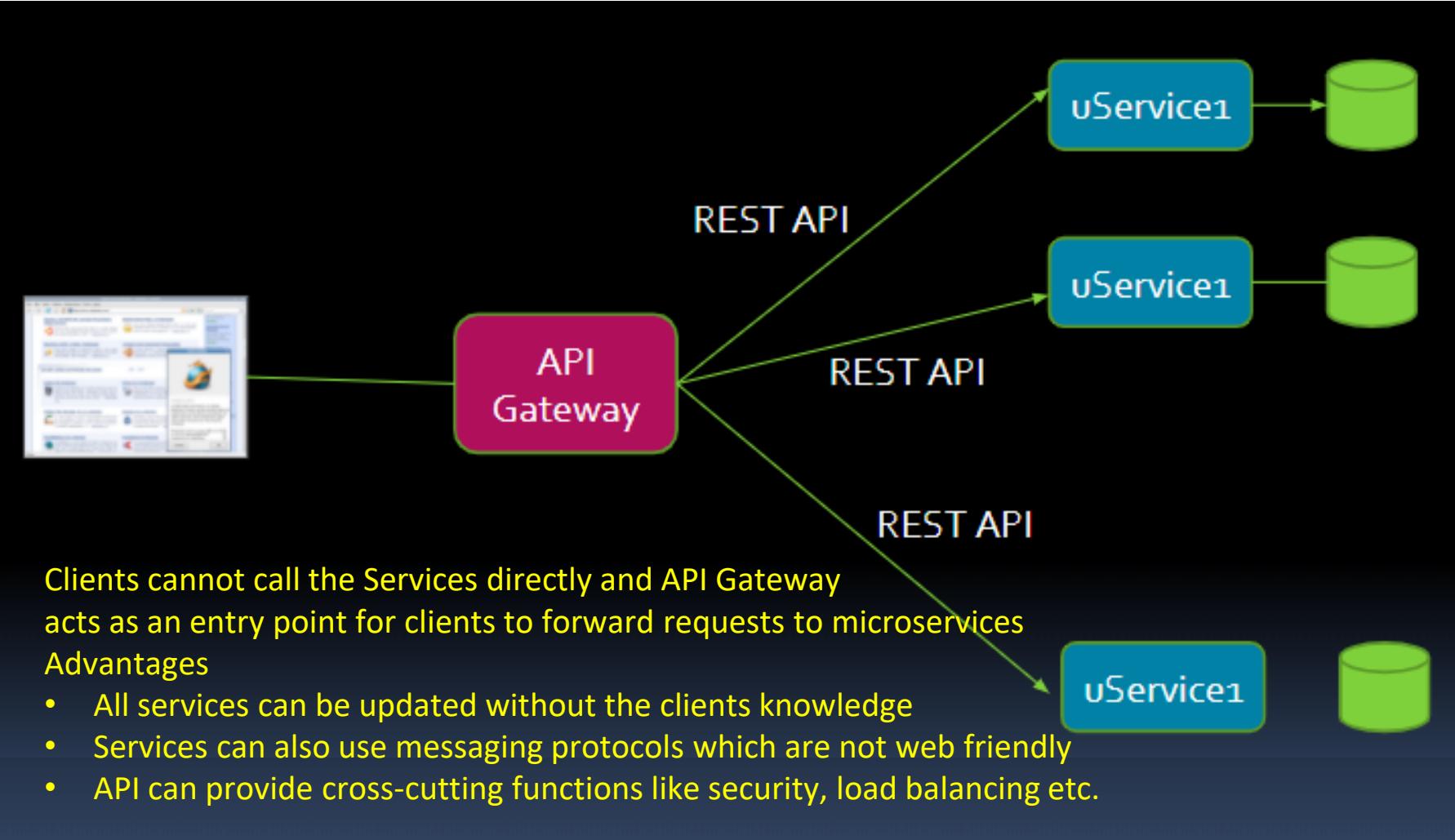
With microservice architecture, it's easy to build complex applications. If the features of the application are analyzed properly, we can break it down into independent components which can be deployed independently. Then, even the independent components can be further broken down into small independent tasks which can be deployed independently as a microservice. Deciding the boundaries of a microservice can be quite challenging. It's actually an evolutionary process, but once we decide on a microservice, it's easy to develop, as there are no limitation in technologies.

5. Dynamic Scalability:

Scalability is a major advantage in microservice architecture. Each microservice can be scaled individually. Since individual microservices are much smaller in size, caching becomes very effective.

6. Continuous deployment:

Continuous deployment becomes easier. In order to update one component, we have to redeploy only that particular microservice.



There are the following principles of Microservices:

- **Single Responsibility principle**

The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

- **Modelled around business domain**

Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

- **Isolate Failure**

The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

- **Infrastructure automation**

The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

- **Deploy independently**

Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other services.

Microservices Limitations

Building: Upfront time needs to be spent in terms of identifying dependencies between your services. Be aware that completing one build might trigger several other builds, due to those dependencies. You also need to consider the effects that microservices have on your data.

Testing: Integration testing, as well as end-to-end testing, can become more difficult as a failure in one part of the architecture could cause something a few hops away to fail, depending on how you've architected your services to support one another.

Versioning: When you update to new versions, keep in mind that you might break backward compatibility. You can build in conditional logic to handle this, but that gets unwieldy and nasty, fast. Alternatively, you could stand up multiple live versions for different clients, but that can be more complex in maintenance and management.

Deployment: Needs initial investment towards automation as the complexity of microservices becomes overwhelming for human deployment. Think about how you're going to roll services out and in what order.

Logging: With distributed systems, you need log management as the scale makes it hard to manage.

Monitoring: This may need to have a centralized view of the system to pinpoint sources of problems.

Debugging: Remote debugging through your local integrated development environment (IDE) isn't an option and it won't work across dozens or hundreds of services. Its difficult with no single answer to how to debug.

Connectivity: Consider service discovery, whether centralized or integrated

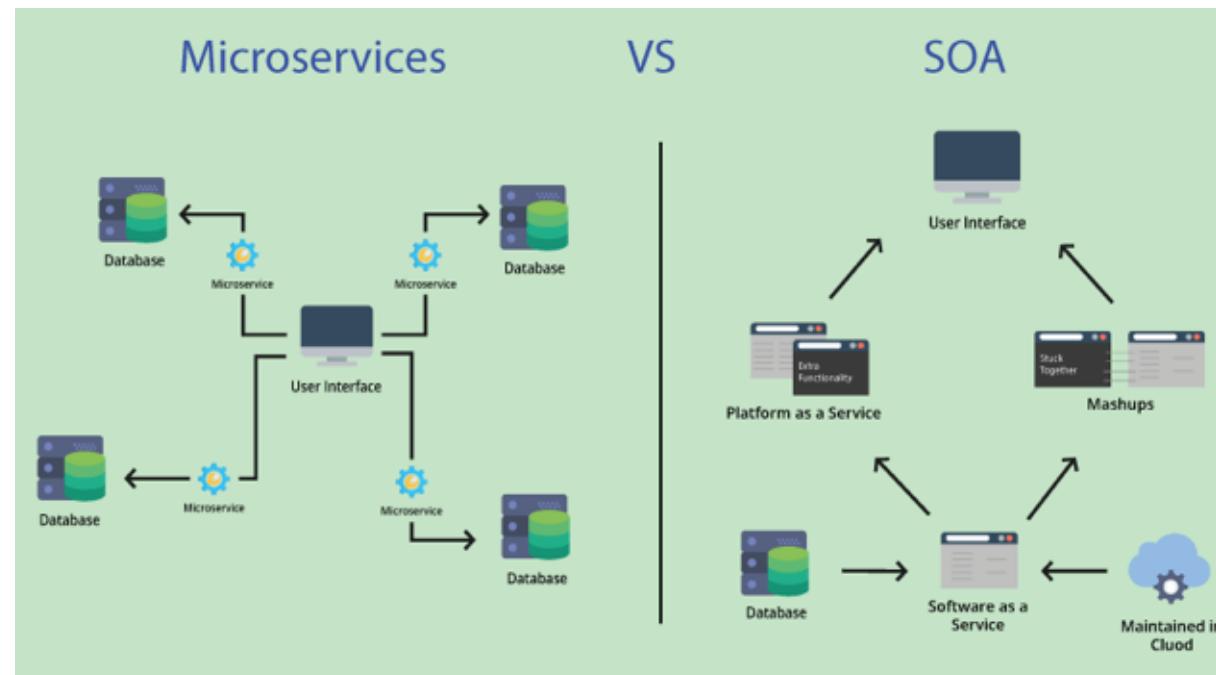
Microservices Limitations (others)

- Quick setup needed: You cannot spend a month setting up each microservice. You should be able to create microservices quickly.
- Automation: Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring, etc.
- Visibility: You now have a number of smaller components to deploy and maintain, maybe 100 or maybe 1000 components. You should be able to monitor and identify problems automatically. You need great visibility around all the components.
- Configuration Management: You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution

What is SOA?

Service-oriented architecture was largely created as a response to traditional, monolithic approaches to building applications. SOA breaks up the components required for applications into separate service modules that communicate with one another to meet specific business objectives.

Microservice architecture is generally considered an evolution of SOA as its services are more fine-grained, and function independently of each other

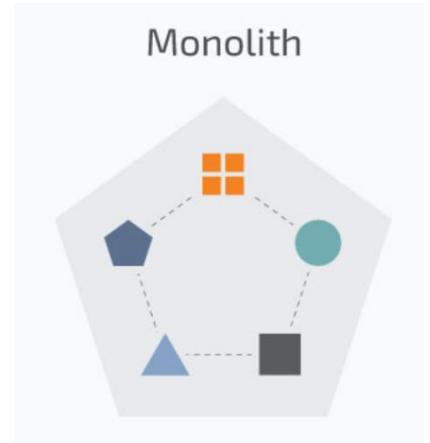


Comparison with SOA

SOA	Microservice Architecture
Follows “ share-as-much-as-possible ” architecture approach	Follows “ share-as-little-as-possible ” architecture approach
Importance is on business functionality reuse	Importance is on the concept of “ bounded context ” or Single Responsibility
They have common governance and standards	They focus on people, collaboration and freedom of other options
Uses Enterprise Service bus (ESB) for communication	Simple messaging systems
They support multiple message protocols	They use lightweight protocols such as HTTP/REST etc.
Multi-threaded with more overheads to handle I/O	Single-threaded usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on decoupling
Traditional Relational Databases are more often used	Modern Relational Databases are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

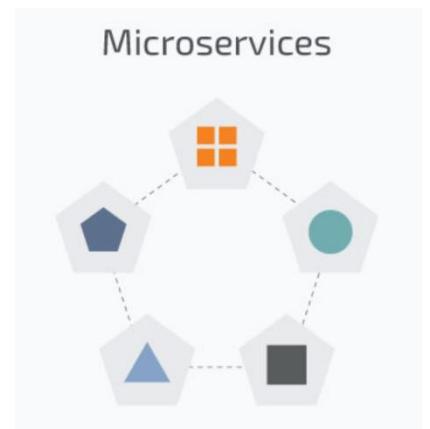
Monolithic Architecture

It is considered to be a traditional way of building applications. A monolithic application is built as a single unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database.



Microservices Architecture

While a monolithic application is a single unified unit, a microservices architecture breaks it down into a collection of smaller independent units. These units carry out every application process as a separate service. So all the services have their own logic and the database as well as perform the specific functions.





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Challenges of migrating monolithic applications to microservices

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

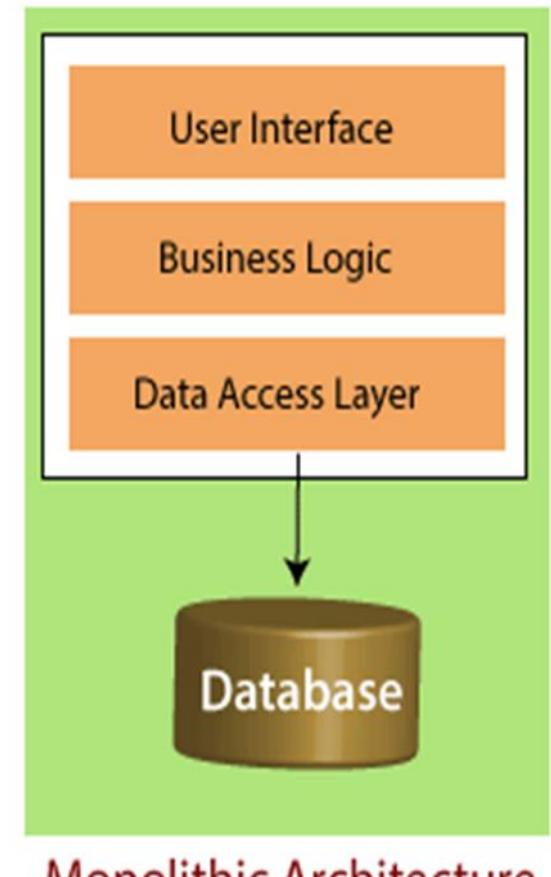
Need for Migration of applications to the Cloud

There are many benefits for applications to be on the cloud as seen earlier. Some of these could be

- **Scalability** : Applications which are based on the cloud, have a better ability to scale up or down based on your IT requirements and business plan.
- **Cost**: Reduce operational costs while improving IT processes. Applications in the cloud only pay for what they use, and have no need to maintain costly data centers when your important information is hosted in the cloud
- **Integration** : Cloud allows business to seamlessly connect systems together and improve efficiency with all your services. This alleviates risks like refresh of hardware etc.
- **Access** : all data is stored in the cloud, it can be accessed no matter what happens to your physical machinery.
- **Security** : In the cloud environment cloud infrastructure companies have built their cloud offering with privacy and security in mind making the applications to be secure. Most cloud providers also take care of issues like keeping unwanted traffic outside a specific scope from accessing the machines on which your business data and apps reside and ensuring automatic security updates are applied to their systems to keep from being vulnerable to the latest known security threats.

Need for Migration of applications to the Cloud

- Applications are designed and built with different approaches or patterns or techniques or have different Application Architectures
 - These could be built with Architectures which are not designed in a manner which can utilize all the benefits provided by the Cloud
- Eg. Monolithic Architectures discussed in the previous session which is built as a single and indivisible unit.
- This brings back the challenges like not easily integrated, scalability etc.



What is Migration

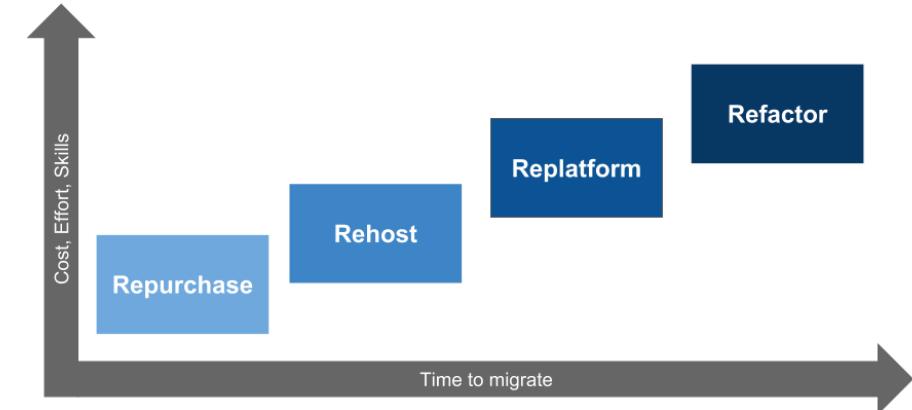
- Migrating to the cloud involves moving critical services and applications from on-premise and co-located hardware to Amazon Web Services, Google Cloud Platform, Microsoft Azure, or other cloud providers to take advantage of benefits of being on the cloud.
- These services allow you to manage IT infrastructure entirely remotely without the security risk, inconvenience, and cost of maintaining on-premise hardware.
- **Application migration** is the process of moving software applications from one computing environment to another. This can include migrating applications from one data center to another, such as from a public to a private cloud, or from a company's on-premises server to a cloud provider's environment.
- Organizations migrate applications to the cloud to take advantage of an improved cost structure, responsive scalability, and the ability to quickly update apps to meet changing demands etc.

CLOUD COMPUTING

Migration strategies

There are many ways to migrate applications to the cloud. The 6R framework proposes 6 different strategies for application migration depending on the amount of cloud nativity, business needs and level of effort. These 6Rs are:

- Re-host
- Re-platform
- Re-architecting
- Re-purchase
- *Retire*
- *Retain*



Of these 6 strategies, retire and retain are design strategies used post-migration. Retire means to remove features of the application that are no longer in use while retain means to only keep those features that are critical to the application. Re-purchase strategy is the method of moving perpetual licenses to a software-as-a-service model. For example, move from a customer relationship management (CRM) to Salesforce.com.

Ref - <https://cloudsoft.io/blog/a-practical-guide-to-understanding-the-6rs-for-migration-to-aws>

<https://www.heptabit.at/blog/6rs-application-migration-strategies#:~:text=6R%20framework%20proposes%20a%20separation,purchase%2C%20Retire%2C%20and%20Retain.&text=It%20is%20essential%20to%20prioritize,immediate%20value%20from%20the%20process.>

- Re-hosting: This is also known as the lift-and-shift migration strategy. Applications are migrated to the cloud without changes. It is most suitable for organizations that need to meet a business objective quickly. It is the fastest and easiest migration strategy. However, applications will not be able to fully reap the benefits of the cloud and have limited scalability.
- Re-platforming: This migration strategy is also known as lift-tinker-and-shift. The basic architecture of the application still remains the same but some optimizations may be made to achieve some of the benefits of the cloud. One of the most common alterations done to these applications is by managing database instances to a database-as-a-service platform like Amazon RDS.

Re-architecting

Re-architecting is one of the migration strategies that results in migrating non cloud native applications to the cloud, this leads to the highest return of investment and exploits complete cloud nativity. SOA or microservices are few of the architectures that can be considered while re-architecting applications. This strategy involves systematic and aggressive remodeling of existing architecture. It requires extensive modification of the application's codebase. There is no standard set of steps to be followed for re-architecting an application as it is entirely dependent on the application's architecture, logical components and functionalities.

The different application architectures are -

monolithic, service oriented architecture(SOA), microservice.

Therefore in re-architecting we can convert the existing architecture of the application to another.

Example: 1) monolithic to SOA

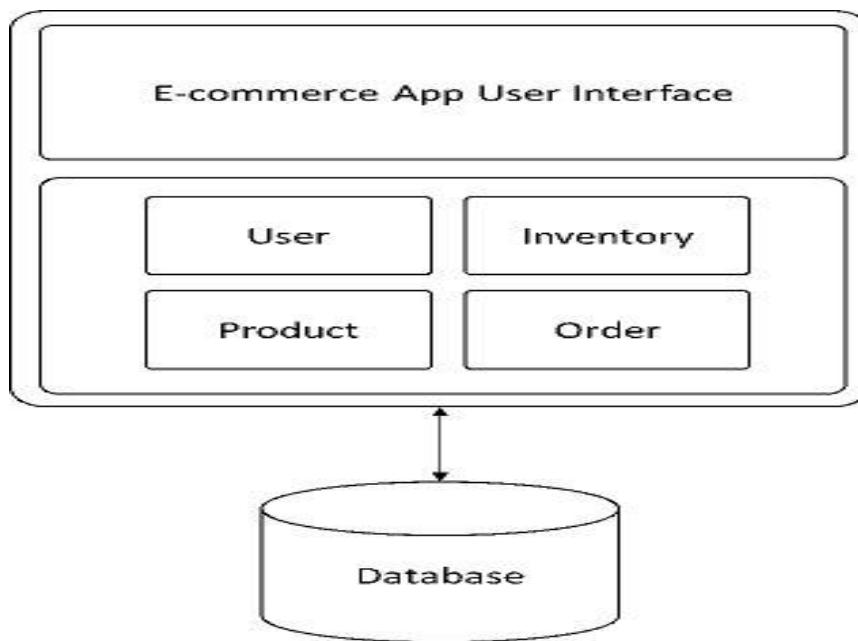
- 2) monolithic to microservice
- 3) SOA to microservice

Microservice architecture reaps all the features of cloud. Hence can be called cloud-native (built for cloud).

Migration is not easy

Microservices architecture promises to solve the shortcomings of monolithic applications, so many enterprises are interested in migrating their applications to be microservices. This migration is a worthwhile journey, but not an easy one.

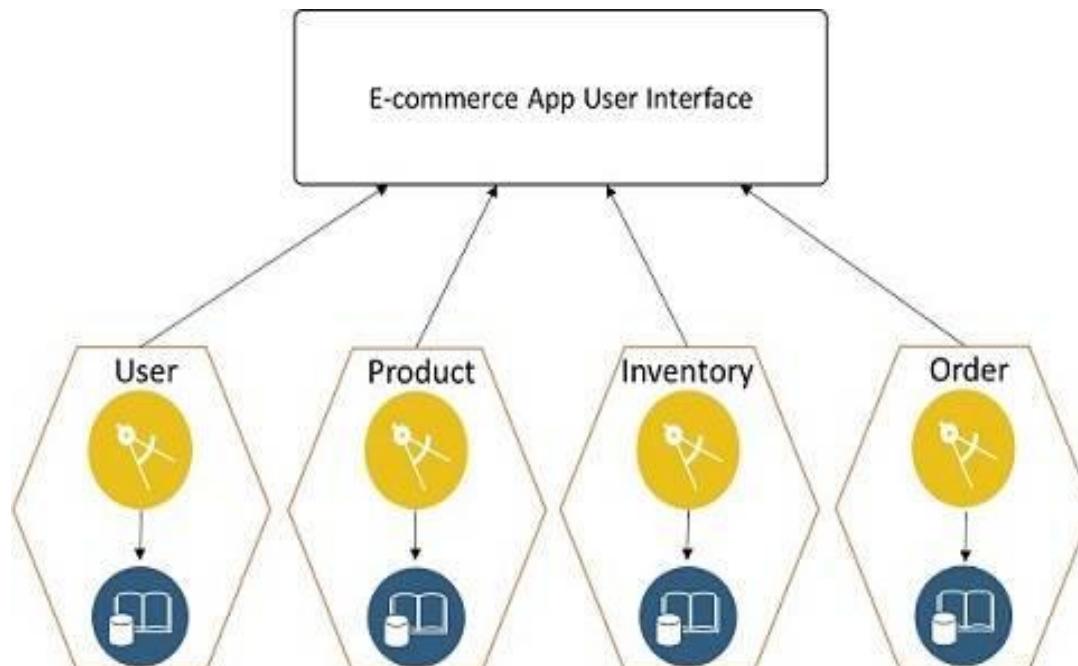
Most enterprises have a large application in place for their business use cases. Take the example of an e-commerce application used by an online retailer. The diagram below shows its monolithic architecture, where all services interact with a single database.



Monolithic architecture seems like a good idea to quickly get the application started, it becomes a problem over time. As the business and user base grows, customers expect newer user experiences, and integration requirements increase, the monolithic approach becomes a bottleneck to growth.

In contrast, Figure below shows how the same e-commerce application can be broken down into a microservices architecture.

Each service has its own database.



Migration is not easy

The idea seems simple but is sometimes difficult to perceive. The architecture team should consider the following challenges:

1. Service decomposition : Possible candidates that could represent a microservice.
2. Persistence : Monolithic database decomposition
3. Tackling transaction boundaries
4. Performance
5. Testing
6. Inter-service communication

In transitioning an existing monolith to a microservices, you would typically need to decompose the existing application into granular microservices. This is a challenging task as we need to identify possible independent components in a monolithic application.

Although breaking down a monolithic application can seem like an uphill task to start, with certain guidelines even this mammoth task is possible:

1. Stop adding more things to the monolithic app. Fix what is broken and accept only small changes.
2. Find out the so-called seams in your monolithic app. Identify components that are more loosely coupled than others and use them as a starting point.
3. Identify low hanging fruits, such as the components for which business units want to add more advanced features.

During this transitioning process, you would typically need to decompose the monolith to building more and more granular microservices to suit the business needs.

Once this is accomplished, there would be more moving parts in the application. As a result, this would lead to operational and infrastructural overheads, i.e., configuration management, security, provisioning, integration, deployment, monitoring, etc.

One of the ways to reduce these complexities is in using containerization. In using containerization, provisioning, configuration, and deployment of microservices would be simplified.

Persistence : Monolithic database decomposition

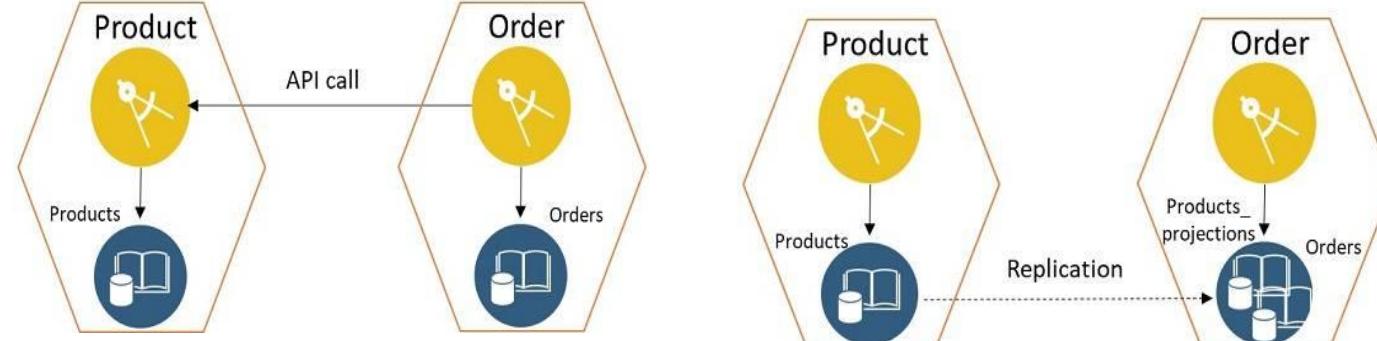
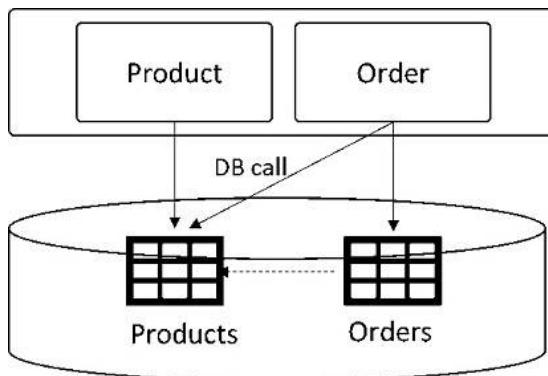
A monolithic application typically contains a single data store. In contrast, a microservices-based application contains multiple databases, typically one for each service. One of the key challenges in transitioning to microservices-based architecture is in understanding and handling decentralized data management.

Splitting the data model of a monolithic application to fit the autonomous data models of a microservices-based application is challenging. Breaking a monolith data model into separate autonomous data models that are local to each microservice is a daunting task, the following challenges need to be considered:

- A. Reference tables
- B. Shared mutable data
- C. Shared table

A. Reference tables :

- The most common type of pattern that is seen within monolithic applications is a reference table.
- In this pattern, the function or module accesses a table that belongs to another function or module.
- The joins between a module's own table and another table are used to retrieve the required info.
Using the previous example of an e-commerce application, the Order module uses a reference to the Products table to retrieve product information.
- In a microservices architecture, the Products table and Order module become separate microservices. Here are two options for you to consider to segregate the database objects :
 1. Data as an API
 2. Projection/Replication of data

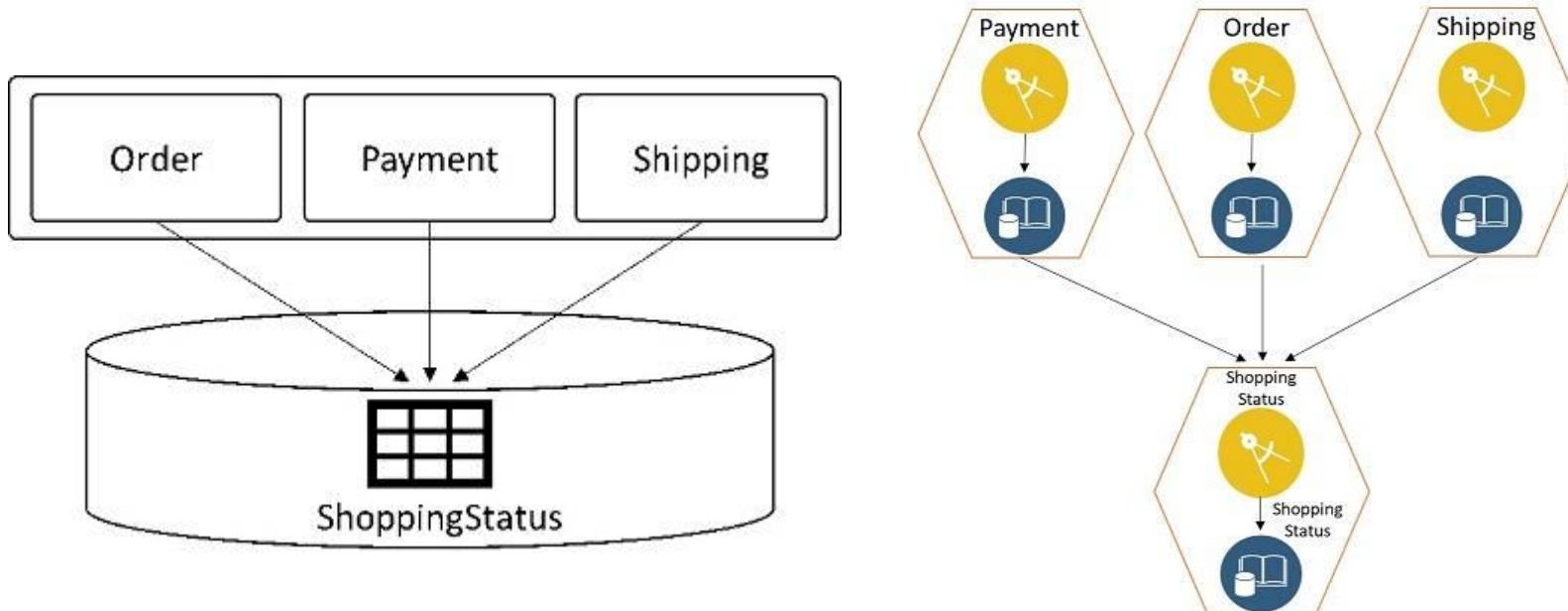


Persistence : Monolithic database decomposition

B. Shared mutable data :

In monolithic applications, there is a common pattern that is known as shared mutable state. This pattern represents certain domain realities that are modeled in a database and accessed by different functionalities or modules of the application. This is mainly done for convenience. For example, in the e-commerce application, the Order, Payment, and Shipping functionalities use the same ShoppingStatus table to maintain the customer's order status throughout the shopping journey.

While moving to microservice architecture they should eventually be modeled as a separate microservice.

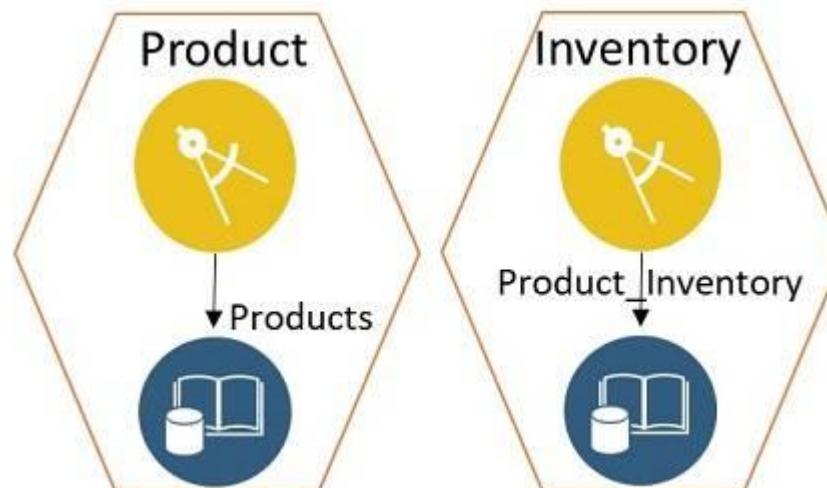
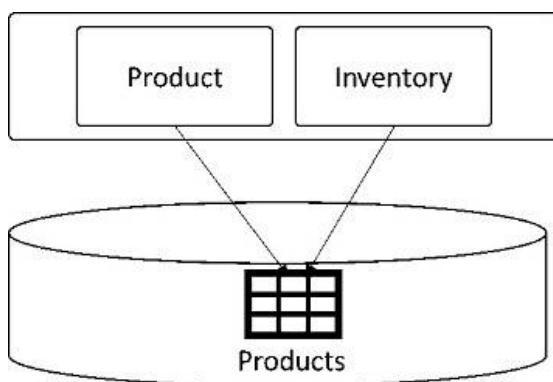


Persistence : Monolithic database decomposition

C. Shared table:

The shared table pattern is very similar to the shared mutable state pattern and is the result of erroneous domain modeling. In this scenario, a database table is modeled with attributes that are needed by two or more functionalities or modules. Again, this is done mostly for convenience reasons. To explain this, Figure below shows a scenario where the Products table is modeled to cater to the Product and Inventory functionalities.

While moving to a microservices architecture, Product and Inventory are modeled as separate services. The Products table is split into individual entities that belong to the specific bounded context of the separate Product and Inventory microservices, as shown in Figure.

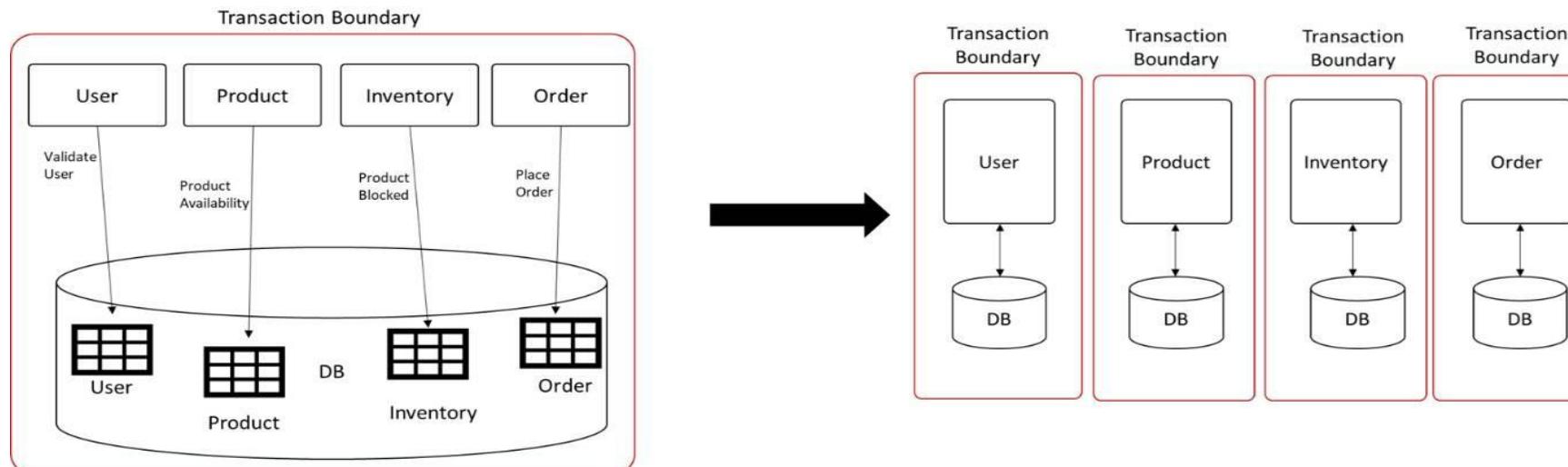


Tackling transaction boundaries

With a monolithic application that has a single database, you could guarantee the ACID (Atomicity, Consistency, Isolation, Durability) properties and ability to perform transactions by keeping locks on rows.

Things changed drastically when enterprises moved from service-oriented architecture to microservices architecture. They now must deal with database per service eventually leading to distributed transaction trouble. Following are the ideal ways to deal with distributed transactions :

- A. Two-phase commit (2PC)
- B. Compensating transactions (Saga transactions)

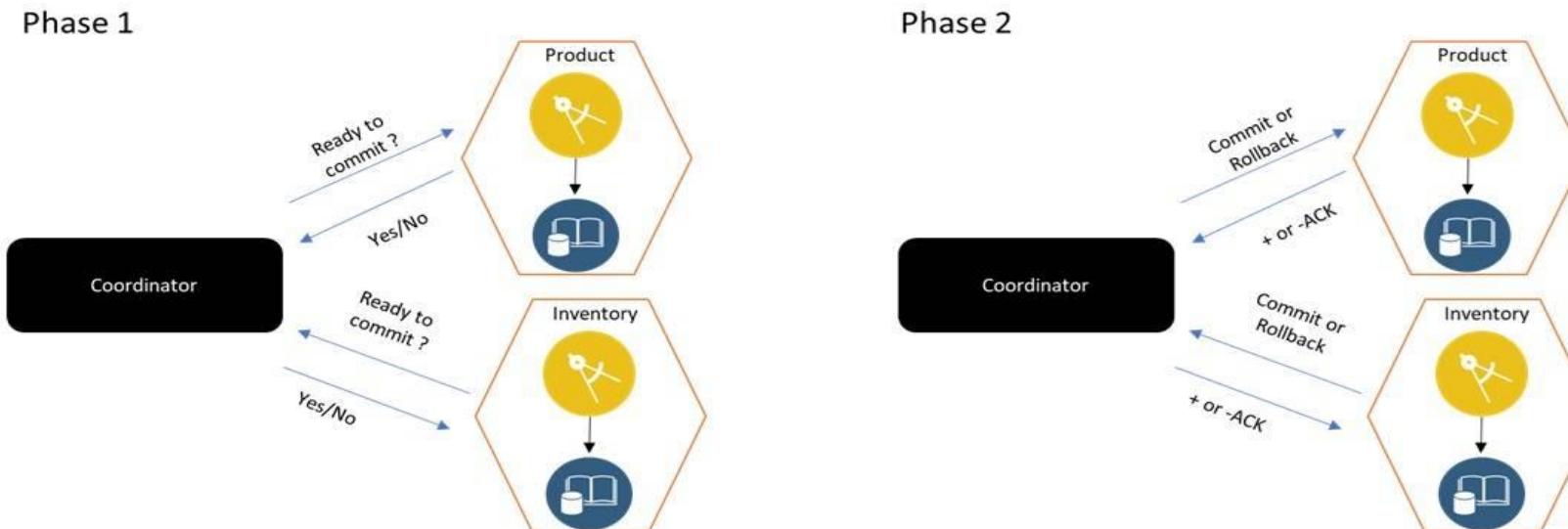


A. Two-phase commit (2PC):

In a two-phase commit, you have a controlling node that houses most of the logic, and a few participating nodes on which the actions are performed. It works in two phases:

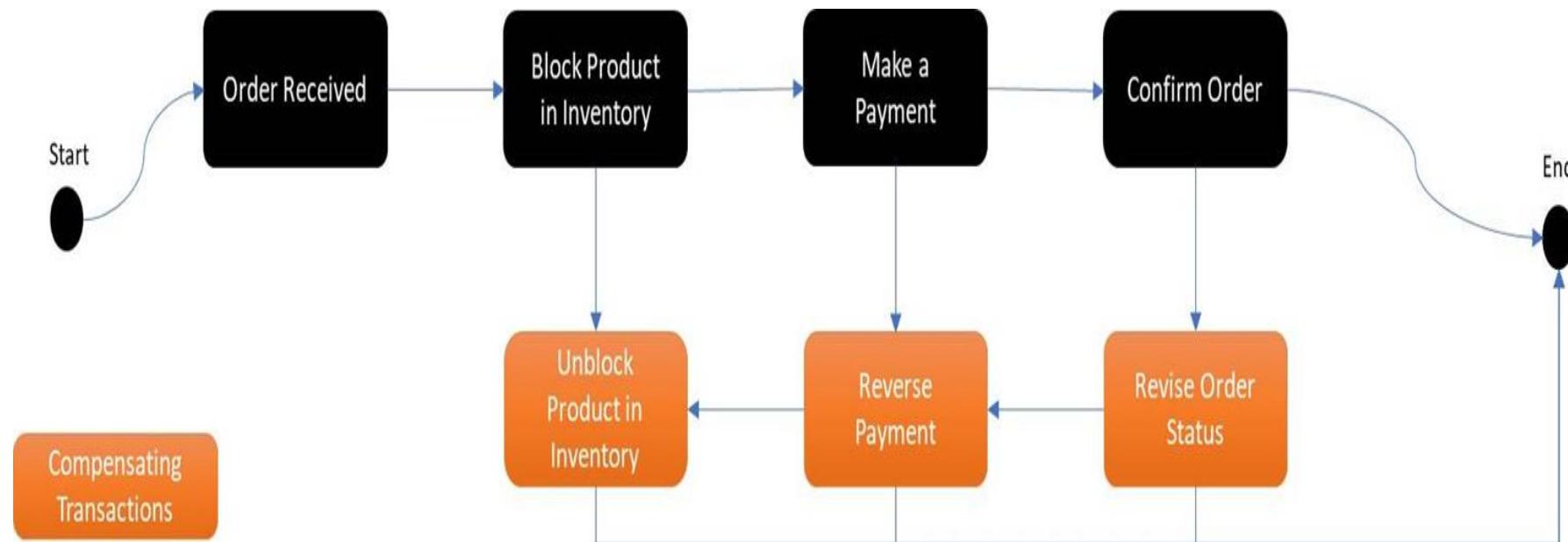
Prepare phase (Phase 1): The controlling node asks all of the participating nodes if they are ready to commit. The participating nodes respond with yes or no.

Commit phase (Phase 2): If all of the nodes replied in the affirmative, then the controlling node asks them to commit. Even if one node replies in the negative, the controlling node asks them to roll back.



B. Compensating transactions (Saga transactions):

A saga is a sequence of local transactions. Each service in a saga performs its own transaction and publishes an event. The other services listen to that event and perform the next local transaction. If one transaction fails for some reason, then the saga also executes compensating transactions to undo the impact of the preceding transactions.



The increase in resource usage may cause a microservices-based application to execute slower.

You can overcome this challenge by :

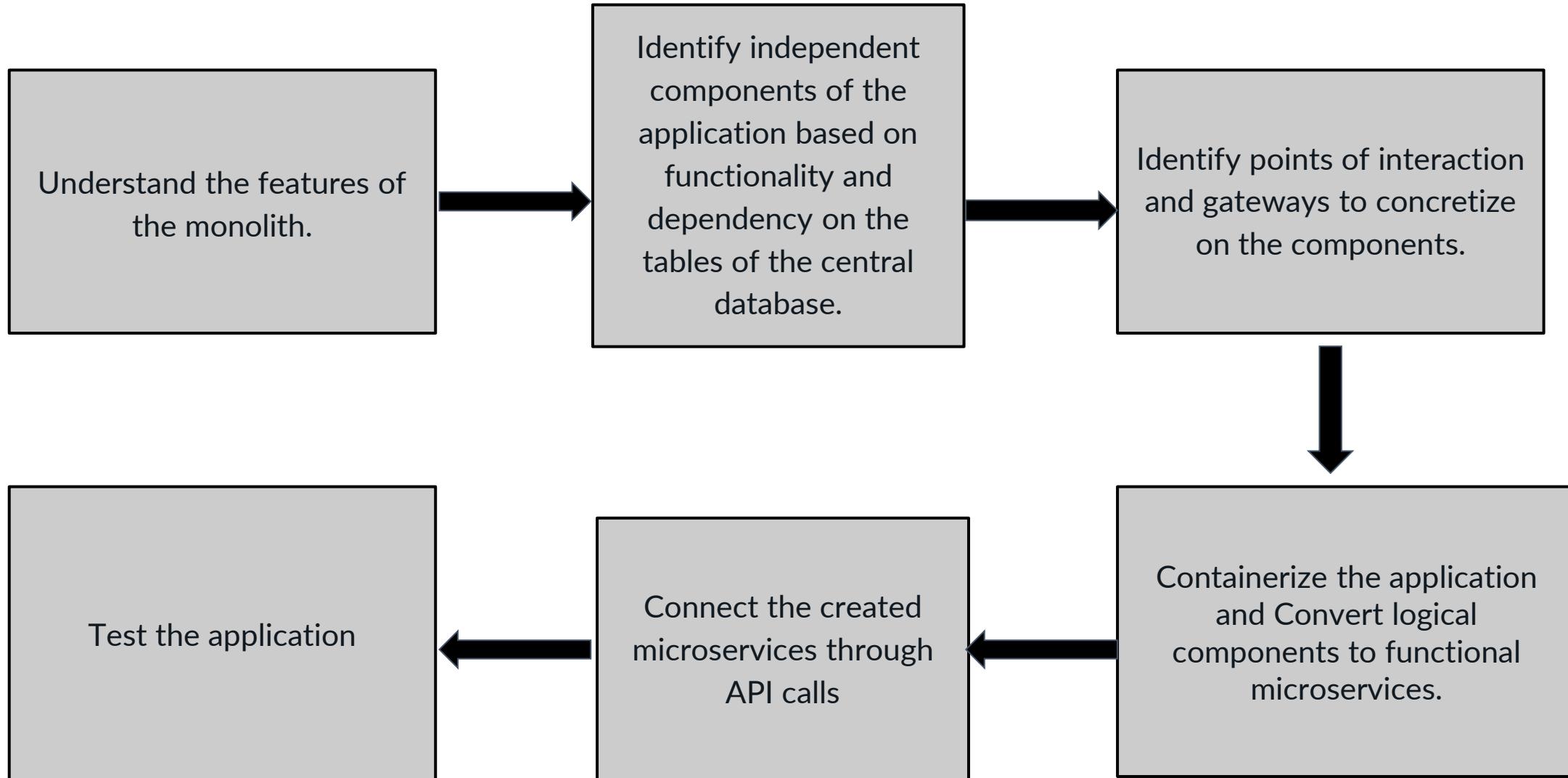
1. Introducing additional servers.
2. You can log performance data and detect the problems. You should take advantage of logging to store performance data in a repository so that you can analyze the data at a later point of time.
3. You can implement throttling, handle service timeouts, implement dedicated thread pools, implement circuit breakers and take advantage of asynchronous programming to boost the performance of microservice-based applications.

- Testing microservices can become challenging, particularly the integration tests. To write an effective integration test case, the QA engineer should have good knowledge of each of the services that are a part of the solution.
- Another reason why testing a microservices-based application is difficult is because microservices-based applications are generally asynchronous. The best approach to solving these testing challenges is adopting various testing methodologies and tools and leveraging continuous integration capabilities through automation and standard agile methodologies.

In a distributed system, the components should be able to communicate with each other and in microservices-architecture, it is no exception. In a monolithic application, the components invoke one another through method or function calls. In contrast, a microservices-based application is distributed in nature with each service running isolated from another service. Hence, it is imperative that services in a microservice-based application communicate using inter-process communication (also known as IPC) mechanisms. However, inter-service communication in a microservices-based application poses a lot of challenges.

Since microservices are distributed in nature, remote invocation of these services is a challenge and you should understand the necessary patterns to overcome the challenges involved. Since services in a microservice-based application communicate with one another using IPC mechanisms you should consider issues like, how the services will interact, how to handle failures, etc.

Generic steps for monolithic to microservices architecture



Some of the most common challenges associated with the conversion of monolithic applications to microservices / re-architecting are as follows:

- Identification of services based on the overall application's functionality
- Ensuring appropriate connection and communication between services
- Conversion of large and bulky desktop applications are challenging
- Identifying the boundary between re-architecting and rebuilding the entire application
- Requires more effort
- Operational complexity is also increased due to the increased demands on managing these services and monitoring them.
- Coordinating a large number of rapidly changing services necessitates automated continuous integration and continuous delivery.

Attributes which can be used for comparison of the migration approaches

These approaches can be compared using some of the attributes like

1. Performance
2. Scalability
3. Throughput
4. Accessibility
5. Loadtime
6. Cloud Nateness

Results from migration

However, it is important to note that not all applications are suited to all types of migration strategies. Based on the need of the business, and the amount of cloud nativeness required, the type of strategy to be used may differ. Consider the following table that shows the suitability of migration strategies based on application and business requirements

Suitability	Rehosting	Replatforming	Rearchitecting
Application suitability	<ul style="list-style-type: none">● Web compatible UI● MVC architecture● Flask, Symphony, .NET can be rehosted without complications● Applications not built with web frameworks but have a web-compatible UI (HTML, PHP) not ideal	<ul style="list-style-type: none">● Web-compatible UI and MVC architecture● Must allow easy connection to the database server● Traditional apps using Xampp-like servers are suitable	<ul style="list-style-type: none">● Monolithic architecture, but are web-compatible are suited● Large codebases● Resource intensive● Desktop applications built using frameworks are not suited
Business suitability	<ul style="list-style-type: none">● Higher network speed● Low technical debt and cost of maintenance.● Small apps that require a temporary boost in performance	<ul style="list-style-type: none">● Database hosting most common● Improve DB scalability● Boosts fault tolerance● Data is more resilient.	<ul style="list-style-type: none">● Looking for large improvements in availability, testability, continuous delivery, reusability and lower infrastructure costs



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Introduction to Cloud Computing
Terminologies
Parallel, Distributed

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for classroom presentation only.

Cloud Computing

- Cloud computing is basically delivering computing at the Internet scale.(T2)
- Compute, storage, networking infrastructure as well as development and deployment platforms are made available on-demand within minutes(T2)
- *Cloud computing is a model for enabling ubiquitous (**where-ever**), convenient, **on-demand** (**when-ever**) network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that may be shared (across applications) and can be rapidly provisioned and released (**quickly**) with minimal management effort or service provider interaction*
- **Data centers** house the physical compute, storage and networking infrastructure (along with components which enable functioning or running of these active IT components). These Data center's are also called *Cloud Data center's* when they host these physical resources and enable Cloud computing over the Internet.
- Large clouds, predominant today, often have functions distributed over multiple locations from central servers
- If the connection to the user is relatively close, it may be designated an edge server.
(https://en.wikipedia.org/wiki/Cloud_computing)

Introduction to Cloud Computing

Features/Characteristics of Cloud Computing(T2)

On demand self-service: The compute, storage or platform resources needed by the user of a cloud platform are self-provisioned or auto provisioned with minimal configuration.

Broad network access: Ubiquitous access to cloud applications from desktops, laptops to mobile devices is critical to the success of a Cloud platform

Resource pooling: Cloud services can support millions of concurrent users; cloud services need to share resources between users and clients in order to reduce costs.

Scalability: Cloud Services can accommodate larger or smaller loads while supporting some of the expectations of QoS like response time

Rapid elasticity: A cloud platform should be able to rapidly increase or decrease computing resources as needed.

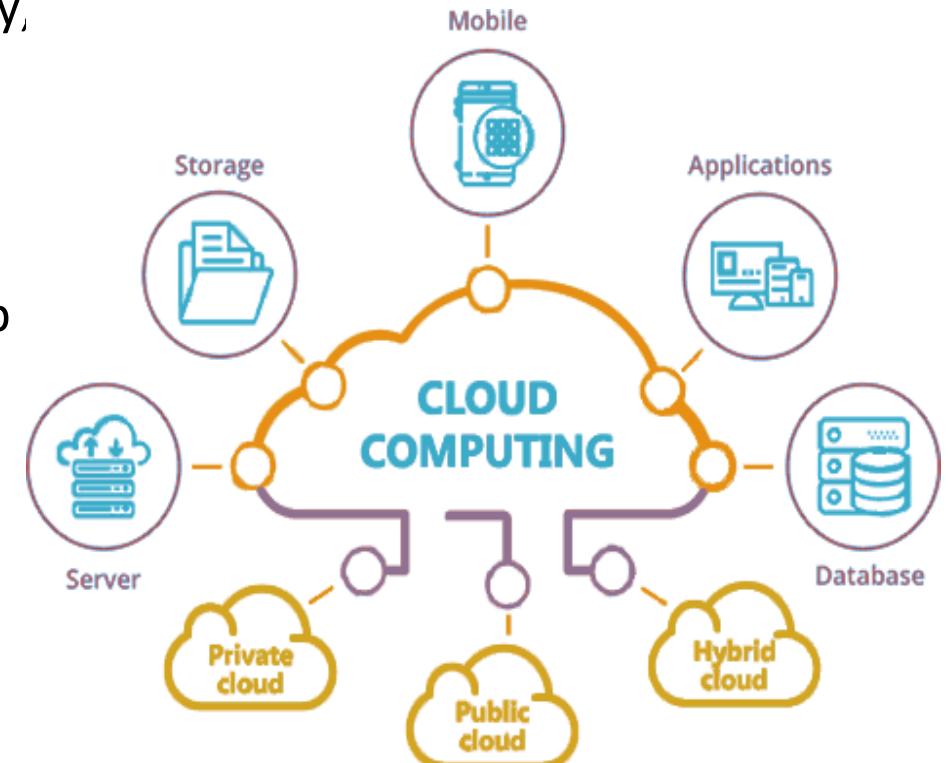
Measured service: One of the compelling business use cases for cloud computing is the ability to “pay as you go,” where the consumer pays only for the resources that are actually used by her applications

Introduction to Cloud Computing

Cloud computing usage (<https://aws.amazon.com/what-is-cloud-computing/>)

Organizations of every type, size, and industry are using the cloud for a wide variety of use cases, such as data backup, disaster recovery, email, virtual desktops, software development and testing, big data analytics, and customer-facing web applications.

- For example, healthcare companies are using the cloud to develop more personalized treatments for patients.
- Financial services companies are using the cloud to power real-time fraud detection and prevention.
- And video game makers are using the cloud to deliver online games to millions of players around the world.



Agility

Quickly spin up resources as you need them—from infrastructure services, such as compute, storage, and databases, to Internet of Things, machine learning, data lakes and analytics, and much more.

Elasticity

Scale these resources up or down to instantly grow and shrink capacity as your business needs change.

Cost savings

The cloud allows you to trade capital expenses (such as data centers and physical servers) for variable expenses, and only pay for IT as you consume it.

Deploy globally in minutes

With the cloud, you can expand to new geographic regions and deploy globally in minutes

- Over the past 60 years, computing technology has undergone a series of platform and environment changes
- There have been number of changes in machine architecture, operating system platform, network connectivity, and application workload.
- We have evolved from the use of a centralized computer to solve computational problems, to the use of parallel and distributed computing with multiple computers to solve large-scale problems over the Internet.
- The general computing trend is to leverage shared web resources and massive amounts of data over the Internet.

Evolution of Cloud Computing (T1 1.1.1.1)

- Billions of users use the internet everyday and need to be provisioned concurrently with High performance computing capability to a large number of users
- High performance computers (HPC) providing this capability for short amounts of time is no longer optimal.
- High throughput computing (HTC) is what is needed using distributed and parallel computing.
- Since 1990, we see the proliferation and the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds.
- These systems are employed by both consumers and high-end web-scale computing and information services
- The general computing trend is to leverage shared web resources and massive amounts of data over the Internet

High performance computing (HPC) is the ability to process data and perform complex calculations at high speeds (mostly for short amount of time .. Hours/days)

High Performance Compute (HPC) could also be viewed as a system which has components which can provide high performance to applications using it.

High Throughput Compute (HTC) could be viewed as ability to handle large amounts of computing, for much longer times (months and years, rather than hours and days)

Evolution of Cloud Computing (T1 1.1.1.1)

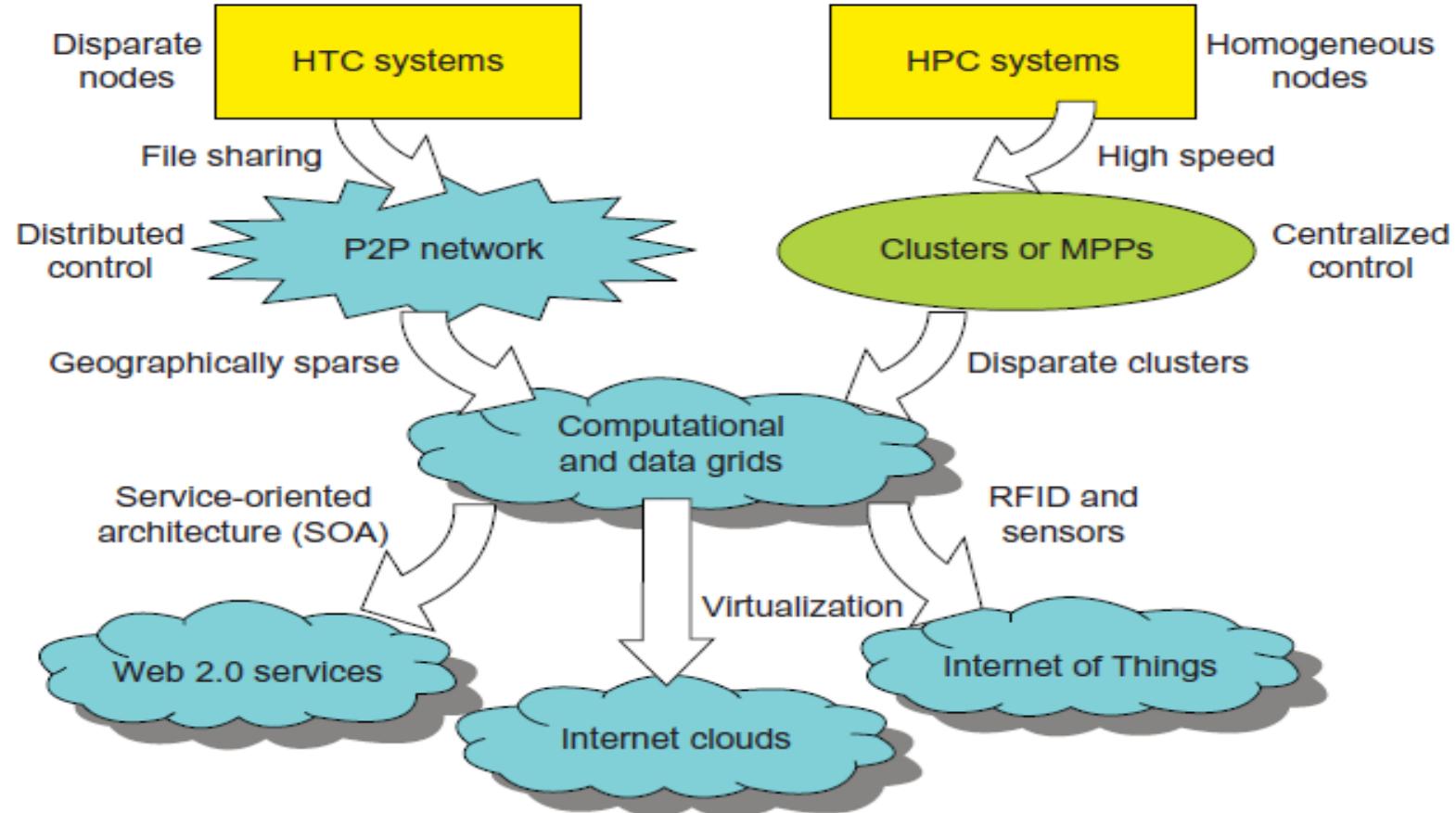


FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

Evolution of Computing - High Performance Computing - Context

- HPC generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business.
- A helpful way to help understand what high performance computers are is to think about the what's in them. You have all of the elements you'd find on your desktop — processors, memory, disk, operating system — just more of them
- High performance computers, which are of interest to small and medium-sized businesses today are really *clusters* of computers.
- People often refer to the individual *computers* in a cluster as *nodes*
- HPC systems are mostly expected to support for short amount of time .. In Hours/days

Evolution of Computing : Distributed Computing

Distributed computing (multiple complete systems)

A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through message passing

Distributed system components can be located on different networked computers that coordinate their actions by communicating via pure HTTP, RPC-like connectors or message queues.

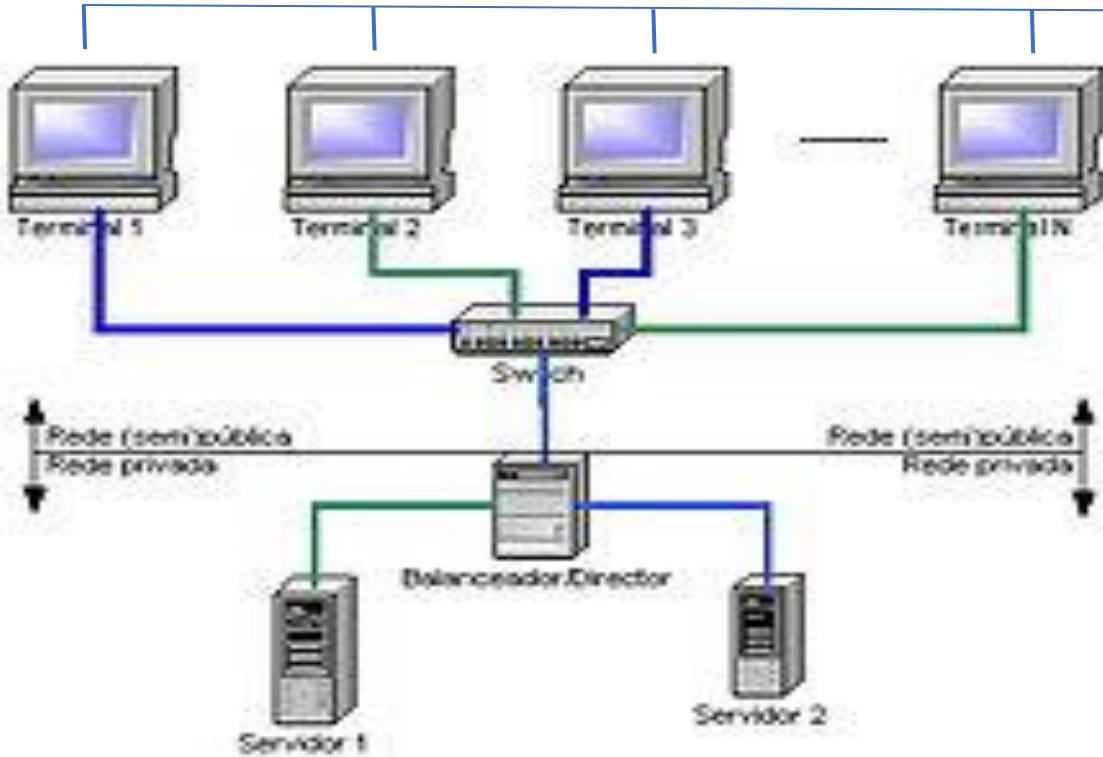
Significant characteristics of distributed systems include independent failure of components and concurrency of components



Distributed Computing

Evolution of Computing : Clusters

A computer cluster is a set of loosely or tightly connected computers that work together so that, in many aspects, they can be viewed as a single system. Clusters have each node set to perform the same task, controlled and scheduled by software.

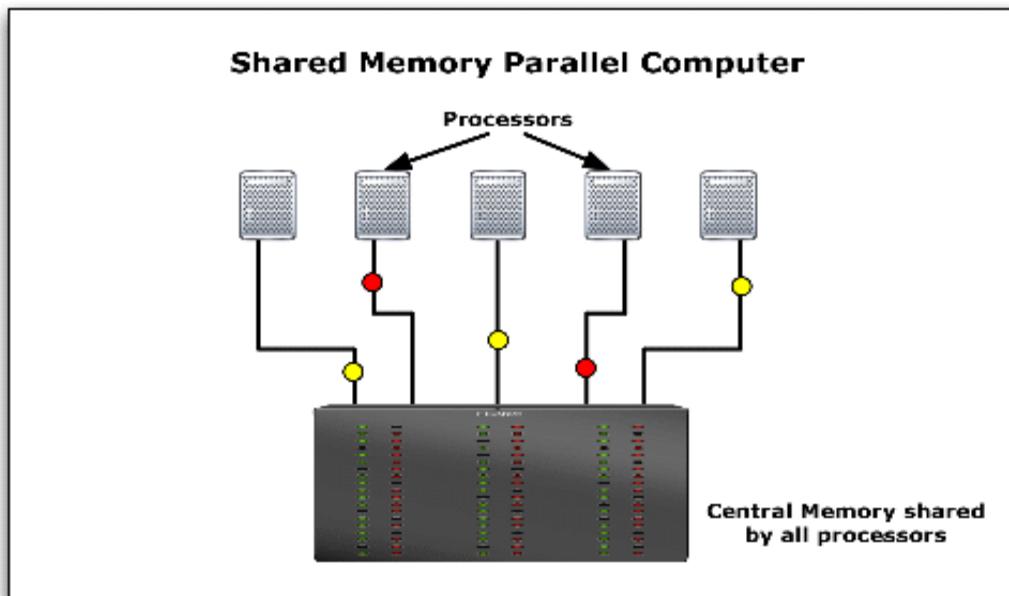


Mostly have similar hardware configuration and run the same OS (different instances though)

Evolution of Computing : Parallel Computing

Parallel Computing (Single system with multiple processors)

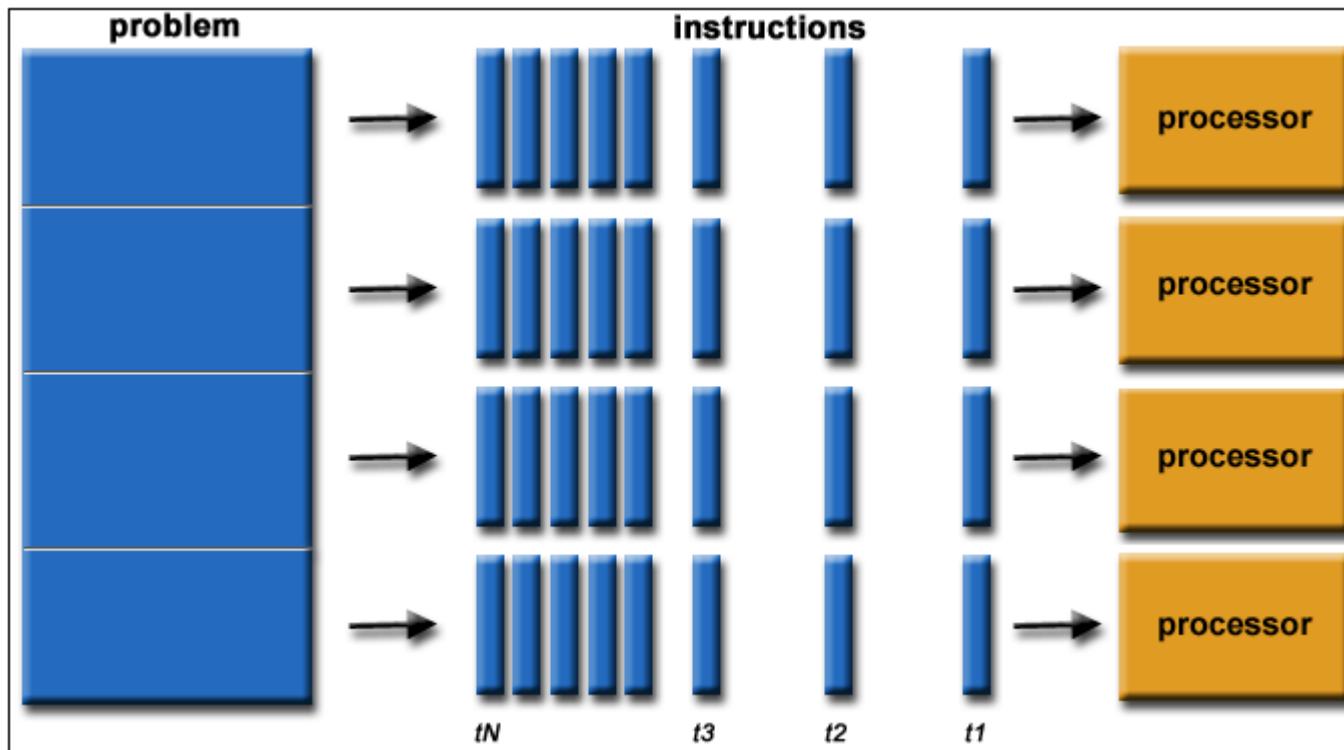
In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Inter- processor communication is accomplished through shared memory or via message passing.



The primary goal of parallel computing is **to increase available computation power** for faster application processing and problem solving

Evolution of Computing : Parallel Computing

In **Parallel Computing** several processors simultaneously execute multiple, smaller calculations broken down from an overall larger, complex problem. These smaller calculations communicate through shared memory and the results of these can be combined as part of an overall program



There are various degrees of parallelism

- Bit Level Parallelism
- Instruction Level Parallelism
- Task Level Parallelism
- Data Parallelism

Evolution of Computing : Parallel Computing

Bit Level Parallelism (<https://www.tutorialspoint.com/types-of-parallelism-in-processing-execution>)

Bit-level parallelism is a form of parallel computing which is based on **increasing processor word size**. In this type of parallelism, with increasing the word size **reduces the number of instructions** the processor must execute in order to perform an operation on variables whose sizes are greater than the length of the word.

E.g., consider a case where an 8-bit processor must add two 16-bit integers. First the 8 lower-order bits from each integer were must added by processor, then add the 8 higher-order bits, and then two instructions to complete a single operation. A processor with 16- bit would be able to complete the operation with single instruction.

Instruction Level Parallelism

A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. So we achieve concurrent execution of multiple instructions from a program. While pipelining is a form of ILP, we must exploit it to achieve parallel execution of the instructions in the instruction stream.

Example

```
for (i=1; i<=100; i= i+1)
    y[i] = y[i] + x[i];
```

This is a parallel loop. Every iteration of the loop can overlap with any other iteration, although within each loop iteration there is little opportunity for overlap.

Data Parallelism

Data Parallelism means concurrent execution of the same task on each multiple computing core.

Let's take an example, summing the contents of an array of size N. For a single-core system, one thread would simply sum the elements $[0] \dots [N - 1]$. For a dual-core system, however, thread A, running on core 0, could sum the elements $[0] \dots [N/2 - 1]$ and while thread B, running on core 1, could sum the elements $[N/2] \dots [N - 1]$. So the Two threads would be running in parallel on separate computing cores.

Task Parallelism

Task Parallelism means concurrent execution of the different task on multiple computing cores.

Consider again our example above, an example of task parallelism might involve two threads, each performing a unique statistical operation on the array of elements. Again The threads are operating in parallel on separate computing cores, but each is performing a unique operation.

Evolution of Computing : Parallel Computing terminologies

Parallel Computing Architecture

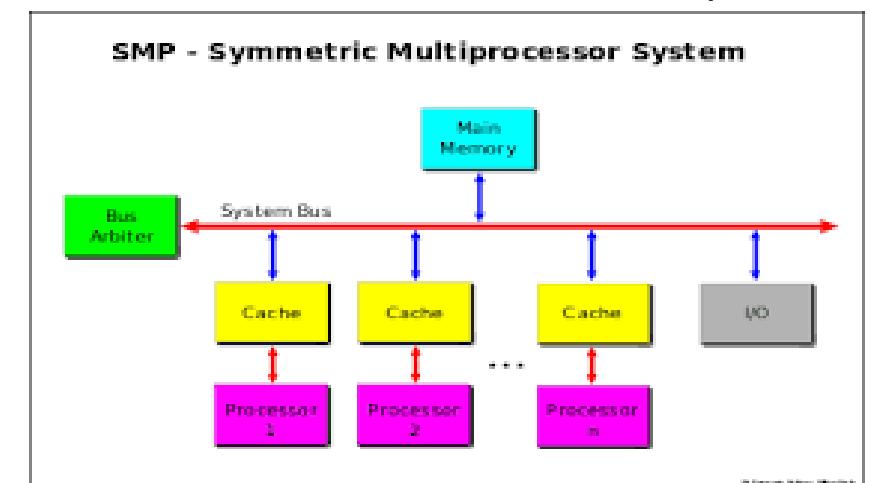
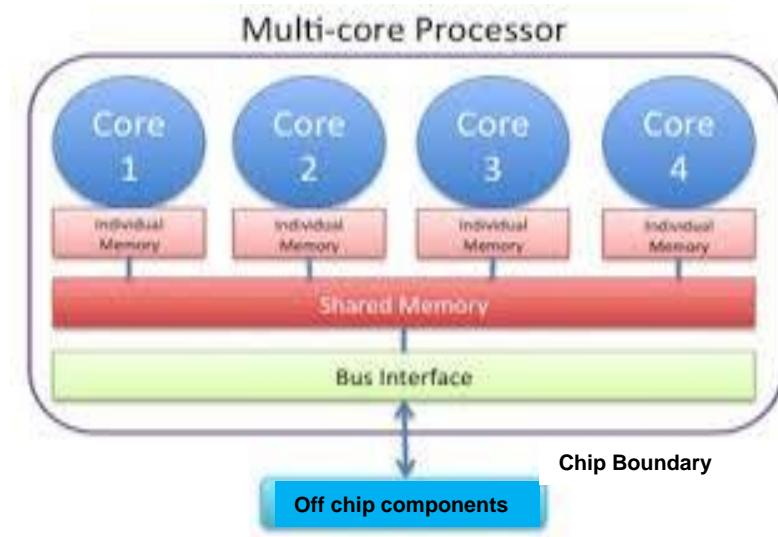
Multi-core computing: A multi-core processor is a computer processor integrated circuit with two or more separate processing cores, each of which executes program instructions in parallel.

Symmetric multiprocessing: multiprocessor computer hardware and software architecture in which two or more independent, homogeneous processors are controlled by a **single operating system** instance that treats all processors equally, and is connected to a **single, shared main memory** with full access to all common resources and devices

Each processor has a private cache memory, may be connected using on-chip mesh networks, and can work on any task no matter where the data for that task is located in memory.

Massively parallel computing: refers to the use of numerous computers or computer processors to simultaneously execute a set of computations in parallel.

One approach involves the grouping of several processors in a tightly structured, centralized computer cluster. Another approach is grid computing, in which many widely distributed computers work together and communicate via the Internet to solve a particular problem.



Parallel Computing Software Solutions and Techniques

- **Application checkpointing:** a technique that provides fault tolerance for computing systems by recording all of the application's current variable states, enabling the application to restore and restart from that point in the instance of failure.
- **Automatic parallelization:** refers to the conversion of sequential code into multi-threaded code in order to use multiple processors simultaneously in a shared-memory multiprocessor (SMP) machine.
- **Parallel programming languages:** Parallel programming languages are typically classified as either distributed memory or shared memory. While distributed memory programming languages use message passing to communicate, shared memory programming languages communicate by manipulating shared memory variables.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Introduction to Cloud Computing Terminologies (Cont.) Grid & Cloud Computing

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

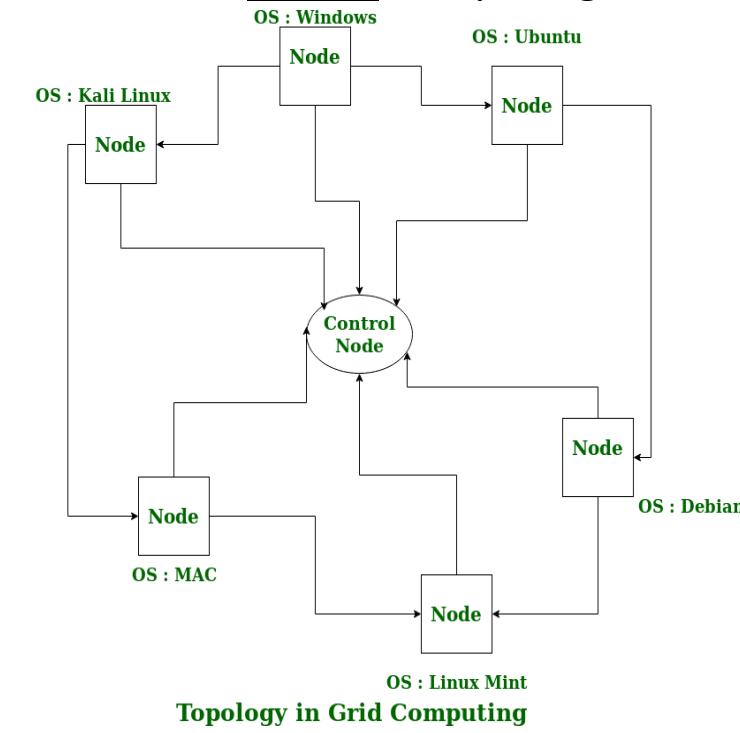
Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Evolution of Computing : Grid Computing

Grid computing is defined as “*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.*”

- Grid computing is the use of widely distributed computer resources to reach a common goal.
- A computing grid can be thought of as a distributed system with non-interactive workloads that involve many files.
- Grid computing is distinguished from conventional high-performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application.
- Grid computers also tend to be more heterogeneous and geographically dispersed (thus not physically coupled) than cluster computers
- Establishing **trust and security models** between infrastructure resources pooled from two different administrative domains become even more important.
- **Resource sharing agreements** needed to be formed among a set of participating parties where sharing meant **DIRECT ACCESS** to the resources.
- Sharing was **highly controlled** with resource providers and consumers grouped into virtual organizations primarily based on sharing conditions.



CLOUD COMPUTING

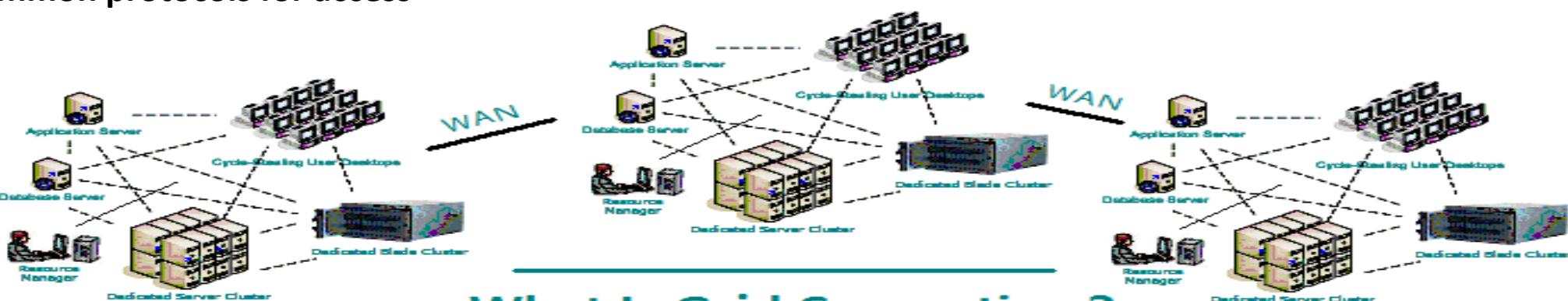
Grid Computing

Cloud computing is frequently compared to grid computing but here are the features of Grid which constrasts.

- The vision of grid computing is to enable **computing to be delivered as a utility**
- grid computing was meant to be used by individual users who gain access to computing devices **without knowing where the resource is located or what hardware it is running**, and so on.
- Initial technological focus of grid computing was **limited to enabling shared use of resources with common protocols for access**

Features of a Grid

1. Co-ordinates resources that are not subject to centralized control
2. Using standard, open, general purpose protocols and interfaces
3. To deliver non-trivial quality of service
4. Uses a common standard for authentication, authorization, resource discovery and resource access



What Is Grid Computing ?

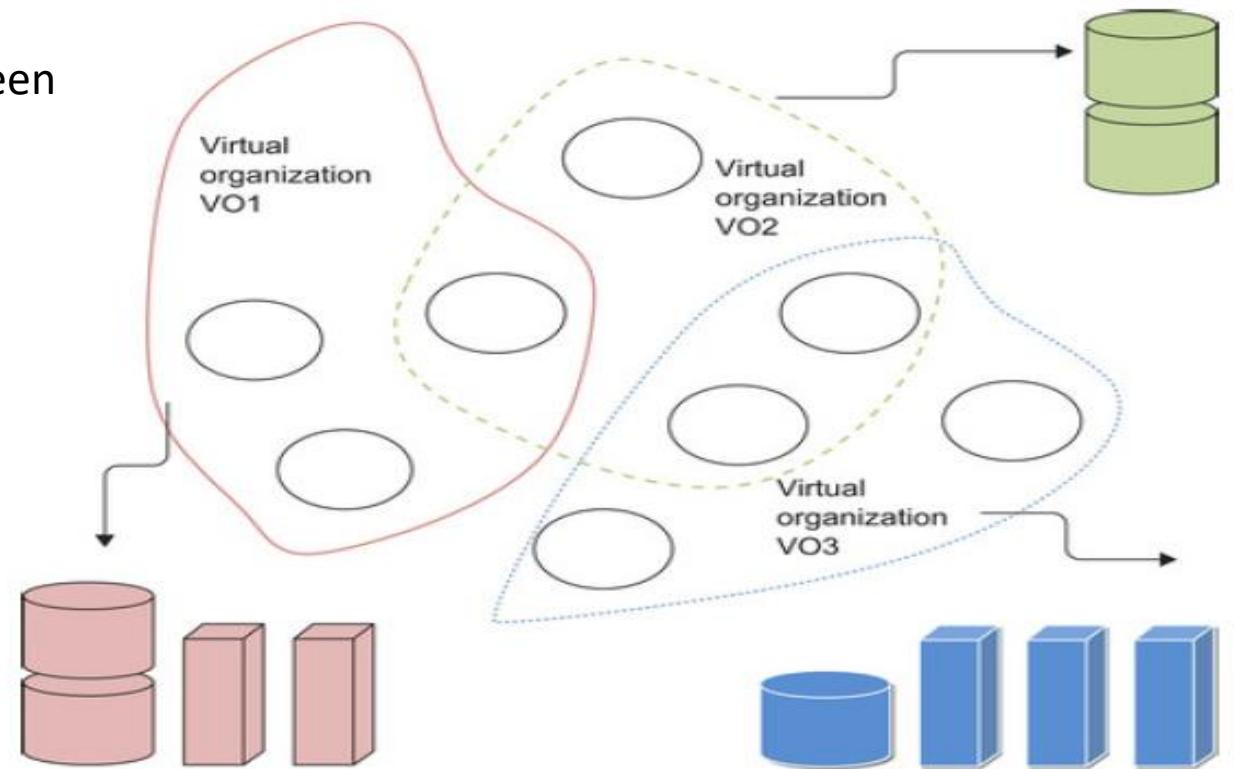
CLOUD COMPUTING

Grid Computing

- Uses the **Concept of Virtual Organization (VO)** which is basically a dynamic collection of individuals or institutions from multiple administrative domains is used to enable flexible, coordinated, secure resource sharing among participating entities (T2)
- A VO forms a **basic unit for enabling access to shared resources** with specific resource-sharing policies applicable for users from a particular VO
- Grid Technology enables sharing of resource between parties who may not have any trust earlier

Benefits

- Exploit Underutilized resources
- Resource load Balancing
- Virtualize resources across an enterprise
- Data Grids, Compute Grids
- Enable collaboration for virtual organizations



Grid Computing : Computational and Data Grid

“A **computational grid** is a **hardware** and **software** infrastructure that provides dependable, consistent, pervasive, and inexpensive access to **high-end computational capabilities**.”

Example : Science Grid (US Department of Energy)

A **data grid** is the storage component of a grid computing system that deals with the **controlled sharing and management of large amounts of distributed data**.

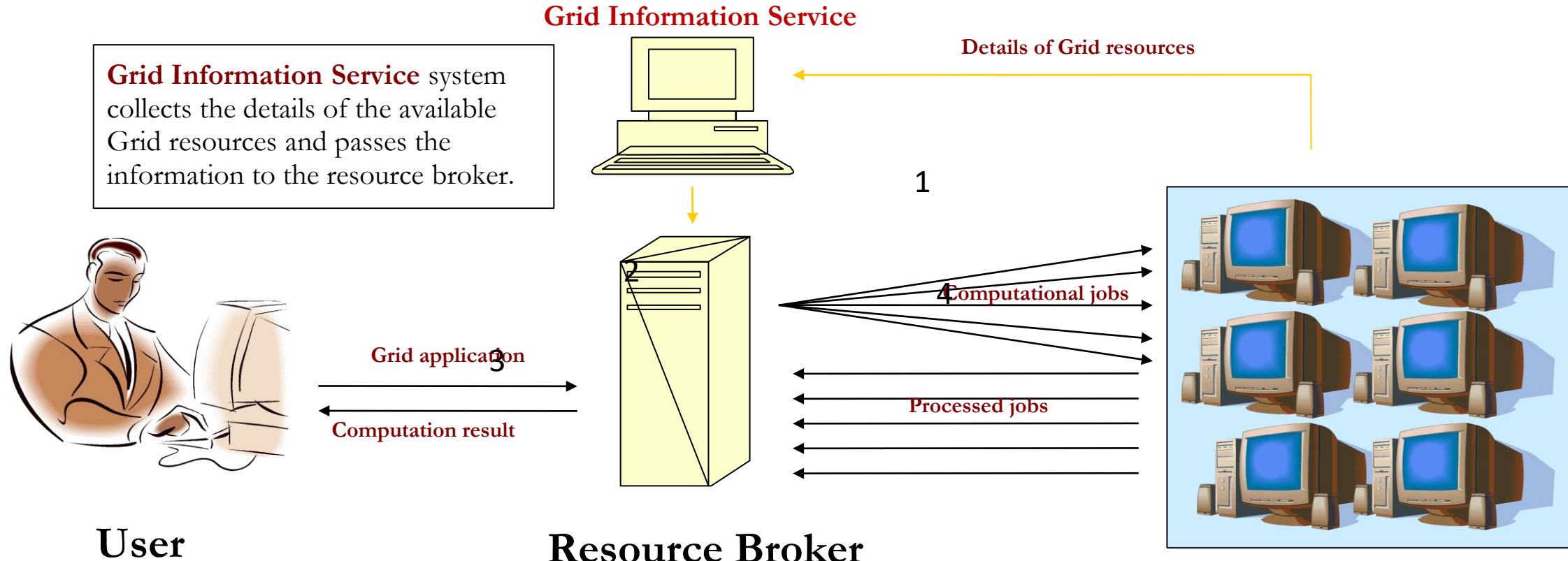
Scientific and engineering applications require access to large amounts of data, and often this data may be widely distributed. A data grid provides seamless access to the local or remote data required to complete compute intensive calculations.

Examples of Data Grid's :

Biomedical informatics Research Network (BIRN),

Southern California earthquake Center (SCEC).

Grid Computing : A typical view of the Grid environment



A **User** sends computation or data intensive application to Global Grids in order to speed up the execution of the application.

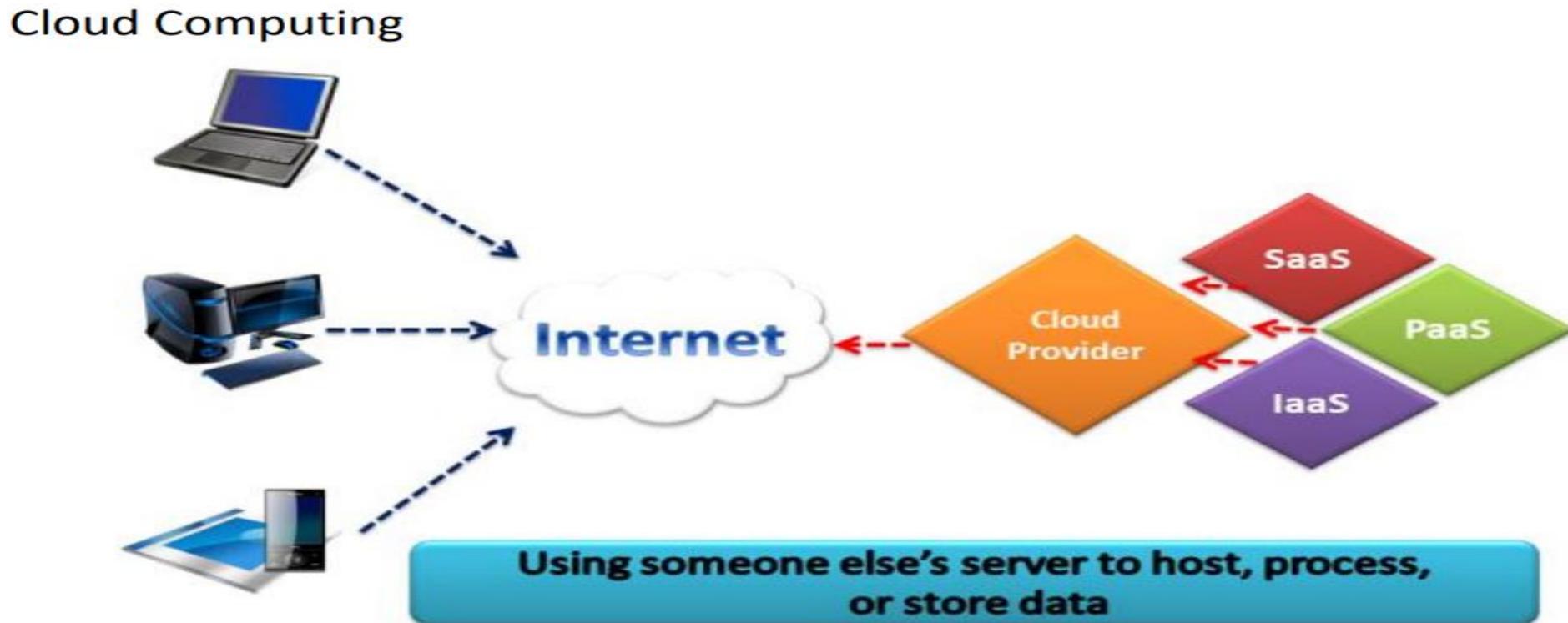
A **Resource Broker** distribute the jobs in an application to the Grid resources based on user's QoS requirements and details of available Grid resources for further executions.

Grid Resources (Cluster, PC, Supercomputer, database, instruments, etc.) in the Global Grid execute the user jobs.

Evolution of Computing : Cloud Computing

Cloud Computing

An Internet cloud of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed.



Evolution of Computing : Cloud Computing

Applications of Cloud Computing (T1 1.1.2.2)

Domain	Specific Applications
Science and engineering	Scientific simulations, genomic analysis, etc. Earthquake prediction, global warming, weather forecasting, etc.
Business, education, services industry, and health care	Telecommunication, content delivery, e-commerce, etc. Banking, stock exchanges, transaction processing, etc. Air traffic control, electric power grids, distance education, etc. Health care, hospital automation, telemedicine, etc.
Internet and web services, and government applications	Internet search, data centers, decision-making systems, etc. Traffic monitoring, worm containment, cyber security, etc. Digital government, online tax return processing, social networking, etc.
Mission-critical applications	Military command and control, intelligent systems, crisis management, etc.

Cloud Computing Design Objectives

Design objectives of cloud computing

- **Efficiency** measures the utilization rate of resources in an execution model by exploiting massive parallelism in HPC. For HTC, efficiency is more closely related to job throughput and power efficiency. All of these at lowest cost.
- **Dependability** measures the reliability and self-management from the chip to the system and application levels. The purpose is to provide high-throughput service with Quality of Service (QoS) assurance, even under failure conditions.
- **Adaptation** in the programming model measures the ability to support billions of job requests over massive data sets and virtualized cloud resources under various workload and service models.
- **Flexibility** in application deployment measures the ability of distributed systems to run well in both HPC (science and engineering) and HTC (business) applications.

Evolution of the Web (A key enabler in cloud computing)

Web 1.0

Four design essentials of a Web 1.0 site include:

1. Static pages.
2. Content is served from the server's file-system.
3. Pages built using Server Side Includes or Common Gateway Interface (CGI).
4. Frames and Tables used to position and align the elements on a page.

Web 2.0 –

Web 2.0 refers to world wide website which highlight user-generated content, usability & interoperability for end users. Web 2.0 is also called participative social web. Ajax & JavaScript frameworks extensively used in Web 2.0

Major features of Web 2.0 –

1. Permits users to retrieve and classify the information collectively.
2. Dynamic content that is responsive to user input.
3. Information flows between site owner and site users by means of evaluation & online commenting.
4. Developed APIs to allow self-usage, such as by a software application.

Web 3.0

1. Semantic Web

Generate, share and connect content through search and analysis based on **the ability to understand the meaning of words**, rather than on keywords or numbers.

2. Artificial Intelligence

Combining this capability with natural language processing, in Web 3.0,

3. 3D Graphics

The three dimensional design is being used extensively in websites and services in Web 3.0. Museum guides, computer games, ecommerce, geospatial contexts, etc. are all examples that use 3D graphics.

4. Ubiquity

Content is accessible by multiple applications, every device is connected to the web, the services can be used everywhere.

Contrasting Cloud Computing and Grid Computing

Cloud Computing	Grid Computing
Cloud Computing follows client-server computing architecture.	Grid computing follows a distributed computing architecture.
Scalability is high.	Scalability is normal.
Cloud Computing is more flexible than grid computing.	Grid Computing is less flexible than cloud computing.
Cloud operates as a centralized management system.	Grid operates as a decentralized management system.
In cloud computing, cloud servers are owned by infrastructure providers.	In Grid computing, grids are owned and managed by the organization.
Cloud computing uses services like IaaS, PaaS, and SaaS.	Grid computing uses systems like distributed computing, distributed information, and distributed pervasive.
Cloud Computing is Service-oriented.	Grid Computing is Application-oriented.
It is accessible through standard web protocols.	It is accessible through grid middleware.

Compute as an Utility (T1 1.1.2.3)

Utility computing focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers.

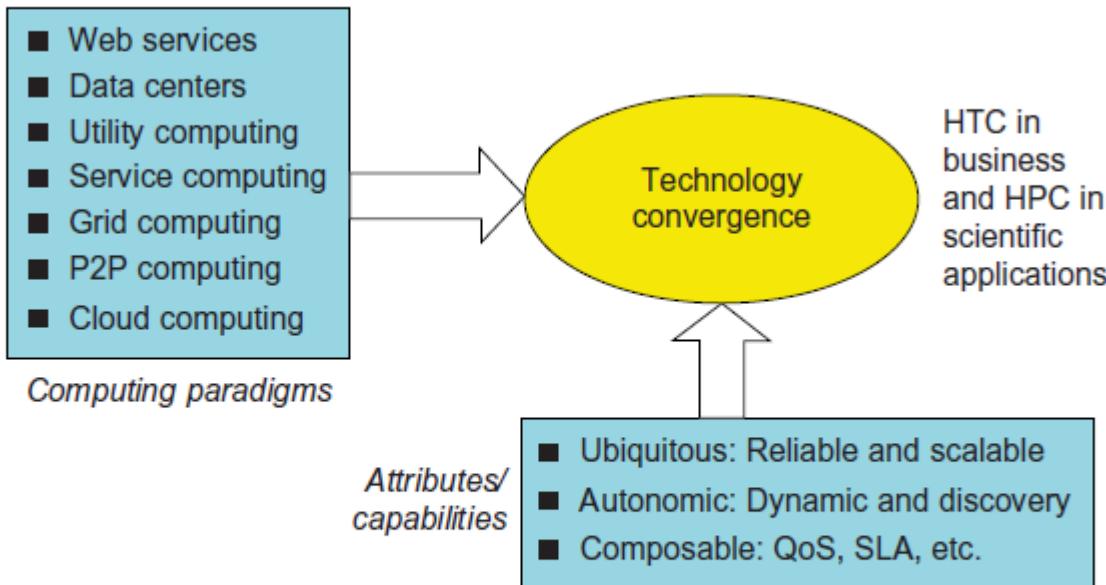


FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

(Modified from presentation slide by Raj Buyya, 2010)



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

CLOUD COMPUTING MODELS

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Cloud Computing Models

Recap: Cloud computing is a model for enabling ubiquitous , convenient, on demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction

- NIST (National Institute of Standards)

Key Terms

Ubiquitous - Wherever

On Demand - When

Shared Pool of configurable resources

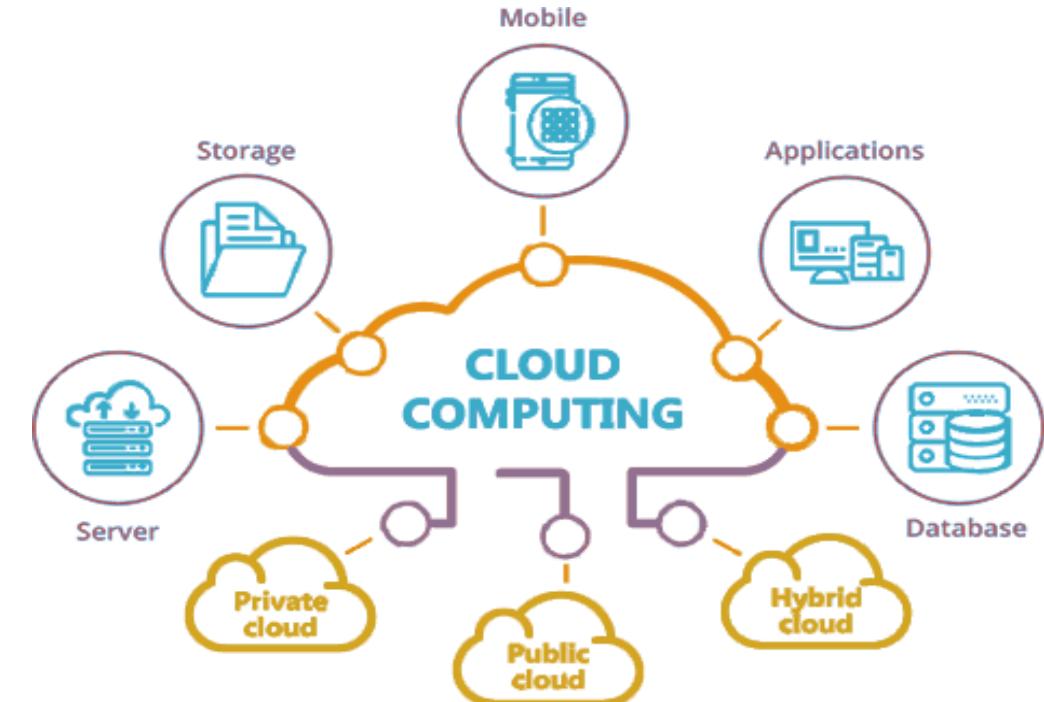
Rapidly Provisioned

Deployment Models

1. Private Cloud – where computing resources are deployed for one particular organization and are governed, owned and operated by the same organization.

2. Public Cloud – where the computing resource is owned, governed and operated by government, an academic or business organization and the resources are available for any individual/organization willing to pay

3. Hybrid Cloud where the cloud resources can be used for both type of interactions – B2B (Business to Business) or B2C (Business to Consumer). This deployment method is called hybrid cloud as the computing resources are bound together by different clouds.

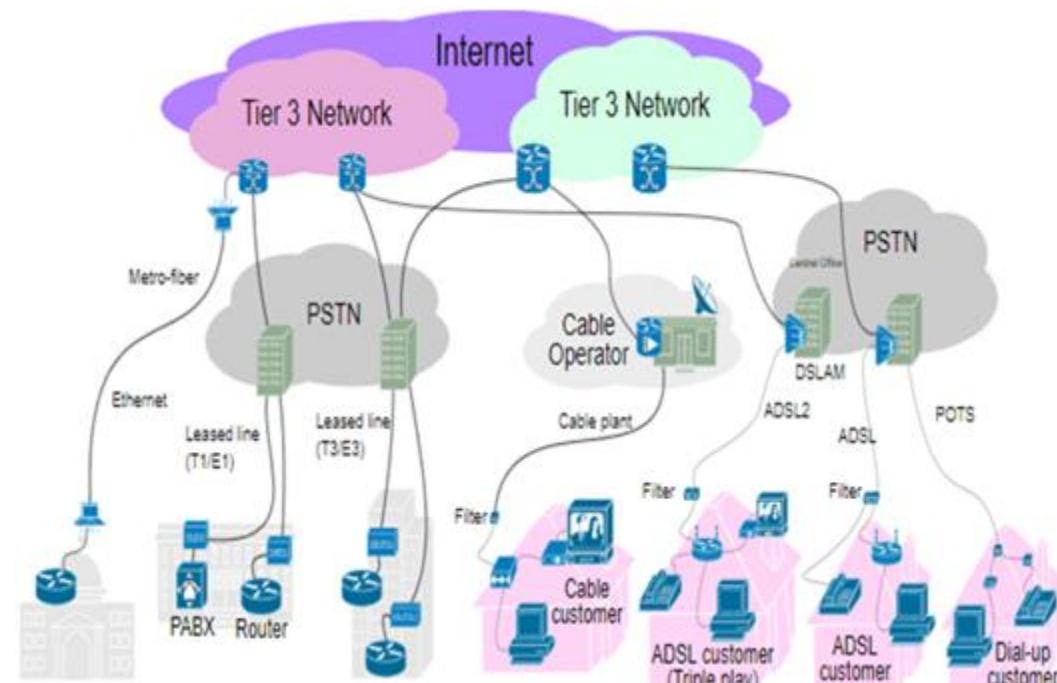


Cloud Computing Models : Enabling Technologies

Technologies that enable cloud computing

1. Broadband networks and internet architecture

1. All clouds must be connected to a network
2. Internet – can be visualized as a network of **autonomous co-operative** networks which are interconnected by routers
3. Core of the internet would be a connectionless packet switching Router based network. The edge devices like systems and desktops connected through switches and routers to local ISPs, which in turn are connected to regional ISPs and finally to a series of large backbone ISPs who interconnect through cables across ocean floors/countries.



2. Data center technology (# of Devices .. could be commodity devices forming the Infrastructure)

3. Virtualization technology (Server, Storage and Network)

4. Web technology (Uniform resource locator (URL), Hypertext transfer protocol (HTTP), Markup languages (HTML, XML))

5. Multitenant technology (Tenants on the same server)



Cloud Computing Models

There are typically 3 Cloud Service Models/Classifications (T2)

- **Infrastructure as a service (IAAS)**

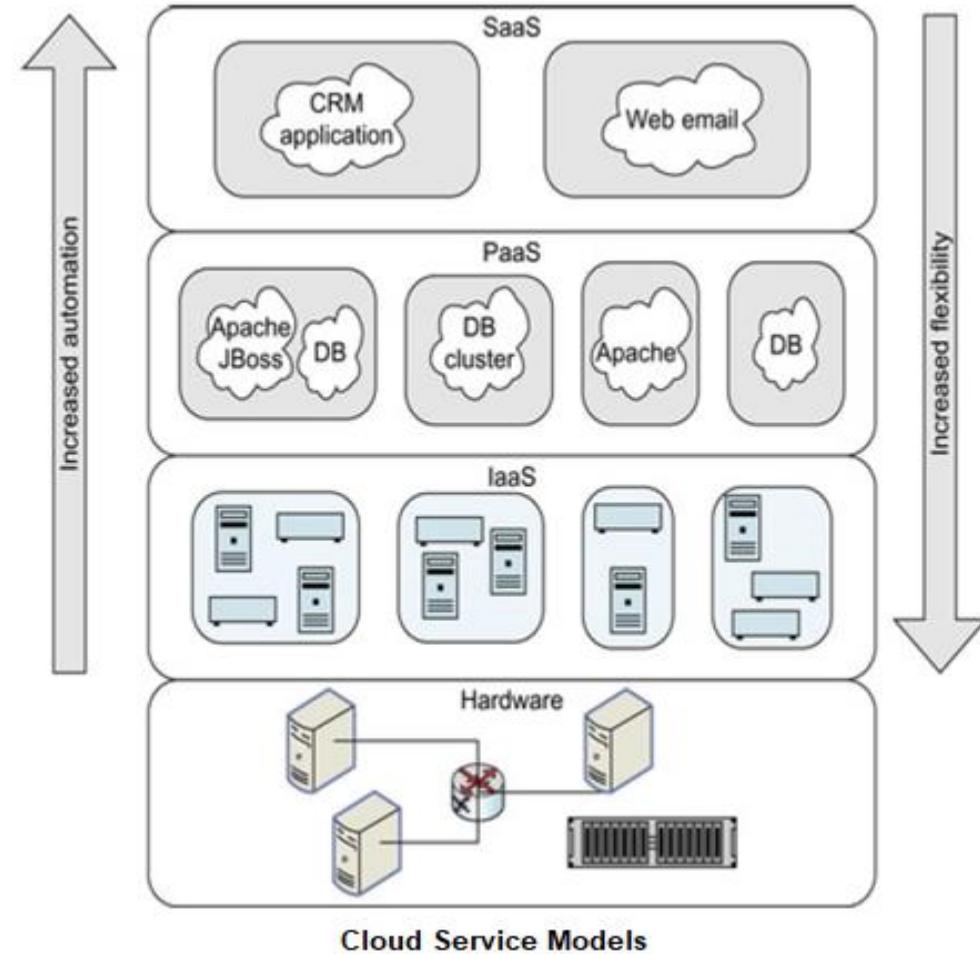
The physical hardware (servers, disks, and networks) is abstracted into virtual resources and allocated

- **Platform as a service (PAAS)**

Provides a platform built on top of the abstracted hardware that can be used by developers to create cloud applications
Commands provided allow allocation of middleware servers (e.g., a database of a certain size), configure and load data into the middleware

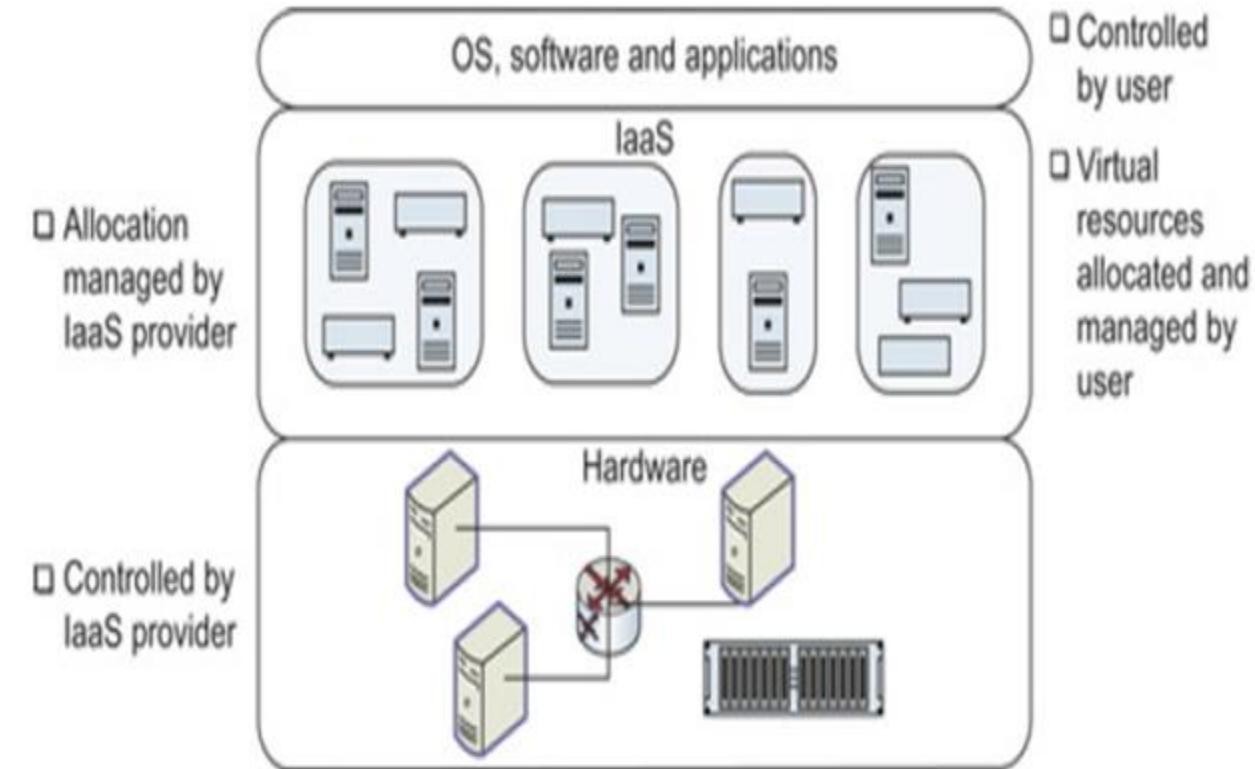
- **Software as a service (SAAS)**

Provides the complete application (or solution) as a service, enabling consumers to use the cloud without worrying about all the complexities of hardware, OS or even application installation



Cloud Computing Models – IaaS (Infrastructure as a Service)

- IaaS is a way to deliver a cloud computing infrastructure like server, storage, network and operating system.
- The customers can access these resources over cloud computing platform i.e Internet as an on-demand service.
- The customer buy's these cloud resources rather than buying space, server, software, or network equipment.
- The customer then deploys and runs arbitrary software, which can include operating systems and applications.
- The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls)
- The IaaS provider has control over the actual hardware and the cloud user can request allocation of virtual resources



Cloud Computing Models – IaaS (Infrastructure as a Service) (Cont.)

- The user of IaaS has single ownership of the hardware infrastructure allotted to him (may be a virtual machine)
- It is a Cloud computing platform based model.

Advantages of IaaS

- In IaaS, user can dynamically choose a CPU, memory storage configuration according to need.
- Users can easily access the vast computing power available on IaaS Cloud platform.
- IaaS is well suited for users who want complete control over the software stack that they run

Disadvantages of IaaS

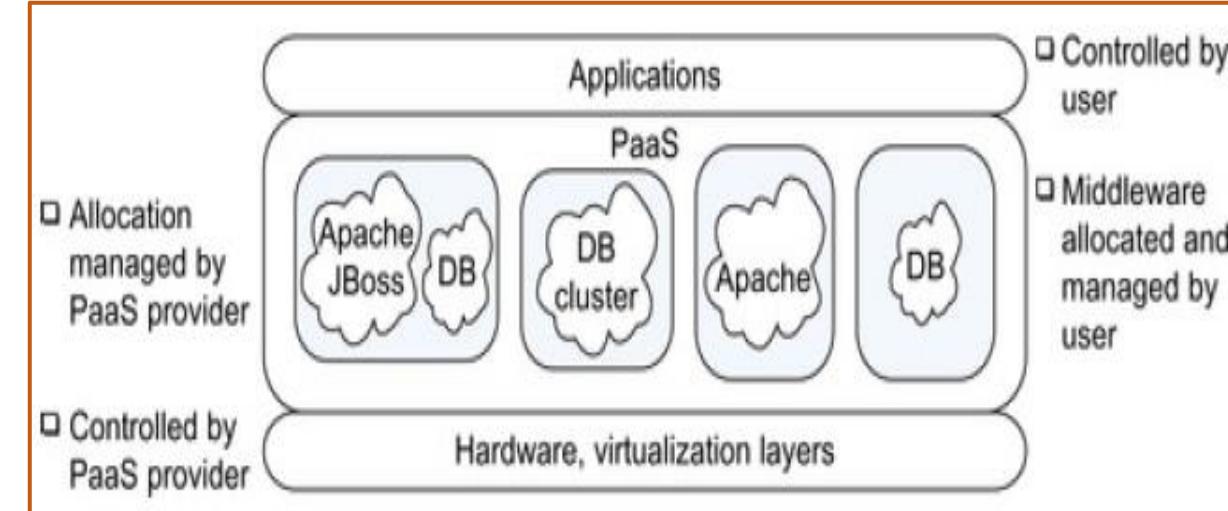
- IaaS cloud computing platform model is dependent on availability of Internet & virtualization services.

Examples of IaaS Service Providers

- Amazon Web Services.
- Microsoft Azure.
- Google Cloud.
- IBM Cloud.
- Oracle Cloud.

Cloud Computing Models – PaaS (Platform as a Service)

- The PaaS model is to provide a system stack or platform for application deployment as a service
- PaaS is a programming platform for developers for developing applications. This platform is generated for the programmers to create, test, run and manage the applications.
- The cloud user using the PaaS can configure and build on top of this middleware, such as define a new database table in a database
- A developer can easily write the application and deploy it directly into PaaS layer.
- Summarily the user does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment (say the database) configurations
- The hardware, as well as any mapping of hardware to virtual resources, such as virtual servers, is controlled by the PaaS provider. The PaaS provider also provides the supported middleware, such as a database, web application server



Cloud Computing Models – PaaS (Platform as a Service) (Cont.)

PaaS platforms are well suited to those cloud users who find that the middleware they are using matches the middleware provided by one of the PaaS vendors

Advantages of PaaS

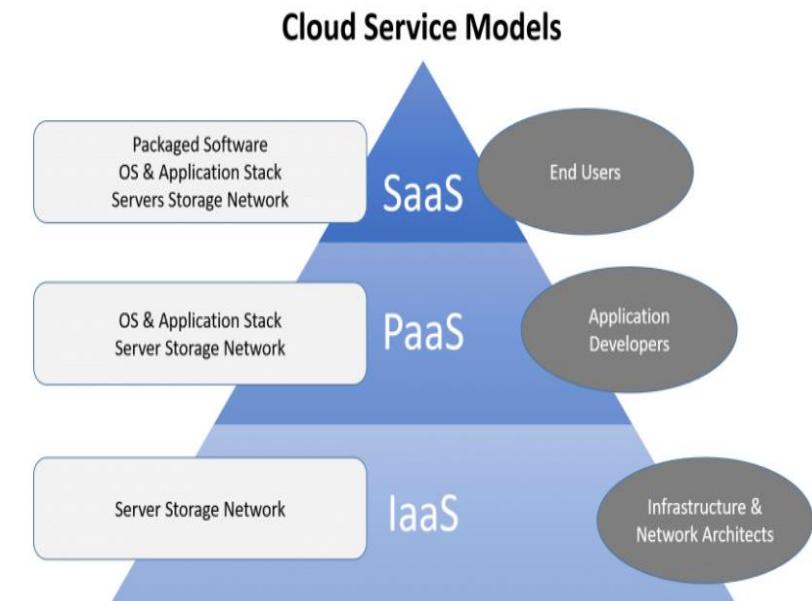
- PaaS platforms are well suited to those cloud users who find that the middleware they are using matches the middleware provided by one of the PaaS vendors
- PaaS is easier to develop. Developer can concentrate on the development and innovation without worrying about the infrastructure.
- In PaaS, developer only requires a PC and an Internet connection to start building applications.

Disadvantages of PaaS

- One developer can write the applications as per the platform provided by PaaS vendor hence the moving the application to another PaaS vendor is a problem.

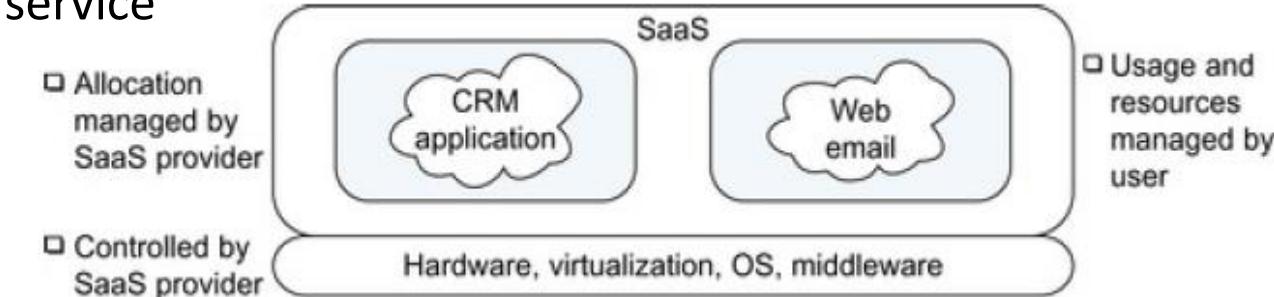
Examples

- Google Apps Engine(GAE), Windows Azure, SalesForce.com are the examples of PaaS.



Cloud Computing Models – SaaS (Software as a Service)

- SaaS is about providing the complete application as a service
- In this model, the provider's application running on a Cloud Infrastructure is publicized and would be used by the users over the Internet.
- The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email).
- In SaaS, associated data and software are also mostly hosted centrally on the cloud server and typically can be considered as a software distribution model
- The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.
- Users who log into the SaaS service can both use the application as well as configure the application for their use
- When configuration settings are changed, the SaaS infrastructure performs any management tasks needed (such as allocation of additional storage) to support the changed configuration



Cloud Computing Models – SaaS (Software as a Service) (Cont.)

Advantages of SaaS

- SaaS is easy to buy because the pricing of SaaS is based on monthly or annual fee and it allows the organizations to access business functionalities at a small cost, which is less than licensed applications.
- SaaS needed less hardware, because the software is hosted remotely, hence organizations do not need to invest in additional hardware.
- Less maintenance cost is required for SaaS and do not require special software or hardware versions.
- SaaS is almost a no programming Model

Disadvantages of SaaS

- SaaS applications are totally dependent on Internet connection. They are almost un-useable without Internet connection.
- It is difficult to switch amongst the SaaS vendors.

Examples

- CRM, Office Suite, Email etc. are software applications which are provided as a service through Internet.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

TECHNOLOGIES CHALLENGES & BUSINESS DRIVERS

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

CLOUD COMPUTING

Technology Challenges

- Cloud Computing as discussed earlier with so many advantages also has many challenges.
- These changes could come up from the infrastructure to support requests, or on account of the data-information in the cloud or it could be for effective usage of the same in Businesses
- We will look at few of these listed below
 - Scalability
 - Elasticity
 - Performance Unpredictability
 - Reliability and Availability
 - Security
 - Compliance
 - Multi-Tenancy
 - Lack of Expertise
 - Cost Management
 - Internet Connectivity
 - Password Security
 -
- These are challenges for all the 3 computing models

As and when the challenges occur these are getting addressed and these are active research areas

Technology Challenges : Scalability

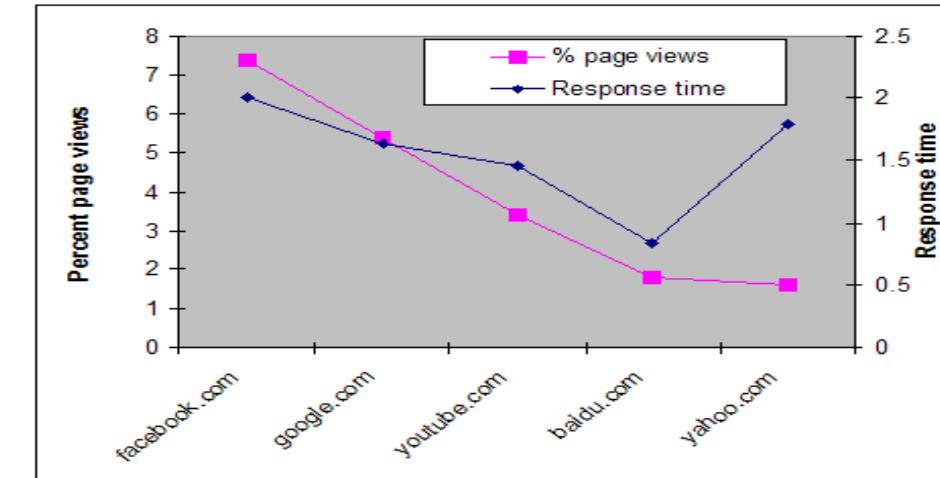
Recap - Scalability is the ability to accommodate larger or smaller loads while supporting some of the expectations of QoS like response time.

This will need to happen supporting

- Scale with the spread of wide range of environments
- Sharing of the same with many users
- Across significantly large computing environments

Consider as an example Facebook or Google,

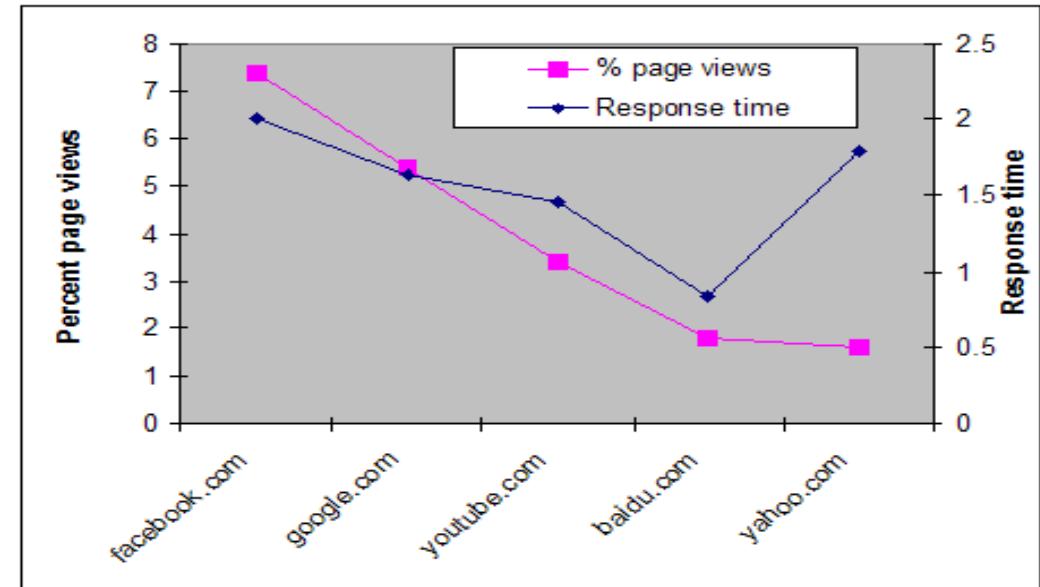
- Many Web sites have high traffic
 - Facebook – 7.5%
 - Google – 5.5%
- But they still maintain good response time
 - Facebook ~2 sec
 - Google ~1.7 sec
- Large amounts of storage
 - Google has many copies of Web
 - Facebook adds ~TB per day of storage



Recap - Elasticity is all about the actual increasing or decreasing of resources to cope with loads dynamically

Continuing with the example of Facebook or Google

- How to scale up (scale out) and scale down quickly?
 - Scale up (Vertical Scaling)
 - Scale Out (Horizontal Scaling)
 - Scale down
- Resource allocation and workload placement algorithms are required.
- Not happening quickly can impact QoS or Cost



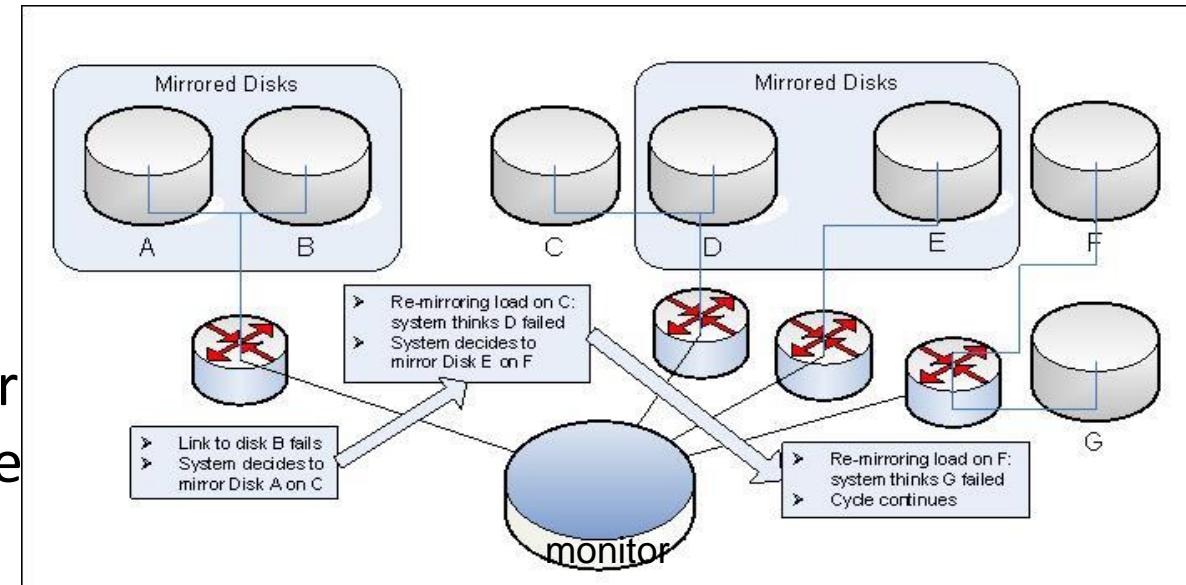
Technology Challenges : Performance Unpredictability

- Resources are shared
- How to guarantee performance isolation
 - Since physical hardware is shared.
 - Many VMs may run concurrently.
 - And affect each other at run time



- Most of the businesses are dependent on services provided by third-party, hence it is mandatory for the cloud systems to be reliable and robust.
- In large-scale environments, hardware failures and software bugs can be expected to occur relatively frequently.
- The problem is complicated by the fact that failures can trigger other failures, leading to an avalanche of failures that can lead to significant outages.
- Factors affecting reliability and availability
 - High number of components
 - Complexity

Eg. Consider diagram where few of the disks are mirrored and we have a process monitoring the load and behavior of disks



Availability

Service is important and many businesses run mission critical applications on the cloud

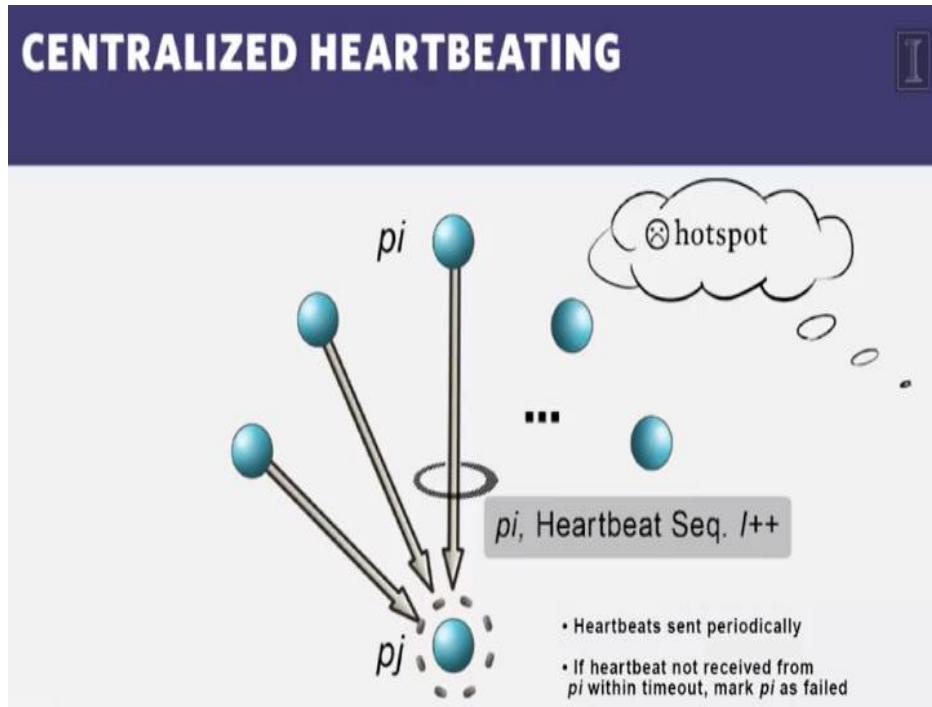
Availability is often achieved by using redundancy at the infrastructure , middleware or at the application level

There are 2 techniques that cloud models use for ensuring high availability

1. The first technique is **failure detection**, where the cloud infrastructure detects failed application instances, and avoids routing requests to such instances
2. The second technique is **application recovery**, where failed instances of application are restarted.

There are two techniques **of failure monitoring** [T2].

Failure Monitoring Heart Beats



The first method is **heartbeats**, where each application instance periodically sends a signal (called a heartbeat) to a monitoring service in the cloud. If the monitoring service does not receive a specified number of consecutive heartbeats, it may declare the application instance as failed.

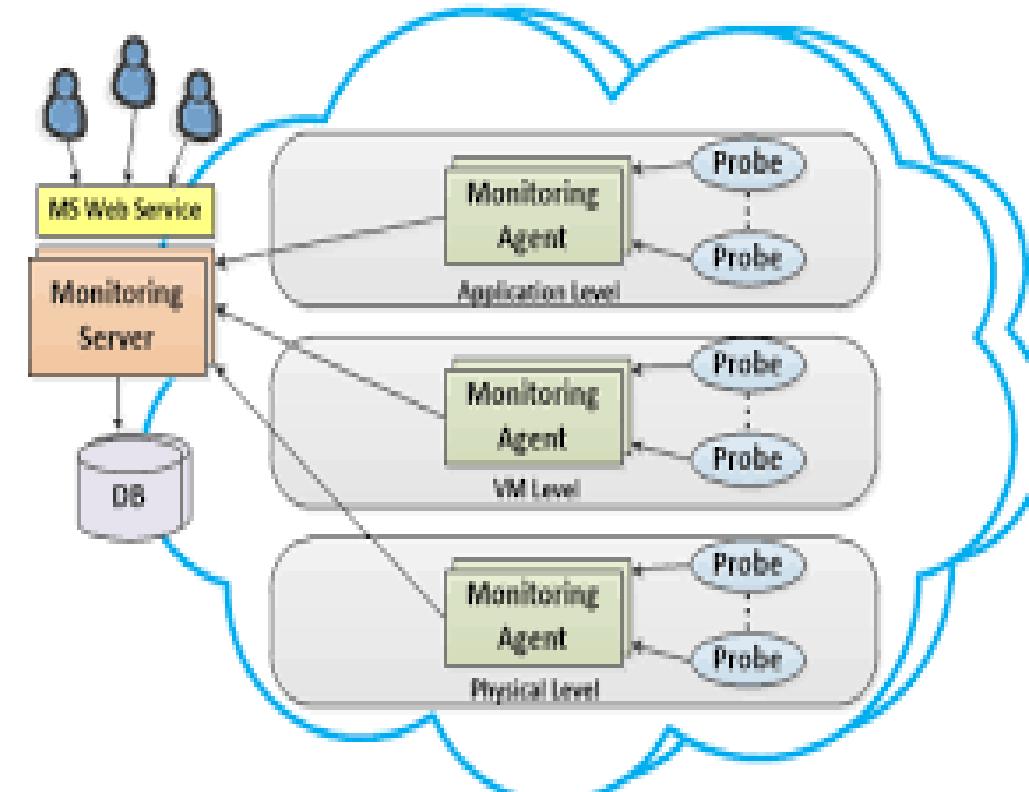
Probes

The second is the method of **probes**. Here, the monitoring service periodically sends a probe, which is a lightweight service request, to the application instance. If the instance does not respond to a specified number of probes, it may be considered failed.

There is a trade-off between speed and accuracy of detecting failures. To detect failures rapidly, it may be desirable to set a low value for the number of missed heartbeats or probes.

Recovery

Redirection: After identifying failed instances, it is necessary to avoid routing new requests to these instances. A common mechanism used for this in HTTP-based protocols is HTTP-redirection.



https://www.researchgate.net/figure/Abstract-Architecture-View-monitoring-metrics-from-the-Cloud-element-they-reside-on-and_fig1_271428939

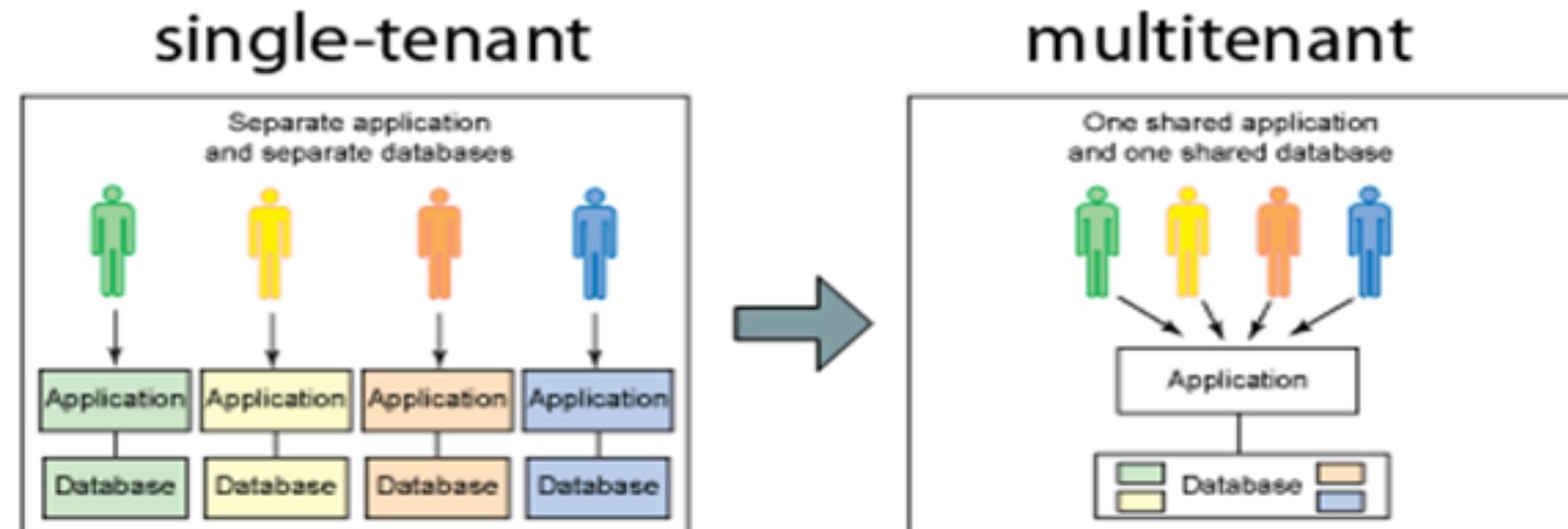
- Considerations towards violation of confidentiality, data privacy, and data leakage and loss
- Is security possible?
 - Data is getting stored and processes by a third party vendor whom you cannot see and a lot of times you are unaware of the location where its stored or processed
 - Vast amount of data will need to be travelling on networks
- Sony incident
 - Hacker used Amazon to hack Sony web site
 - Can we be secure if we are in same cloud as hacker?
- Isolation of users
- Legal and process issues
 - Physical security
- Cloud Service providers are providing auditable and safe identity management, access control procedures for authentication and authorization, Use firewalls, encryptions, privacy protocols, recovery policies, SLAs etc.

Technology Challenges : Compliance

- Compliance is the process of meeting the requirements of the service users or it could be the laws of the country.
- Cloud technologies will need to enable business operations to comply with the expectations of a customer or the laws of the land.
- The challenge for an user would be to know if a cloud provider is complying with privacy rules or the laws and for the Cloud service provider to be enabled by technology for compliance.
- E.g.
Healthcare organizations in the USA for **protecting and securing health Information** have to comply with HIPAA (Health Insurance Portability and Accountability Act).
- Sarbanes Oxley 2002 which sets the process and oversight mechanisms for **audit and reporting standards** for public companies to **prevent accounting frauds** post Enron and few scandals.
- India SEBI Clause 49 (**Corporate Governance Practices**)

Multitenancy is the mode of operation where a single instance of the component serves multiple tenants or groups of users. The instances (tenants) are logically isolated, but physically integrated and the architecture provides every tenant with a dedicated share of the instance, including configuration and data.

- Sharing of same database by multiple users
- Share without compromising security



Interoperability

- The application on one platform should be able to incorporate services from the other platform. This is known as Interoperability.
- It is becoming possible through web services, but to develop such web services is complex.

Portability (Migration)

- The applications running on one cloud platform can be moved to new cloud platform and it should operate correctly without making any changes in design, coding.
- The portability is not possible, because each of the cloud providers uses different standard languages for their platform.

Network Capability and Computing Performance

- High network bandwidth is needed for data intensive applications on cloud, this results in high cost.
- In cloud computing, low bandwidth does not meet the desired computing performance.

Application Recovery

- In addition to directing new requests to a server that is up, it is necessary to recover old requests
- An application independent method of doing this is **checkpoint/restart**. Here, the cloud infrastructure periodically saves the state of the application
- If the application is determined to have failed, the most recent checkpoint can be activated, and the application can resume from that state.
- Checkpointing is also present in certain middleware such as Dockers.
- Global and local snapshots are taken from which recovery is performed. There are off course challenges associated with Checkpointing.

CLOUD COMPUTING

Business Drivers

Benefits	Group	Drawbacks
Speed to Market	O P E R A T I O N A L	Security
Scalability		Compliance
Agility to new requests and need for capacity		Vendor Lock-In
Automated		
Flexibility to support changing scales		
Assurance – Assurance on account of the expertise and experience in technical and management of the Cloud Infrastructure by the specialized service provider		
Supports increase of Total Customer Experience		
Different Deployment Models like Private Cloud for Single Enterprise, Public Cloud for Service providers and users, Hybrid Cloud for a Mix and Commodity for a group		
Pay Per Use		
Lower Capital Investment		
Reduced Financial Risk	C O S T	
Reduced Cost		
Significantly Automated Management		
Expansion of the customer global footprint and thus Business		

Worldwide Public Cloud Revenue Forecast (Billions of US\$)

Table 1. Worldwide Public Cloud Service Revenue Forecast (Billions of U.S. Dollars)

	2017	2018	2019	2020	2021
Cloud Business Process Services (BPaaS)	42.2	46.6	50.3	54.1	58.1
Cloud Application Infrastructure Services (PaaS)	11.9	15.2	18.8	23.0	27.7
Cloud Application Services (SaaS)	58.8	72.2	85.1	98.9	113.1
Cloud Management and Security Services	8.7	10.7	12.5	14.4	16.3
Cloud System Infrastructure Services (IaaS)	23.6	31.0	39.5	49.9	63.0
Total Market	145.3	175.8	206.2	240.3	278.3

BPaaS = business process as a service; IaaS = infrastructure as a service; PaaS = platform as a service; SaaS = software as a service

Note: Totals may not add up due to rounding.

Source: Gartner (September 2018)

Comparison of public vs private cloud

Table 1.1 Hypothetical Cost of Public Vs Private Cloud

(in USD)	Private Cloud			Public Cloud		
	Year 1	Year 2	Year 3	Year 1	Year 2	Year 3
Hardware	70,000	40,000	20,000			
Setup Costs	30,000			5,000		
Software (Licensing)	200,000	400,000	700,000			
Labor costs	200,000	200,000	200,000			
Service costs				300,000	600,000	1,000,000
WAN costs				15,000	30,000	56,000
Cost for year	500,000	640,000	920,000	305,000	600,000	1,000,000
Total	2,060,000			2,006,000		



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

**Public, Private and Hybrid Cloud
Cloud Deployment Models**

Dr. H.L. Phalachandra

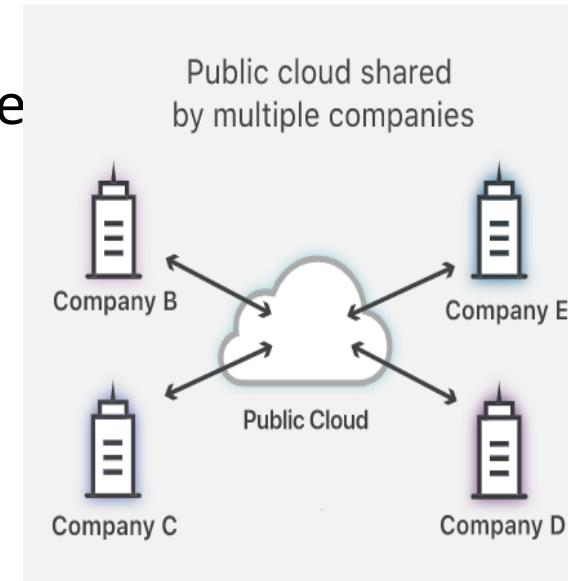
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Public Cloud: It's a platform or an IT model where a set of infrastructure resources located in one or more different physical locations are made available as a service in a pay-as-per-use model to multiple general public customers by a cloud service provider.

- The infrastructure resources are provided as a service, say as software (SaaS), as software platforms on which applications can be built and run (PaaS) or as virtual IT components like compute or storage (IaaS)
- Any user who wants to use it is provided with this shared public platform accessible through the internet
- The cloud user pays the cloud vendor for using the infrastructure.
- The resources of the public cloud is owned and managed by the cloud vendor or also called service provider.
- In a public cloud, these are made available through APIs or through a self-service interface.



A public cloud is like renting an apartment. It's public from the perspective that the resources are publicly available to anyone, available to be used on a cost basis and is available on the public Internet.

Characterizing this a little more :

1. **Shared Resource Allocation** – Users or tenants outside the provider's firewall share cloud services and virtual resources that come from the provider's set of infrastructure, platforms, and software.
2. **Usage Agreements** – Some of the resources available could be costed and may cost based on the usage in the pay-as-you-use, there could be some which may be available at no cost. (Example: Massachusetts Open Cloud).
3. **Management** - At a minimum, the provider maintains the hardware underneath the cloud, supports the network, and manages the virtualization software.

Advantages/Motivations for a Public Cloud

1. Cost :

- Public cloud has a **lower** cost than private, or hybrid cloud, as it shares the same infrastructure resources with a large number of consumers and thus gets the benefit of the cost getting distributed
- Public cloud service providers given their volumes and specialization, would handle it more efficiently
- Public cloud costs are also **expense** thus reducing the need for significant investments.

2. Less server management - If an organization uses a public cloud, internal teams don't have to spend time managing servers – as they do for legacy on-premises data centers or for internal private clouds.

3. Time Saving - The cloud service provider is responsible for the management and maintenance of data centers, the client can save the time required to establish connectivity, deploy new products, release product updates, configure servers etc.

4. Location Independence – Given that the resources can exist anywhere, it provides location independence

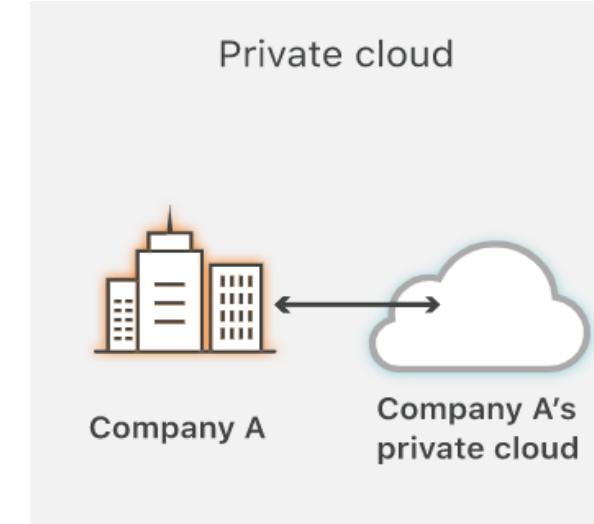
5. Analytics - Public cloud services can perform analytics on high volumes and accommodate a variety of data types to present business insights.

6. Virtually unlimited scalability - Cloud capacity & resources rapidly expand to meet user demands and traffic spikes. Users also achieve greater redundancy and high availability due to the providers' various, logically separated cloud locations.

Disadvantages of Public Clouds

1. **Security** - Verification of the security of data arises as a concern in public clouds, since the data is not being stored by the enterprise. Cloud service providers have attempted to address this problem by acquiring third-party certification.
Multitenancy adds to the security concern due to the probability of data leakage
2. **Compliance** - Many companies might question if the cloud provider is complying with the security rules relating to data. Cloud service providers have attempted to address these issues through certification as well.
3. **Interoperability and vendor lock-in** - Once a particular public cloud has been chosen, it would not be easy to migrate away, since the software and operating procedures would all have been tailored for that particular cloud.
4. Users may be **limited** in terms of the control on the **Infrastructure configurations**

- This is also referred to as an internal or a corporate cloud.
- The private cloud model utilizes the in-house or proprietarily hosted infrastructure to host the different cloud services.
- It enjoys the benefits of cloud computing—including elasticity, scalability, and ease of service delivery—with the access control, security, and resource customization of on-premises infrastructure
- Private cloud is a computing model that offers a proprietary environment dedicated to a single business entity.
- Private cloud could be considered as single tenant from business perspective but could have multiple users using it from within the single business entity and within the intranet cloud
- A private cloud strategy may be comprised of hardware hosted locally at a facility owned by a business, or it may be hosted by a cloud service provider.
- Virtual private clouds are typically paid for on a rolling basis, but provisioned hardware and storage configurations maintain the benefits of a secure, exclusive network.



Internal vs Hosted Private Cloud

Internal Private Cloud



Hosted Private Cloud



Best Private Cloud Providers.

HPE. By most estimates, Hewlett Packard Enterprise (HPE) is a key leader in the private cloud market. ...

VMware. ...

Dell. ...

Oracle. ...

IBM / Red Hat. ...

Microsoft. ...

Cisco.

Advantages/Motivations for a Private Cloud

What motivates organizations to use private cloud?

- 1. More Control** - Private clouds have more control over their resources and hardware than public clouds because it is only accessed by a single organization.
- 2. Security** - Provides the enhanced security of dedicated, physically isolated network, compute and storage.
- 3. Compliance** - For businesses operating in heavily regulated industries, this provides organizations the ability to comply with strict regulations because sensitive data is held on hardware that cannot be accessed by anyone else, thus enabling for necessary policies to support compliance.
- 4. Customization** - Private clouds are fully configurable by the organization, thus by modelling the environment as a Private Cloud enables designing the infrastructure that is best suited for the organizations needs for both Agility and performance.

Hosted private clouds offer the same advantages but require no on-site setup.

Disadvantages of Private Clouds

1. **Cost** - With exclusivity comes increased cost. If you plan to build your own private cloud, you face a large capital outlay.
2. **Under-utilization** - With a private cloud, the cost of capacity underutilization is a cost to you, not to your provider. Therefore managing and maximizing utilization becomes your concern.
3. **Platform scaling** - Large upward changes in your requirements are likely to require scaling of the physical infrastructure. This is fine but may take longer than simply scaling a virtual machine within existing capacity.

What factors influence whether an organization use a private cloud or public cloud?

1. Infrastructure -

- The private cloud model utilizes the in-house infrastructure to host the different cloud services. The cloud user here typically owns the infrastructure. Scaling of the infrastructure is inflexible and incurs capital cost
- The infrastructure for the public cloud on the other hand, is owned by the cloud vendor. The cloud user pays the cloud vendor for using the infrastructure. Public cloud is much more amenable to provide elasticity and scaling-on-demand since the resources are shared among multiple users

2. Network bandwidth constraints and cost - Disruptions in the connectivity between the client and the cloud service will affect the availability of cloud-hosted applications. On a low bandwidth network, the user experience for an interactive application may also get affected. If the storage is intended to be used for a longer term, then it may be more cost-effective to buy storage and compute and use it as a private cloud. Thus, it can be seen that one of the factors dictating the use of a private cloud or a public cloud for storage is how long the resources are needed.

3. Control and Security - Some businesses may prefer to use a private cloud, especially if they have extremely high security standards. Using a private cloud eliminates intercompany multitenancy (there will still be multitenancy among internal teams) and gives a business more control over the cloud security measures that are put in place.

Public Cloud vs Private Cloud

Public Cloud	Private Cloud
Cloud Computing infrastructure shared to public by service provider over the internet. It supports multiple customers i.e, enterprises.	Cloud Computing infrastructure shared to private organisation by service provider over the internet. It supports one enterprise.
Multi-Tenancy i.e, Data of many enterprise are stored in shared environment but are isolated. Data is shared as per rule, permission and security.	Single Tenancy i.e, Data of single enterprise is stored.
Cloud service provider provides all the possible services and hardware as the user-base is world. Different people and organization may need different services and hardware. Services provided must be versatile.	Specific hardware and hardware as per need of enterprise are available in private cloud.
It is hosted at Service Provider site.	It is hosted at Service Provider site or enterprise.
It is connected to the public internet.	It only supports connectivity over the private network.
Scalability is very high, and reliability is moderate.	Scalability is limited, and reliability is very high.
Cloud service provider manages cloud and customers use them.	Managed and used by single enterprise.
It is cheaper than private cloud.	It is costlier than public cloud.
Security matters and dependent on service provider.	It gives high class of security.
Performance is low to medium.	Performance is high.
It has shared servers.	It has dedicated servers.
Example : Amazon web service (AWS) and Google AppEngine etc.	Example : Microsoft KVM, HP, Red Hat & VMWare etc.

Public Cloud vs Private Cloud

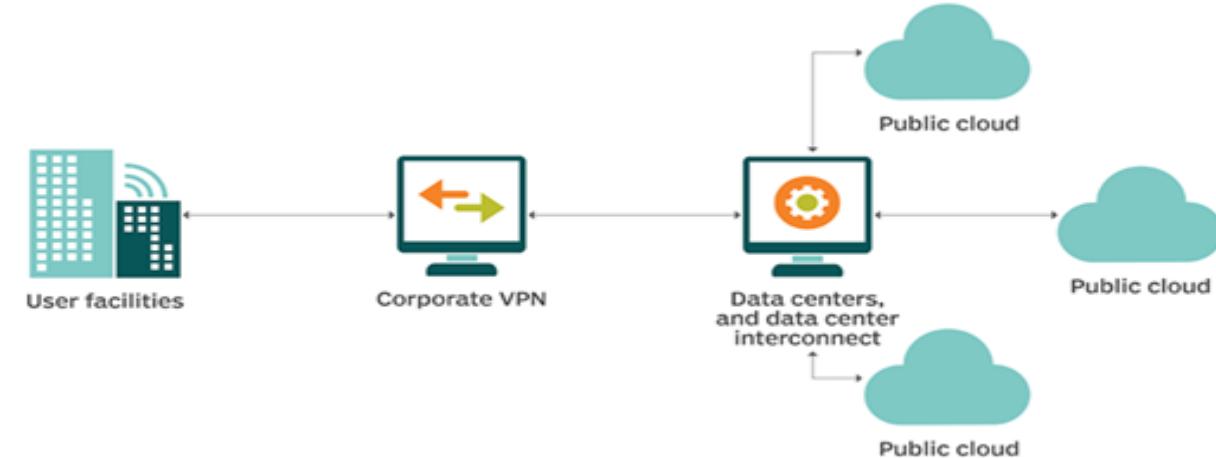
Best Suited for

Public Clouds provide **affordable solutions that offer room for growth**. It is thus ideal for application testing and cloud disaster recovery for small scale companies.

Private Clouds are for **high-performance security and customizable options**. It is, therefore, suitable for protecting sensitive data and applications.

Hybrid Cloud

- A hybrid cloud is a cloud computing environment that uses a mix of on-premises, private cloud and third-party, public cloud services with orchestration between these platforms.
- It combines these Public and Private clouds to create a unified, automated, and well-managed computing environment.
- A hybrid cloud model allows enterprises to deploy workloads in private IT environments or public clouds and move between them as computing needs and costs change.
- Typically non-critical activities are performed by the public cloud and critical activities are performed by the private cloud. This gives business'es greater flexibility and more data deployment options.
- A hybrid cloud workload includes the network, hosting and web service features of an application.



Public Cloud vs Private Cloud

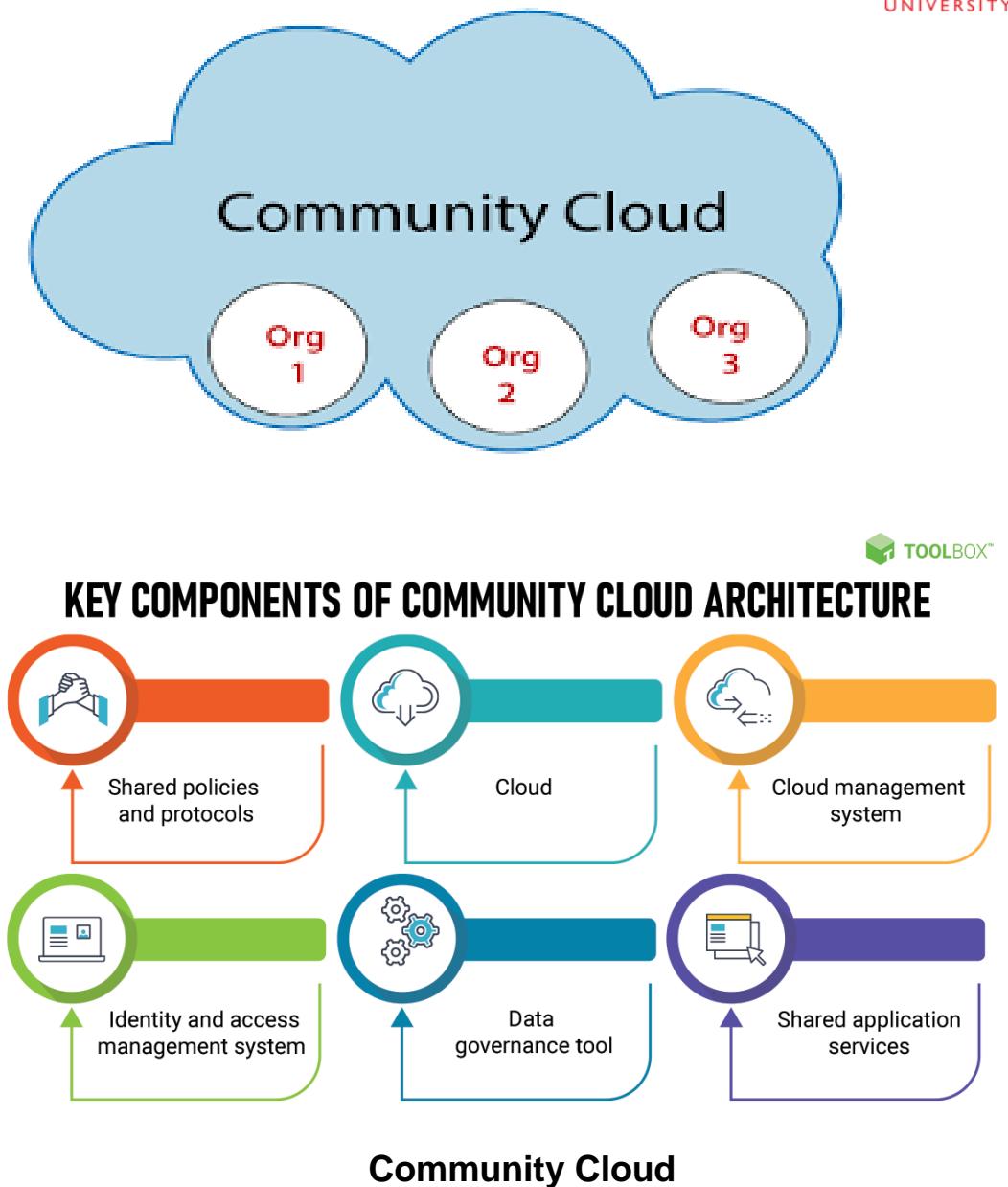
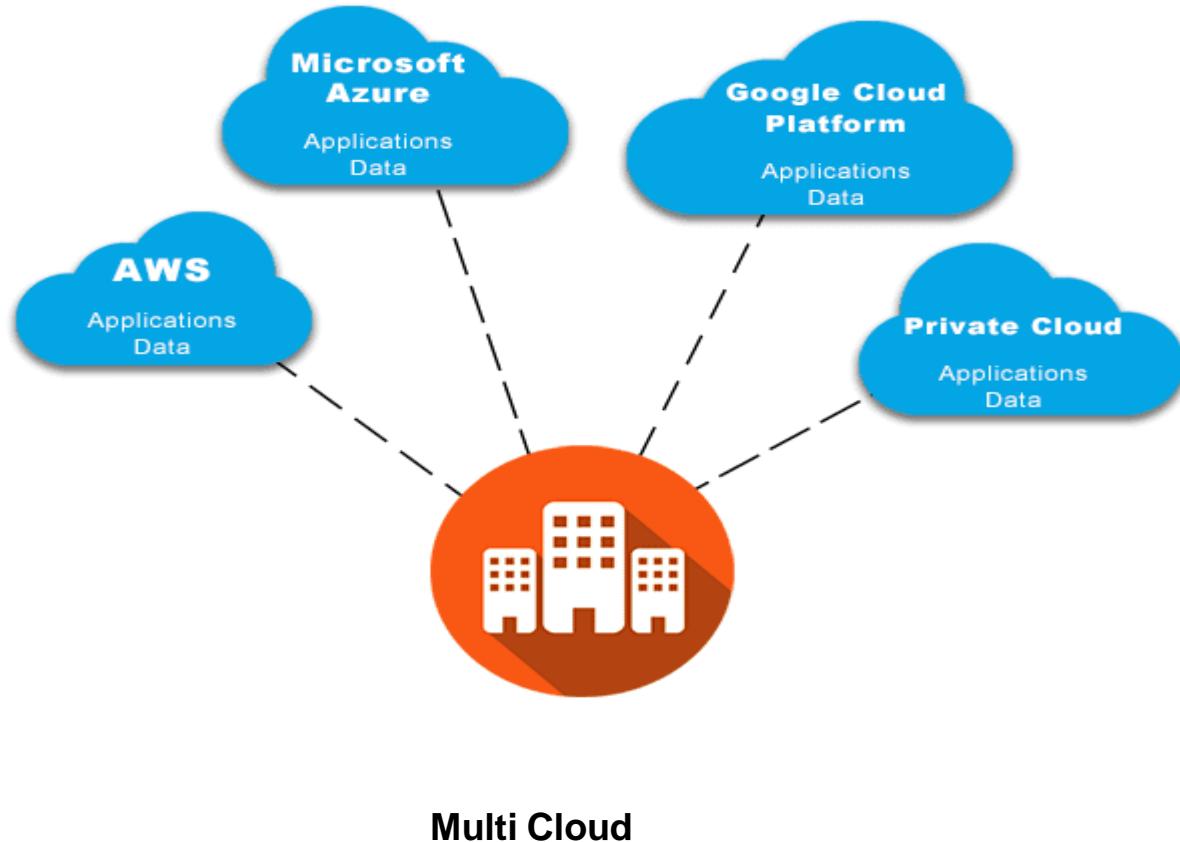
Public Cloud	Private Cloud	Hybrid Cloud
No maintenance costs	Dedicated, secure	Policy-driven deployment
High scalability, flexibility	Regulation compliant	High scalability, flexibility
Reduced complexity	Customizable	Minimal security risks
Flexible pricing	High scalability	Workload diversity supports high reliability
Agile for innovation	Efficient	Improved security
Potential for high TCO	Expensive with high TCO	Potential for high TCO
Decreased security and availability	Minimal mobile access	Compatibility and integration
Minimal control	Limiting infrastructure	Added complexity

Benefits

Drawbacks

CLOUD COMPUTING

Community Cloud & Multi Cloud





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Distributed System Models

Dr. H.L. Phalachandra

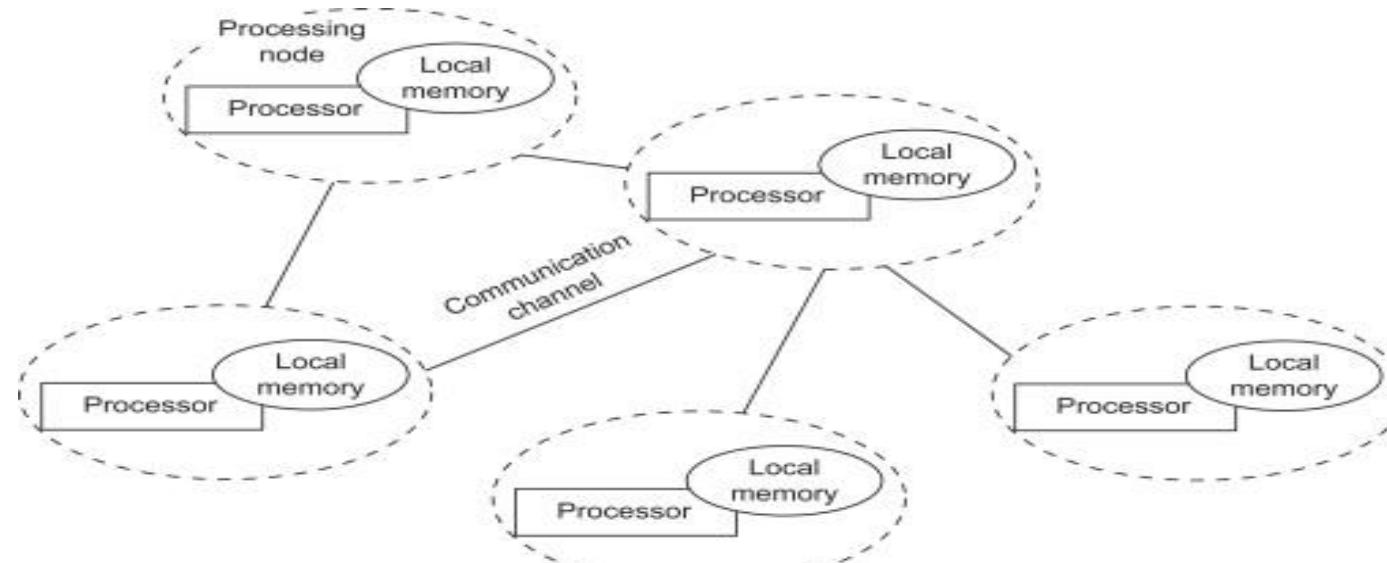
Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

- A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network.
- Information exchange in a distributed system is accomplished through message passing.
- Distributed and cloud computing systems are built over a large number of autonomous computer nodes.
- These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner.

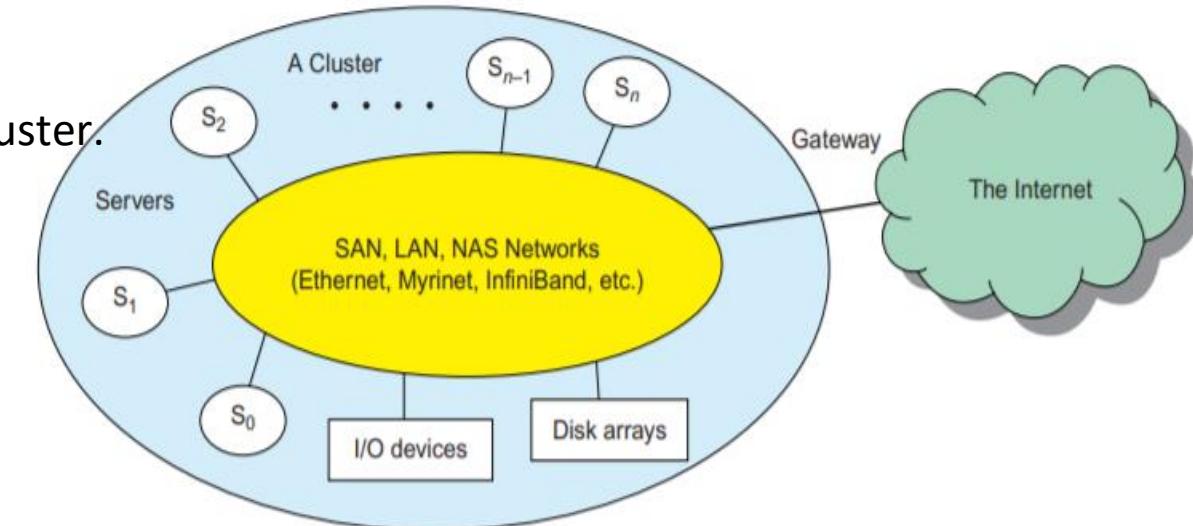
Eg. Scalar Multiplication of a matrix



What is Cluster?

Cluster as a low-latency, high-bandwidth interconnected network of standalone computers which work cooperatively as a single integrated computing resource

- A typical cluster architecture of distributed systems would show a number of hierarchically organized set of compute nodes connected by SAN/LAN or WAN as in the figure below.
- All nodes in a cluster are set to perform the same task with the resources of the node being managed by their own OS
- This cluster supports Scaling by increasing the number of nodes in the hierarchical organization of the systems
- The cluster is connected to the Internet via a virtual private network (VPN) gateway which provides secure connectivity using an encryption.
- The VPN gateway IP address helps to identify and locate the cluster.
- Most clusters will have multiple system images as a result of having many autonomous nodes under different OS control.
- Clustering could be with the focus of improving performance, supporting availability and error handling etc.



- In Cloud computing, resource requests are provisioned into system components which are configured and available as a distributed system
- Depending on the needs of the application, the system components which are part of the cloud infrastructure can be structured and classified as three different Distributed System Models

1. Architectural Models

- a. System Architecture
 - This indicates how the components of a distributed system are placed across multiple machines
 - How the responsibilities are distributed across system components
 - E.g. P2P Model or Client Server Model
- b. Software architecture
 - This indicates the logical organization of software components and their interactions/independence
 - Focusses about the components
 - E.g. 3 Tier Architecture

2. Interaction Models

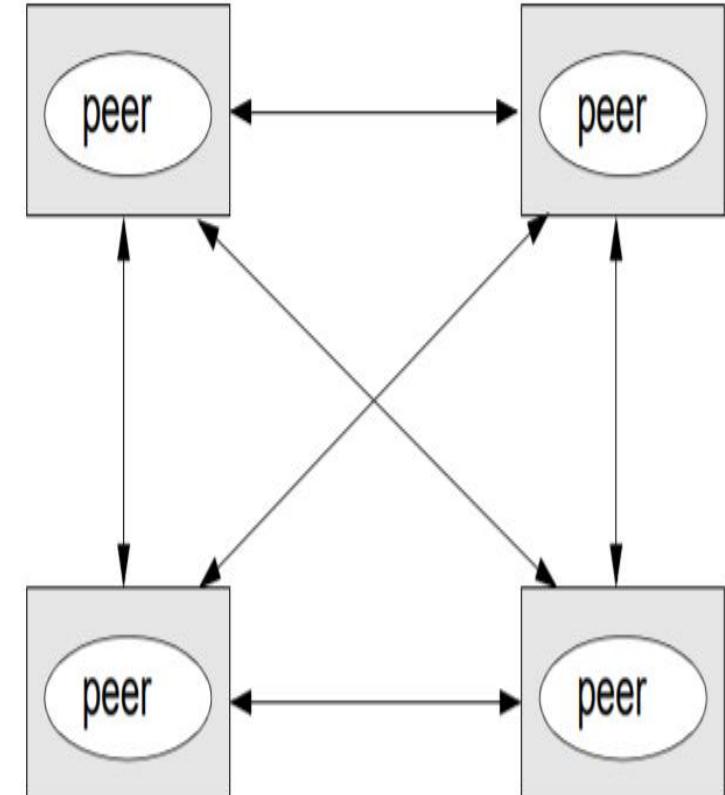
- a. How do we handle time? Are there time limits on process execution, message delivery, and clock drifts?
- b. Ex: Synchronous distributed systems, Asynchronous distributed systems

3. Fault Models

- - a. What kind of faults can occur and what are their effects?
 - b. Ex: Omission faults, Arbitrary faults, Timing faults

What does a Peer-to-Peer System mean?

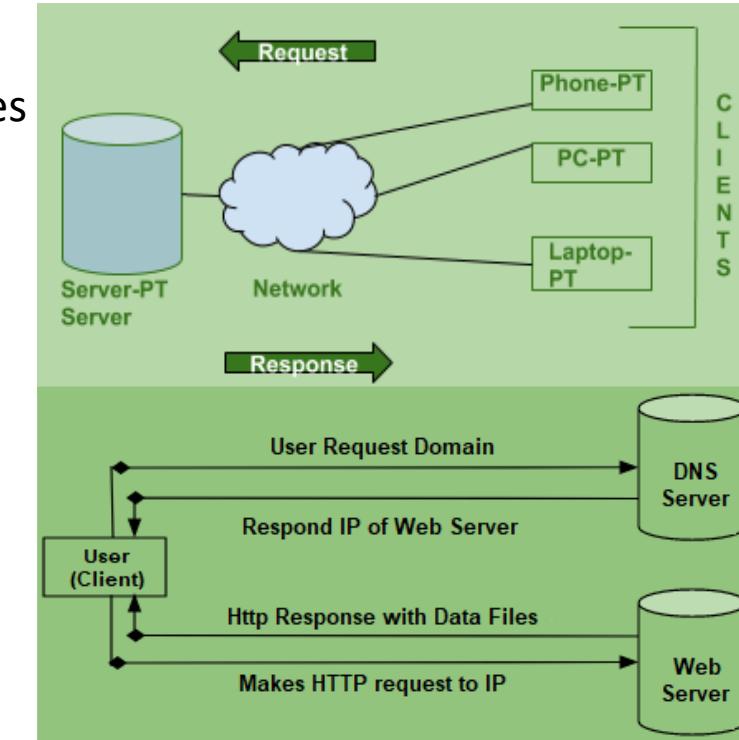
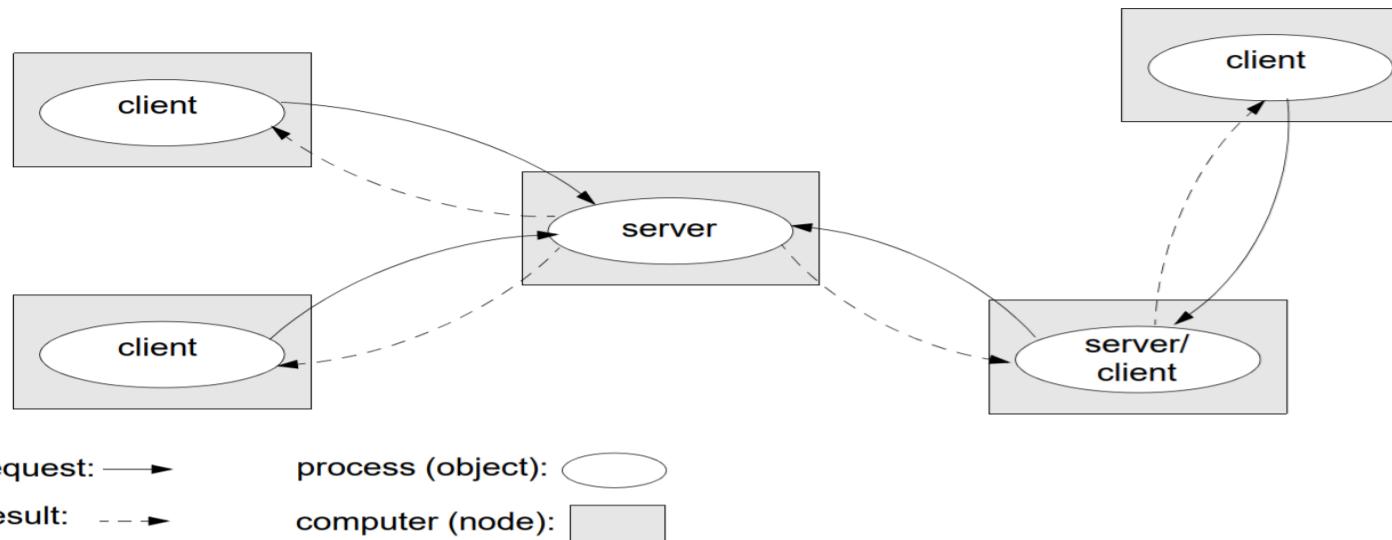
- In a P2P network, every node (peer) acts as both a **client and server**.
- Peers act autonomously to join or leave the network.
- No central coordination or central database is needed.
- This implies that no master-slave relationship exists among the peers.
- The system is self-organizing nodes (recognize and create relatively stable connections to other nodes with similar interests/requirements/capabilities) with distributed control. In other words, no peer machine has a global view of the entire P2P system
- Processing and communication loads for access to objects are distributed across many computers and access links.
- This is the most general and flexible model but securing the overall system would be more challenging as each of the peer would have their own data



What is the Client-Server model?

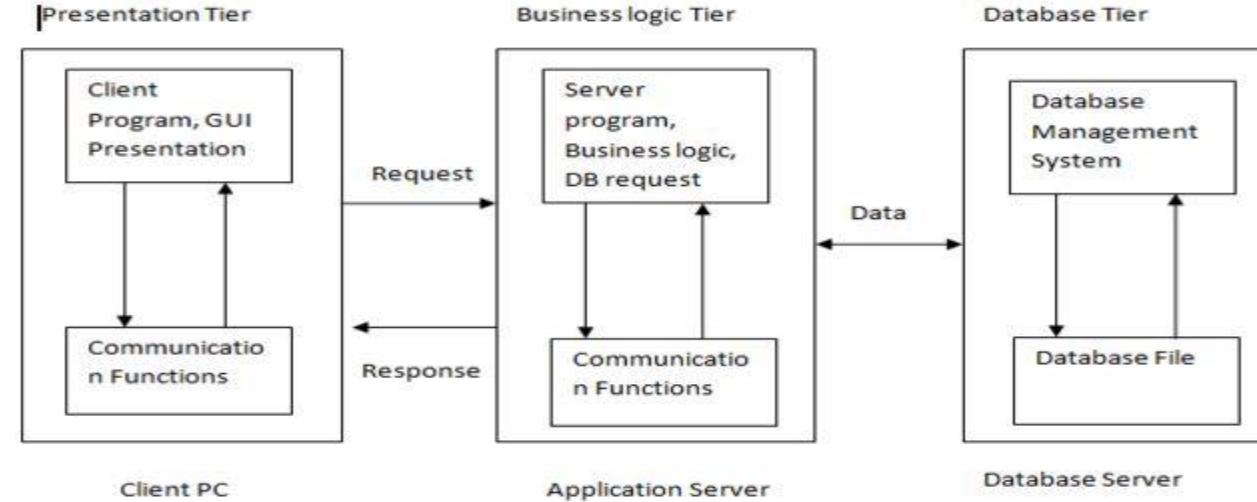
The system is structured where a set of machines called **servers** (which are performing some process), offer services to another set of machines called **clients** for their needs.

- The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls (RPC).
- The client asks the server for a service, the server does the work and returns a result or an error code if the required work could not be done
- This organization by its structure distributes the functionality across different machines



Here are some other architecture models that you should know about:

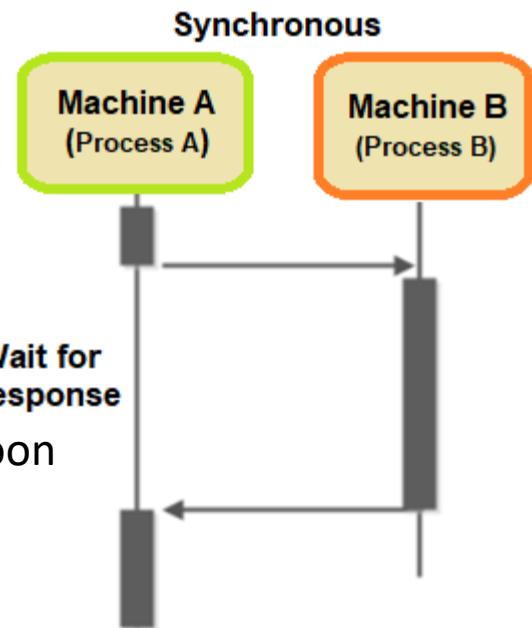
1. **Three-tier** - Architectures that move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are three-tier.



2. **n-tier** - Architectures that refer typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

What are some features of a Synchronous Distributed System?

1. Systems within the Synchronous distributed system have a shared clock (same clock or different synchronized clocks, clocks with known offsets/bounds etc.)
2. Lower and upper bounds on execution time of systems/processes can be set
3. Transmitted messages to be received within a known bounded time.
4. Ordered message delivery or the network will deliver messages in the sent order
5. Lock step execution – all the nodes which are processing identical message would do so as soon as received and generate the output at the same time.
6. Allows us to make assumptions about time and order of events in a distributed system.
7. Not very practical



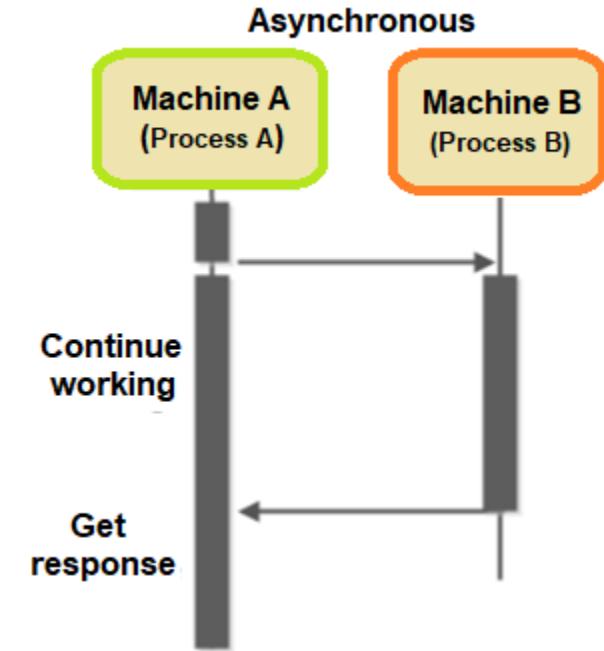
What are the consequences of having a Synchronous Distributed System?

1. Needs a global physical time
2. Needs predictability in terms of timing, as only such systems can be used for hard real-time applications.
3. It is possible and safe to use timeouts in order to detect failures of a process or communication link. (??)

NOTE: Clock drift refers to several related phenomena where a clock does not run at exactly the same rate as a reference clock (NTP). That is, after some time the clock "drifts apart" or gradually desynchronizes from the other clock.

What are some features of an Asynchronous Distributed System?

1. Clock may not be accurate, and can be out of sync
2. No bound on machine/process execution time (nothing can be assumed about speed, load, reliability of computers)
3. No bound on message transmission delays and can be delayed for arbitrary times
4. No constraints on time and ordering of events.
5. Each computer processes independently of others
6. Most suitable for real world scenarios



What are the consequences of having an Asynchronous Distributed System?

1. There is no global physical time, reasoning can be only in terms of logical time.
2. Unpredictable in terms of timing.
3. Cannot use timeouts to diagnose issues
4. They may use mechanisms like queue for asynchronous communication
5. Systems which are using Asynchronous distributed systems have to build algorithms which tolerate different kinds of failures

A system is said to “fail” when it *cannot meet* its promises. A failure is brought about by the *existence* of “errors” in the system. The *cause* of an error is called a “fault”.

A **failure** of a system is brought about due to an **error** in the system caused by a **fault**.

There can be different kinds of faults

- **Transient Faults** : Appears once, then disappears
- **Intermittent Faults** : Occurs, Vanishes, reappears, but no real pattern (worst form of faults)
- **Permanent Faults** : Once it occurs, only the replacement/repair of the faulty component will allow the Distributed System to function normally

What is the use of a Fault Model?

1. Faults can occur both in processes and communication channels. The reason can be both software and hardware.
2. **Fault models** are needed in order to build systems with predictable behavior in case of faults (systems which are fault tolerant).
3. A fault tolerant system will function according to the predictions, only as long as the real faults behave as defined by the “fault model”.

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts.
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests. - A server fails to receive incoming messages. - A server fails to send outgoing messages.
Timing failure	A server's response lies outside the specified time interval.
Response failure <i>Value failure</i> <i>State transition failure</i>	The server's response is incorrect. - The value of the response is wrong. - The server deviates from the correct flow of control.
Arbitrary failure	A server may produce arbitrary responses at arbitrary times.

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a <i>send</i> , but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Process aka Machine

Channel aka Communication Path aka Network Path

<i>Class of Failure</i>	<i>Affects</i>	<i>Description</i>
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

- Applicable in synchronous systems with set time limits
- Not applicable in asynchronous systems since no time limits can be guaranteed.

Process aka Machine

Channel aka Communication Path aka Network Path

References for Distributed System Models

- [Distributed Architecture](#)
- [Distributed Computing Architectures - Wikipedia](#)
- ["Explain Distributed system models with diagram"](#)
- [Make your existing solution tastier with serverless salt: distributed system](#)
- [System Models for Distributed and Cloud Computing - UNF](#)
- [Fundamental Distributed System Models - RIT](#)
- [MODELS OF DISTRIBUTED SYSTEMS - Linköping University](#)
- [Distributed System Models](#)
- [A Brief Summary of Apache Hadoop: A Solution of Big Data Problem and Hint comes from Google](#)



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Cloud Architecture

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

CLOUD COMPUTING

Cloud Architecture

What is cloud architecture?

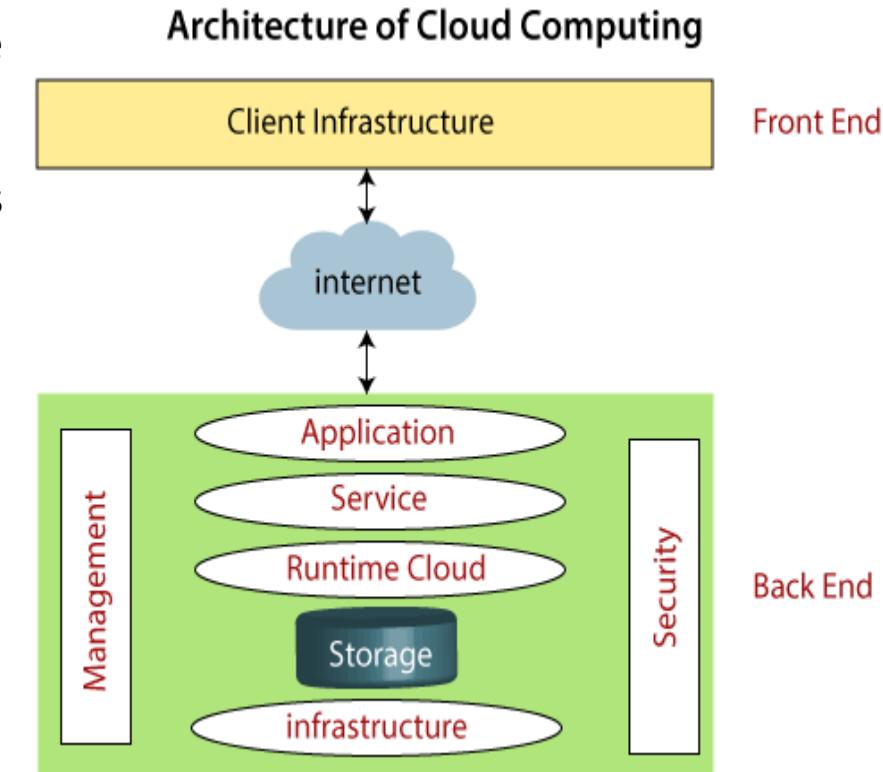
Cloud architecture refers to the way in which technology components which makeup the cloud environment are engineered or combined to leverage the power of cloud resources to solve business problems.

Cloud architecture defines the components as well as the relationships between them.

The components of a cloud architecture include:

- A. A **Front-end** : Client-side interfaces and applications that are required to access the cloud computing platforms. (Web clients, thin or fat clients, tablets, mobile devices)
- B. A **back-end** platform (servers and storage)
- C. A **network** (Internet, Intranet, Intercloud).

Together, these components create a cloud computing architecture on which applications can run, providing end-users with the ability to leverage the power of cloud resources.



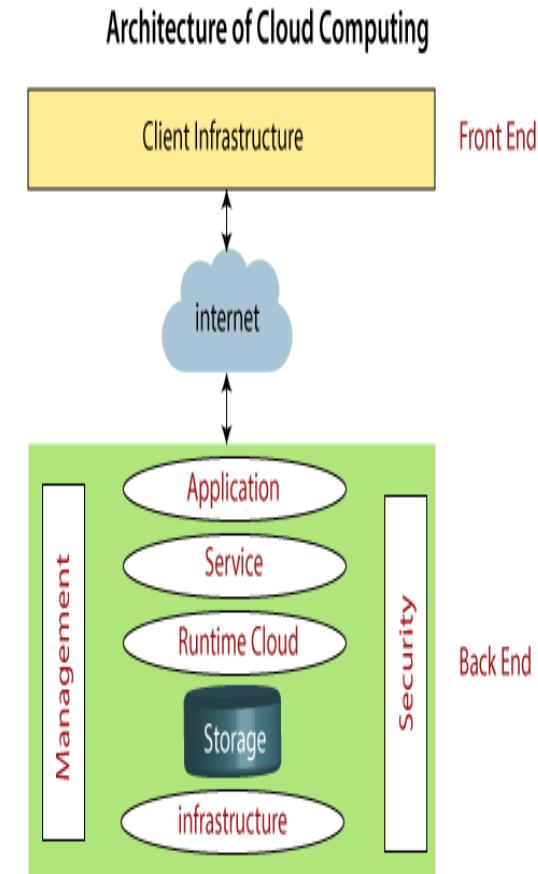
Cloud Architecture – In a little more detail

A. What is a front-end?

- This is the front end part of the cloud Architecture or the **Client Infrastructure** to interact with the cloud.
- This could be client-side interfaces and applications that are required to access the cloud computing platforms.
E.g. Web clients (including Chrome, Firefox, internet explorer etc.), thin & fat clients, tablets, and mobile devices.

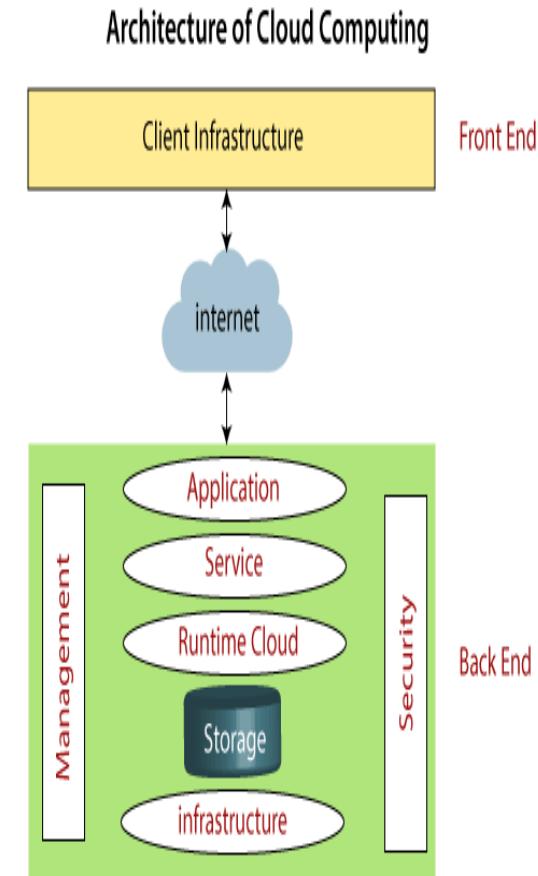
B. What is a back-end?

- The back end part of the cloud architecture is used by the **Service provider**.
- This manages all the resources that are required to provide cloud computing services.
- The detailed set of components involved in the back end are
 1. **Application** - It's the part offered for the Client application which will use the Cloud.
This can either be a software or a platform
 2. **Service** – This is a piece of software, which manages or based on the application needs, will determine and enable the appropriate service to be accessed. This could be an Infrastructure (IaaS) or platform (PaaS) or a software (SaaS) service.



Cloud Architecture – In a little more detail (Cont.)

3. **Runtime Cloud** – Provides the execution and runtime environment to the Virtual Machine dependent on the service model
 4. **Storage** - Storage is one of the most important components of cloud computing. It provides a huge amount of storage capacity in the cloud to store and manage data.
 5. **Infrastructure** - Cloud infrastructure includes hardware and software components such as servers, storage, network devices, virtualization software, and other storage resources that are needed to support the cloud computing model.
 6. **Management** - Components like application, service, etc in the backend need to be managed and coordination between them needs to be established.
 7. **Security** - It implements security mechanisms, for secure cloud resources, systems, files, and infrastructure to end-users.
- C. **Internet** - Internet connection acts as the medium or a bridge between frontend and backend and establishes the interaction and communication between frontend and backend.



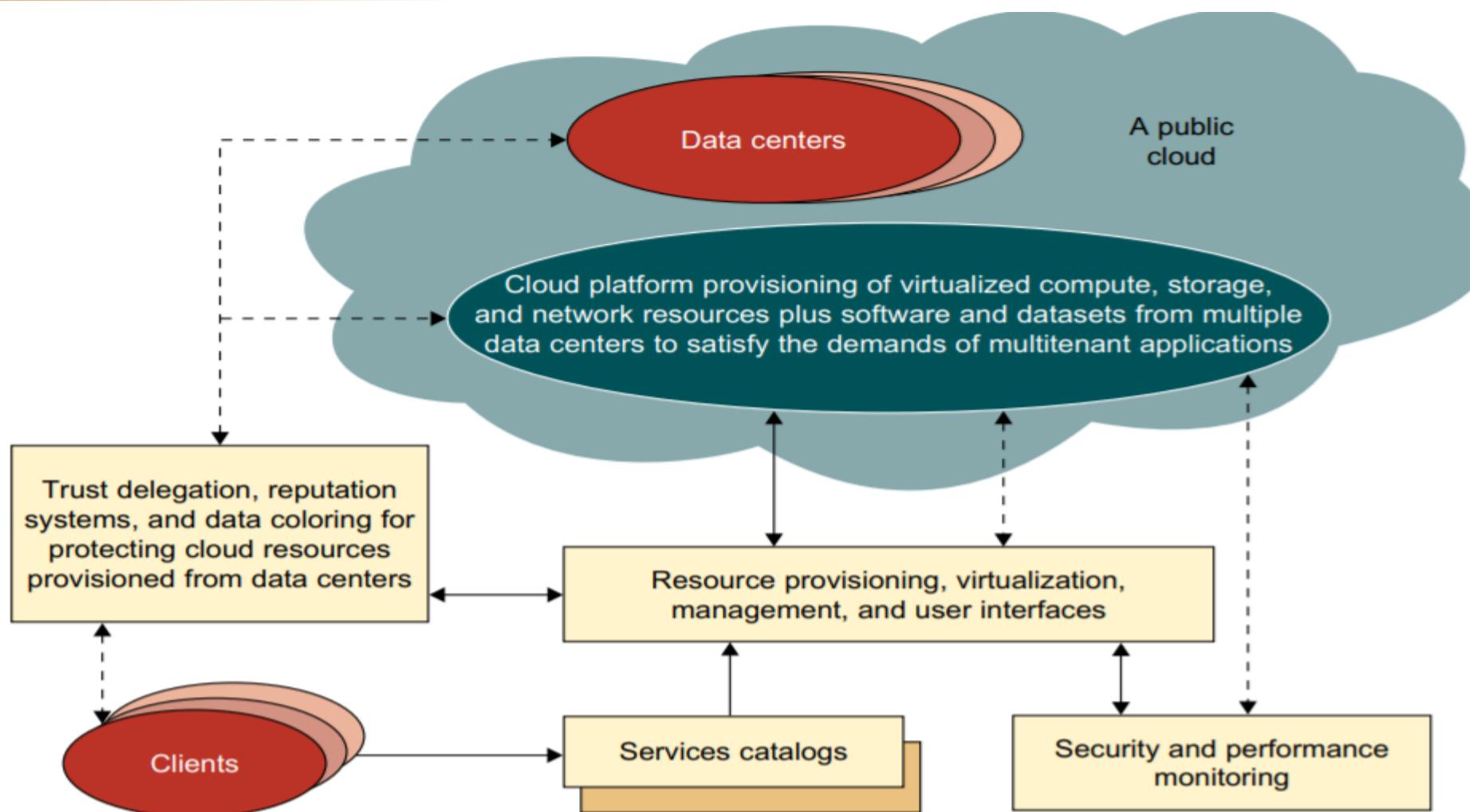
Cloud Platform Design Goals

We have been discussing number of value propositions of using cloud, and these would be the goals targeted for while designing and architecting a Cloud platform. A few of these are

1. Scalability - Up- Out etc
2. Efficiency - Do this quickly with minimal utilization of resources
 - Quickly could also mean bringing up the service which could be across the stack
3. Reliability and Availability
4. Simplifying the User experience

These are typically enabled by architectures like Clustering, Virtualization - Hypervisor, Replication, Elasticity, Simple deployment and scheduling approaches.

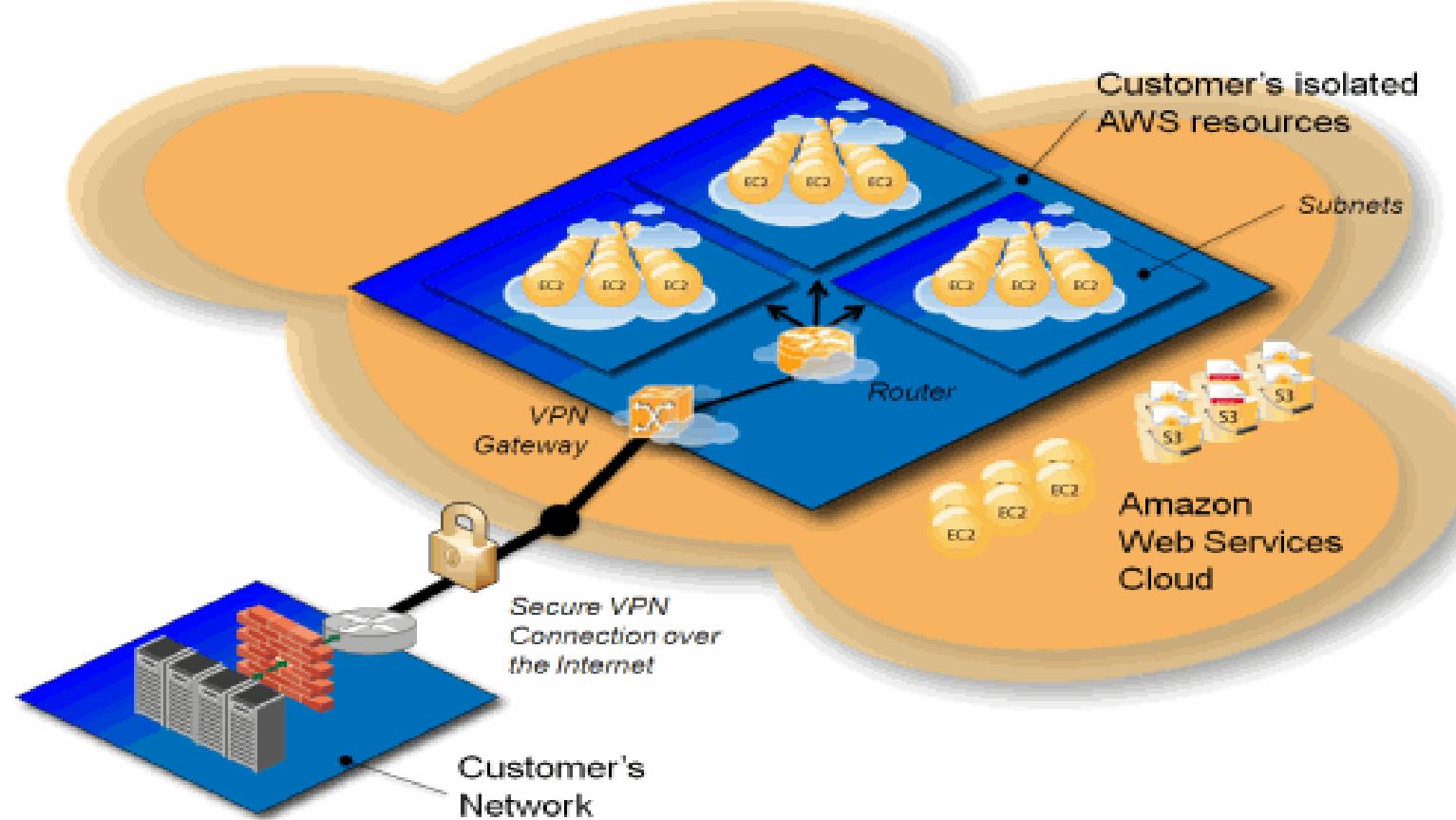
Generic Cloud Architecture Example



Generic cloud platform built with a virtual cluster of VMs, storage, and networking resources over the data-center servers operated by providers which also considers security.

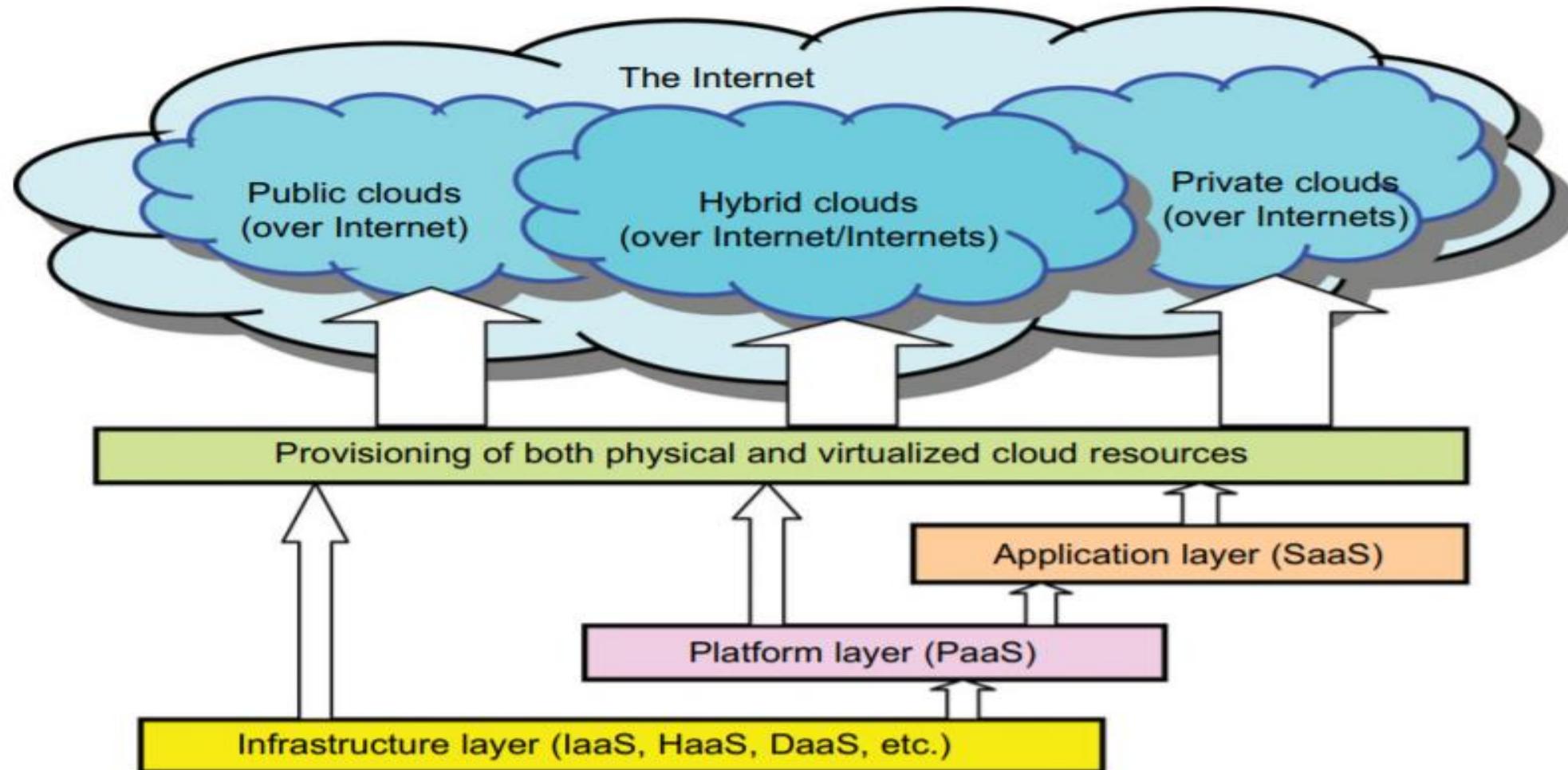
CLOUD COMPUTING

Cloud Architecture : Amazon Example



Other views : Layered Cloud Architecture

The architecture of a cloud can also be visualized into three layers: infrastructure, platform, and application.



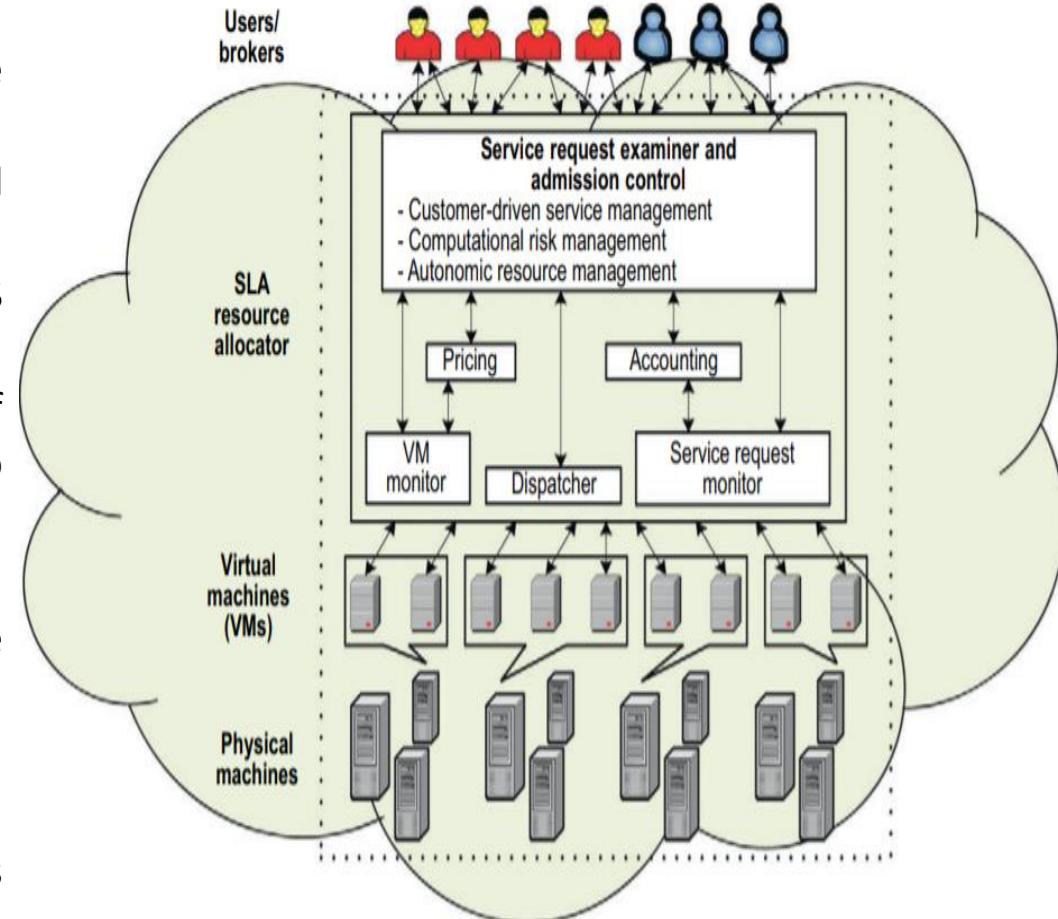
Market-Oriented Cloud Architecture

As consumers rely on cloud providers to meet more of their computing needs, they will require a specific level of QoS to be maintained by their providers, in order to meet their objectives and sustain their operations, which will need to be provisioned as part of deployment.

A market-oriented resource management system regulates the supply and demand of cloud resources to achieve an equilibrium between supply and demand. So the resource allocation mechanism will provision based on the QoS and economic incentives are provided proportional to the QoS needed/provided.

The request examiner ensures that there is no overloading or over-provisioning of resources whereby many service requests can be fulfilled successfully due to constraints in the limit of resources.

- The Pricing mechanism decides how service requests are charged. The VM Monitor mechanism keeps track of the availability of VMs and their resource entitlements.
- The Dispatcher mechanism starts the execution of accepted service requests on allocated VMs.
- The Service Request Monitor mechanism keeps track of the execution progress of service requests.
- Multiple VMs can be started and stopped on demand on a single physical machine to meet accepted service requests, hence providing maximum flexibility.





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Infrastructure as a Service (IaaS)

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

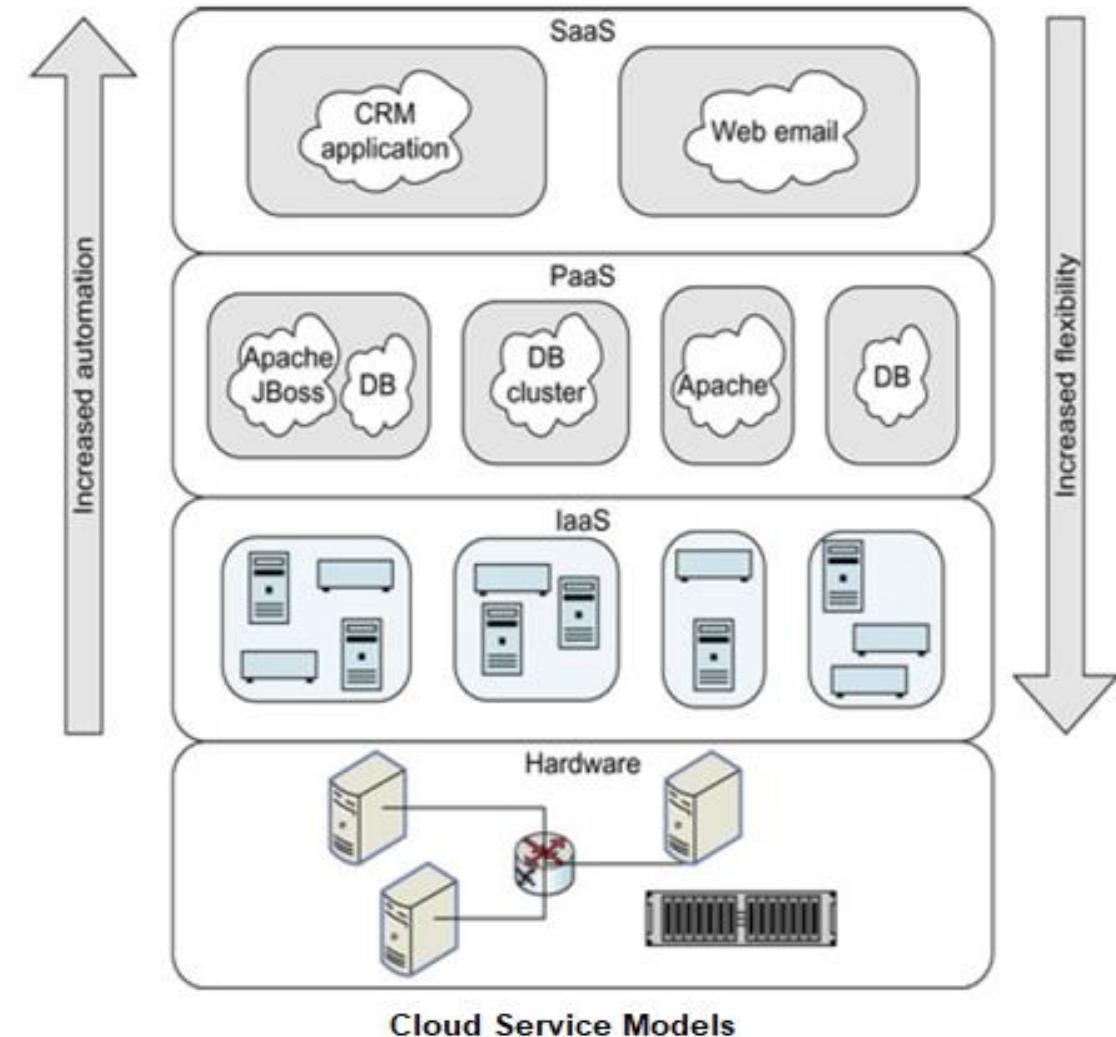
Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Cloud Service Models (Recap)

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers.

The services provided over the cloud have been generally categorized into three different service models:

- Infrastructure as a service
- Platform as a service
- Software as a service



A programming model is an execution model linked into an API or a particular pattern of code. In this, there are actually two execution models in play:

- the **execution model of the base programming language** and
- the **execution model of the programming model.**

A execution model, say of a language like C on a system would be execution of the program one instruction after the other in sequence. The C program statements which are expanded, compiled or converted to assembly, subsequently converted to object code, linked with different object codes to an executable set of instructions, are loaded code into memory and then executed instruction after instruction.

So in the case of a simple program say written in C running on a single system, the programming model would be to execute the code instruction by instruction and use the disk, memory and CPUs locally available on the system.

In cloud environments like IaaS or PaaS or SaaS, the language execution model does not change, but there is an independent execution model of the programming model.

The eco-system for a program execution in terms of the CPU, memory, the persistent memory for storing the programs and data, network ports and IP address which were available locally on your system is now not guaranteed and would be something which needs to be factored in.

Depending on the kind of model, the platforms or frameworks available cannot be assumed to a program as when working on a standalone local node like a physical server or a laptop.

The programming model will need to factor in these and ensure that the requisite environment is setup for each of the different service models of the cloud.

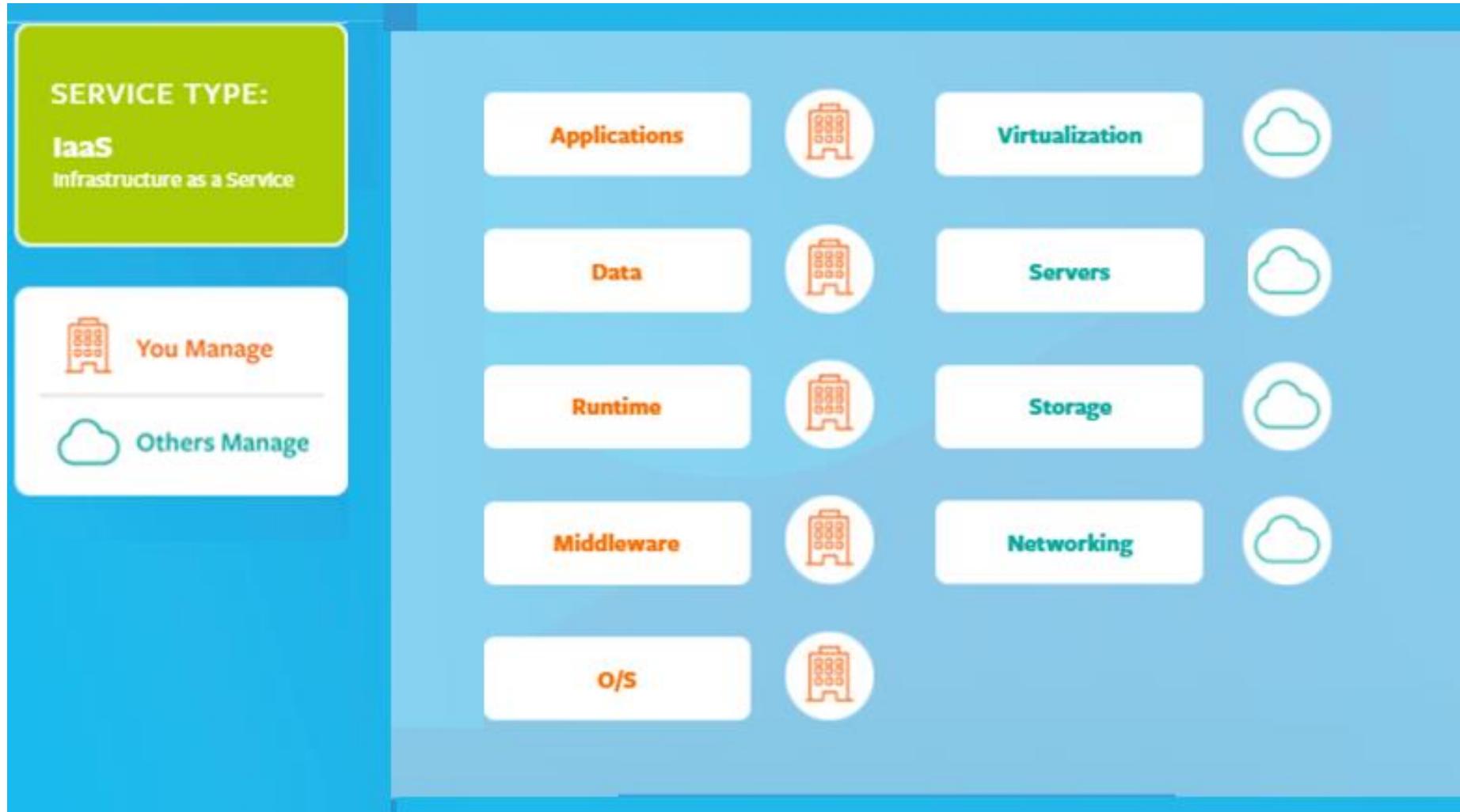
The IaaS model is about providing compute and storage resources as a service.

The capability provided to the consumer is to provision .. processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications.

The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

The user of IaaS has single ownership of the hardware infrastructure allotted to him. The IaaS provider has control over the actual hardware and the cloud user can request allocation of virtual resources.

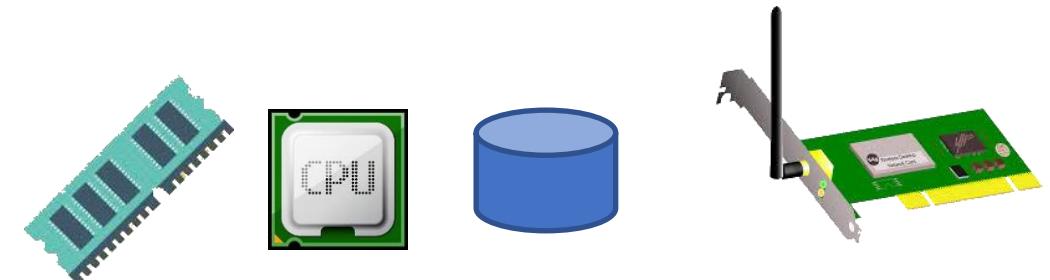
IaaS enables end users to scale and shrink resources on an as-needed basis, reducing the need for high, up-front capital expenditures or unnecessary “owned” infrastructure, especially in the case of “spiky” workloads.



- If you are writing a complex program
 - HelloWorld.exe
- Program needs resources to execute
 - CPU
 - Memory
 - Storage (where the program resides)
 - Possibly network
- Access to resources made available by the OS

HelloWorld.exe

Operating System



Programming model if this needs to run on a Virtual System

- If you are writing a complex program

- Helloworld.exe

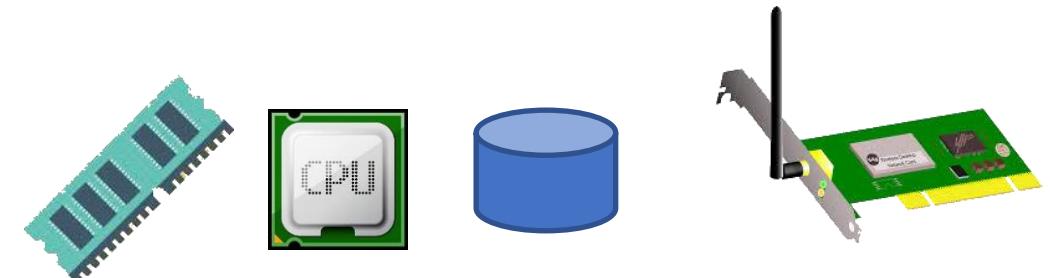
HelloWorld.exe

- Let's assume that this program needs to run on a virtual machine

- How does it get access to

- CPU
 - Memory
 - Storage (where the program resides)
 - Possibly network

???



- Given that your application is running now on a remote server, there will need to be something like a WebServer to accept the application requests

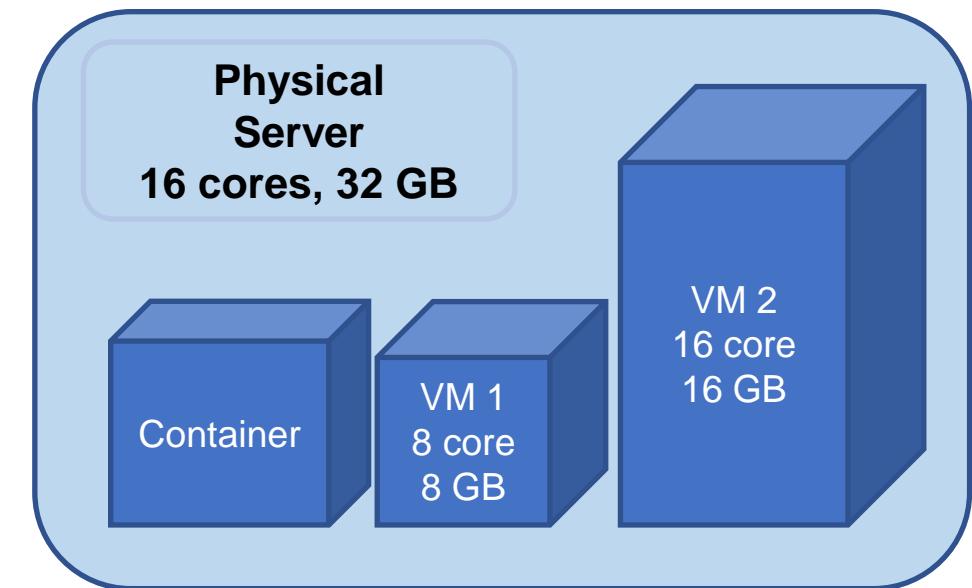
IaaS platform infrastructure - Processing

In IaaS, Infrastructure needs to be provisioned for the applications which will need to be run. IaaS is made up of a collection of physical and virtualized resources that provide consumers with the basic building blocks needed to run applications and workloads in the cloud.

Physical data centers: IaaS providers will manage large data centers, typically around the world, that contain the physical machines with various layers of abstraction on top of them and that are made available to end users over the web.

Compute: Providers provision your compute in the granularity of VMs/Containers on top of the physical Servers using Virtualization techniques which will study in the next unit.

- VMs
 - Each VM "looks like" an actual physical machine to the software
 - Can have its own OS and software
- Container
 - Uses physical server's OS
 - Actually a sandboxed process
 - More lightweight than a VM
- VMs or containers may belong to different users
- Each VM or container is isolated from other VMs
- For now, don't worry about how virtualisation works
 - We will study that in later classes
 - Assume it happens by magic
 - Achieved via the hypervisor



Storage: The three primary types of cloud storage are block storage, file storage, and object storage. An example of how Block storage is shared could be as below.

A way of sharing and pooling physical disks

Sharing:

Pooling: treat disk pool as single large storage say of disks of different capability

E.g., 256 GB virtual disk using 2x128 GB from different physical disks

Can virtual disk be bigger than physical disk?

Each virtual disk can be attached to a VM

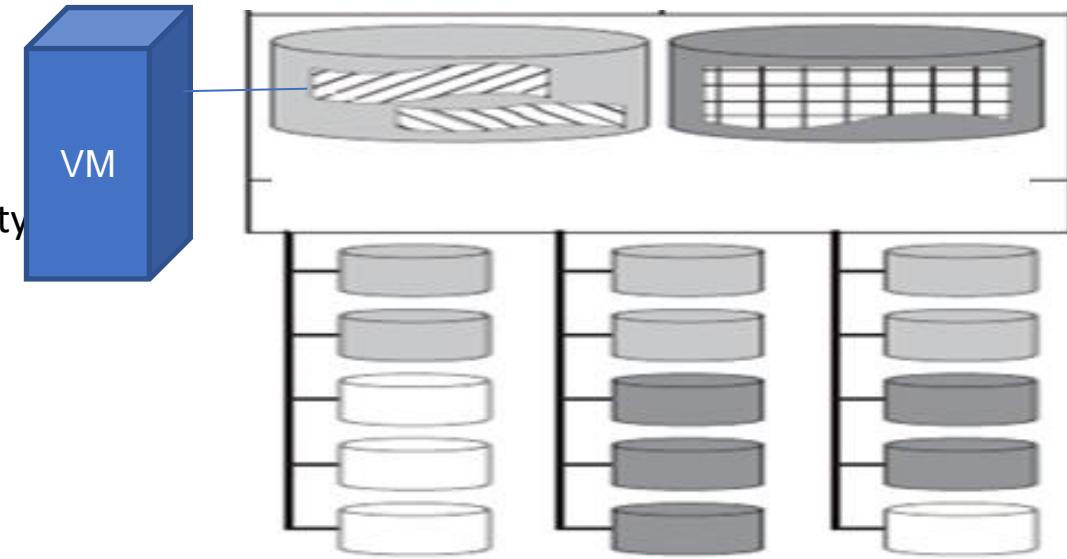
Each virtual disk "looks like" an actual physical disk to the software

Can create files on disk

Can use for paging or swapping

The virtual disks may belong to different users

Each virtual disk is isolated from other virtual disks



The Object storage has thus become the most common mode of storage in the cloud given that it is highly distributed (and thus resilient), it leverages commodity hardware, data can be accessed easily over HTTP, and scale is not only essentially limitless but performance scales linearly as the cluster grows.

We will discuss more on this later.

Network: Networking in the cloud is a form of Software Defined Networking in which traditional networking hardware, such as routers and switches, are made available programmatically, typically through APIs.

- Like physical network, contains virtual adapters and virtual switches
- Each VM has one or more *virtual network adapters* (V)

These have IP addresses, DNS names, just like a physical adapter

This is completely in software

VS is a *virtual network switch*

Completely in software

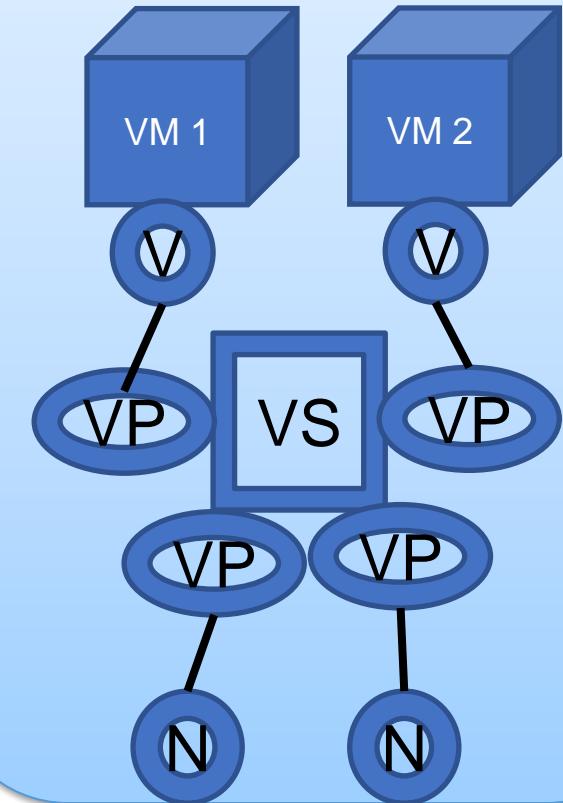
VP is a virtual port

Network adapter, invisible to network

No IP address, only MAC address

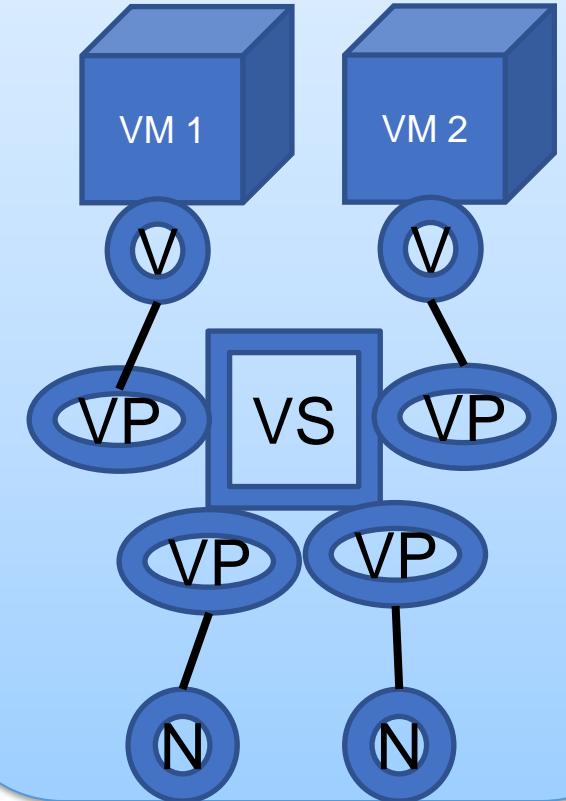
N is the network adapter of the physical server

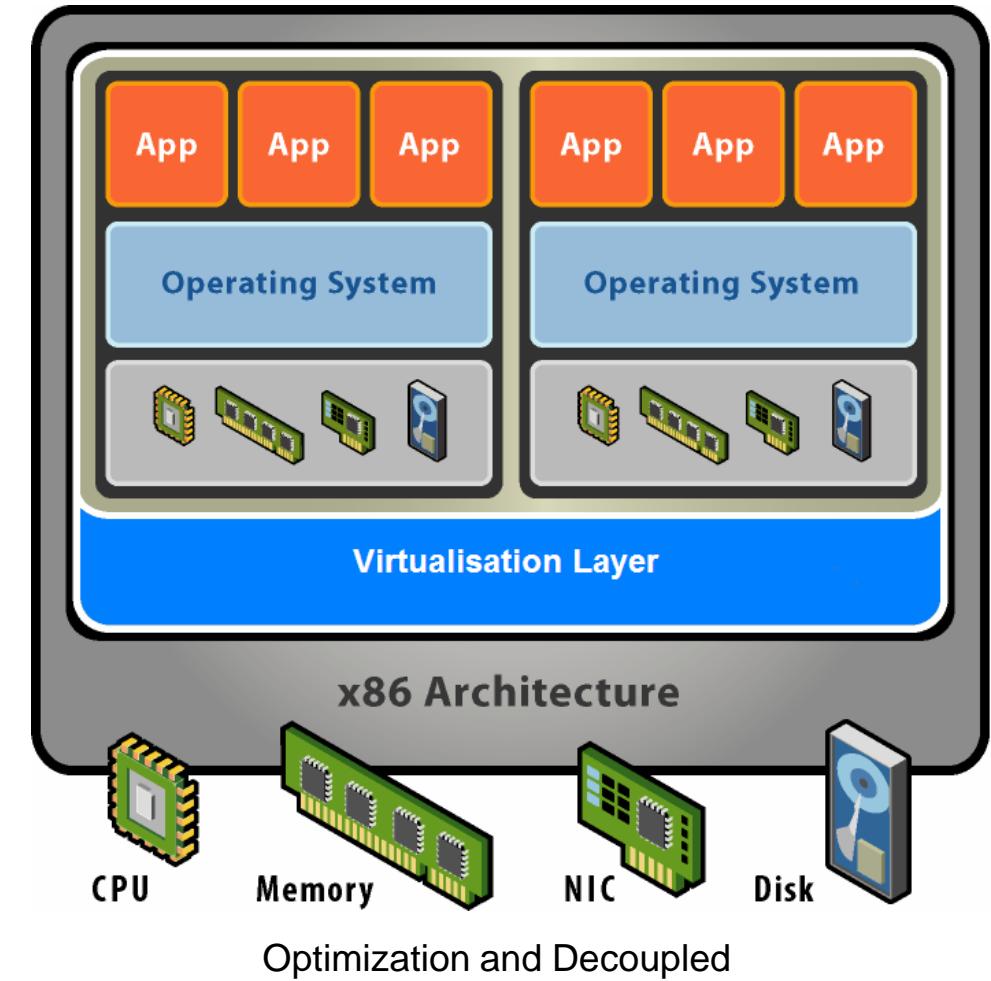
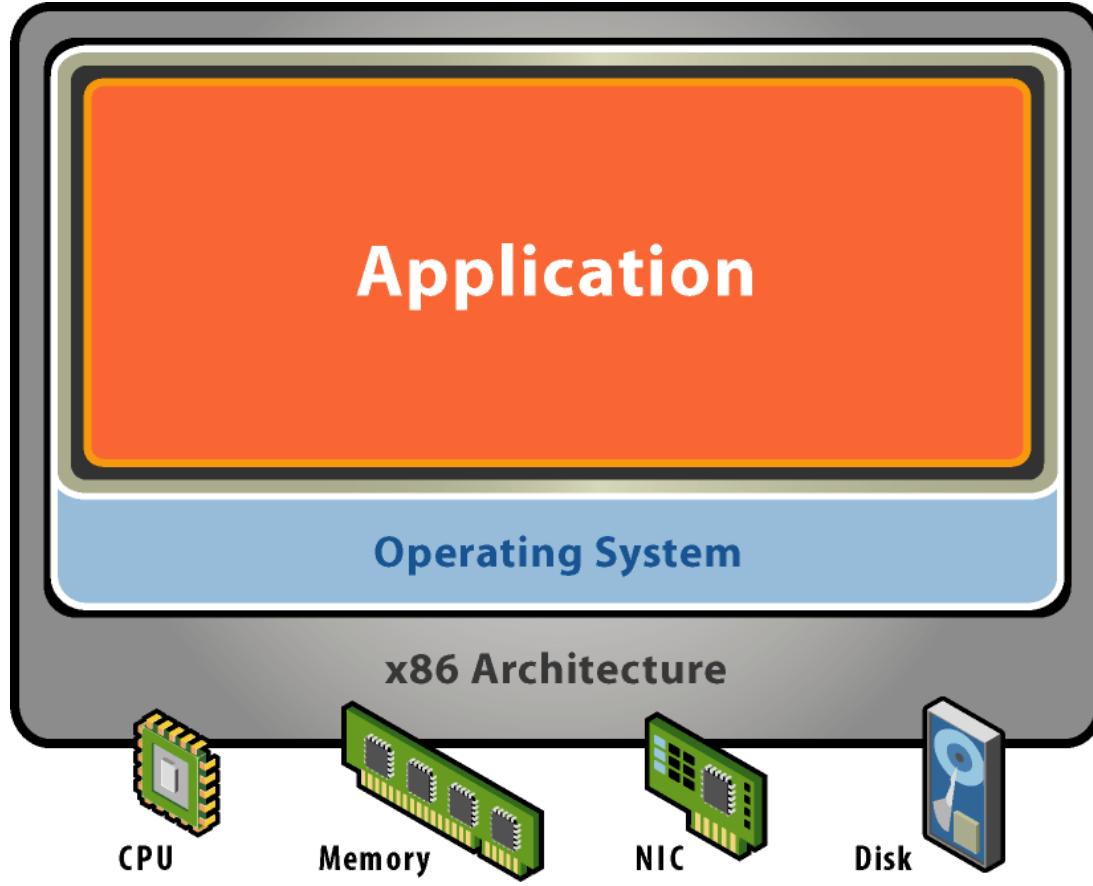
Physical Server



- User specifies
 - VM configuration (e.g., memory)
 - Software
- Cloud software
 - Create actual VM
 - Create virtual disk(s) needed
 - Copy OS and app binaries to boot disk
 - Boot OS
 - Set up virtual network

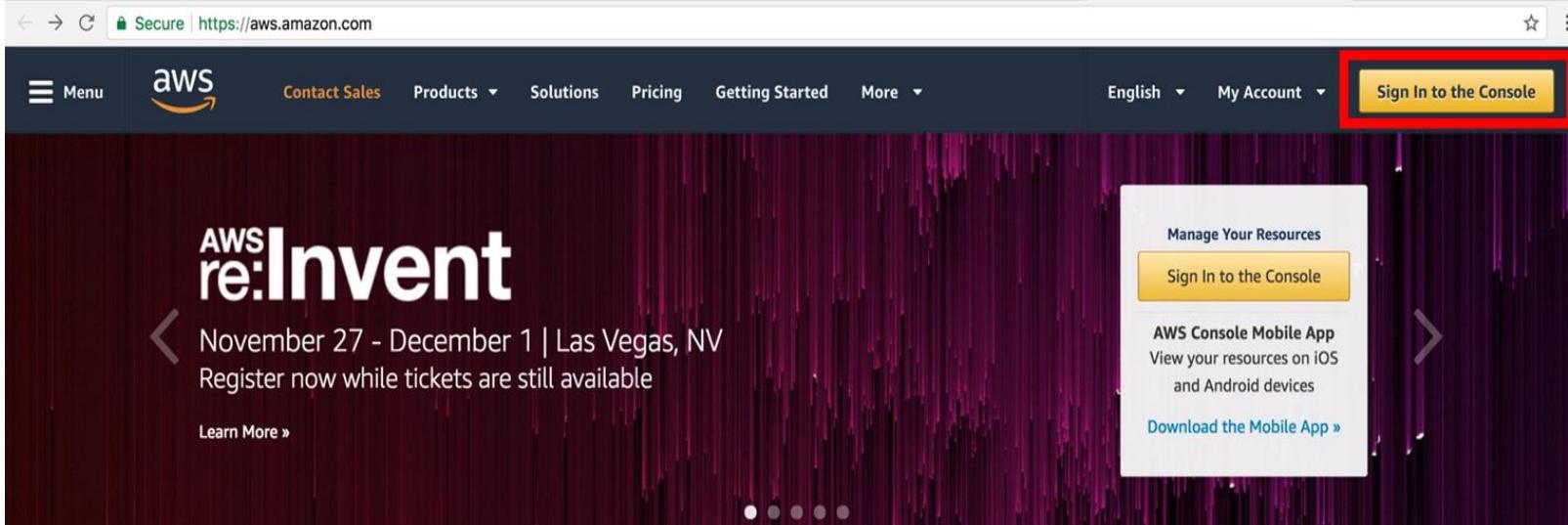
Physical Server





Demo: IaaS on AWS (Compute as a service)

- Step 1: Create an AWS Account and Sign into AWS.

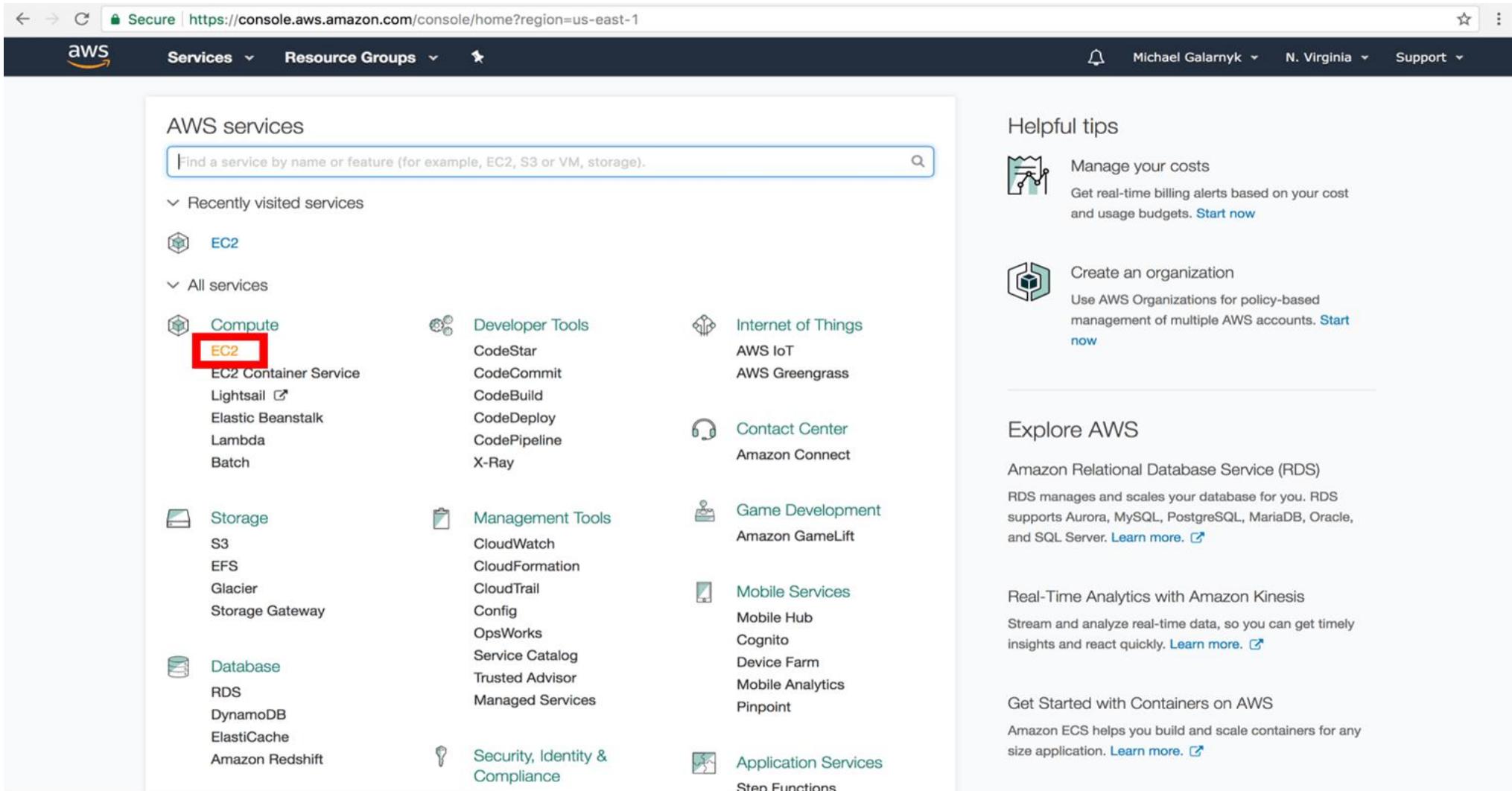


<https://www.datacamp.com/community/tutorials/aws-ec2-beginner-tutorial>

CLOUD COMPUTING

Demo: IaaS on AWS (Compute as a service)

- Step 2: On the EC2 Dashboard, click on EC2..



The screenshot shows the AWS Cloud Services Dashboard. The top navigation bar includes the AWS logo, a 'Services' dropdown, 'Resource Groups' dropdown, user profile 'Michael Galarnyk', location 'N. Virginia', and 'Support'. The main area is titled 'AWS services' with a search bar. Under 'Recently visited services', 'EC2' is listed. Under 'All services', 'Compute' is expanded, showing 'EC2' (which has a red box around it), 'EC2 Container Service', 'Lightsail', 'Elastic Beanstalk', 'Lambda', 'Batch', 'Developer Tools' (listing 'CodeStar', 'CodeCommit', 'CodeBuild', 'CodeDeploy', 'CodePipeline', 'X-Ray'), 'Internet of Things' (listing 'AWS IoT', 'AWS Greengrass'), 'Contact Center' (listing 'Amazon Connect'), 'Management Tools' (listing 'CloudWatch', 'CloudFormation', 'CloudTrail', 'Config', 'OpsWorks', 'Service Catalog', 'Trusted Advisor', 'Managed Services'), 'Game Development' (listing 'Amazon GameLift'), 'Mobile Services' (listing 'Mobile Hub', 'Cognito', 'Device Farm', 'Mobile Analytics', 'Pinpoint'), 'Security, Identity & Compliance', and 'Application Services' (listing 'Step Functions'). To the right, there's a 'Helpful tips' section with 'Manage your costs' (with a graph icon) and 'Create an organization' (with a cube icon). Below that is an 'Explore AWS' section with links to 'Amazon Relational Database Service (RDS)', 'Real-Time Analytics with Amazon Kinesis', and 'Get Started with Containers on AWS'.

AWS services

Find a service by name or feature (for example, EC2, S3 or VM, storage).

Recently visited services

EC2

All services

Compute

EC2

EC2 Container Service

Lightsail

Elastic Beanstalk

Lambda

Batch

Developer Tools

CodeStar

CodeCommit

CodeBuild

CodeDeploy

CodePipeline

X-Ray

Internet of Things

AWS IoT

AWS Greengrass

Contact Center

Amazon Connect

Management Tools

CloudWatch

CloudFormation

CloudTrail

Config

OpsWorks

Service Catalog

Trusted Advisor

Managed Services

Game Development

Amazon GameLift

Mobile Services

Mobile Hub

Cognito

Device Farm

Mobile Analytics

Pinpoint

Security, Identity & Compliance

Application Services

Step Functions

Helpful tips

Manage your costs

Get real-time billing alerts based on your cost and usage budgets. [Start now](#)

Create an organization

Use AWS Organizations for policy-based management of multiple AWS accounts. [Start now](#)

Explore AWS

Amazon Relational Database Service (RDS)

RDS manages and scales your database for you. RDS supports Aurora, MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. [Learn more](#).

Real-Time Analytics with Amazon Kinesis

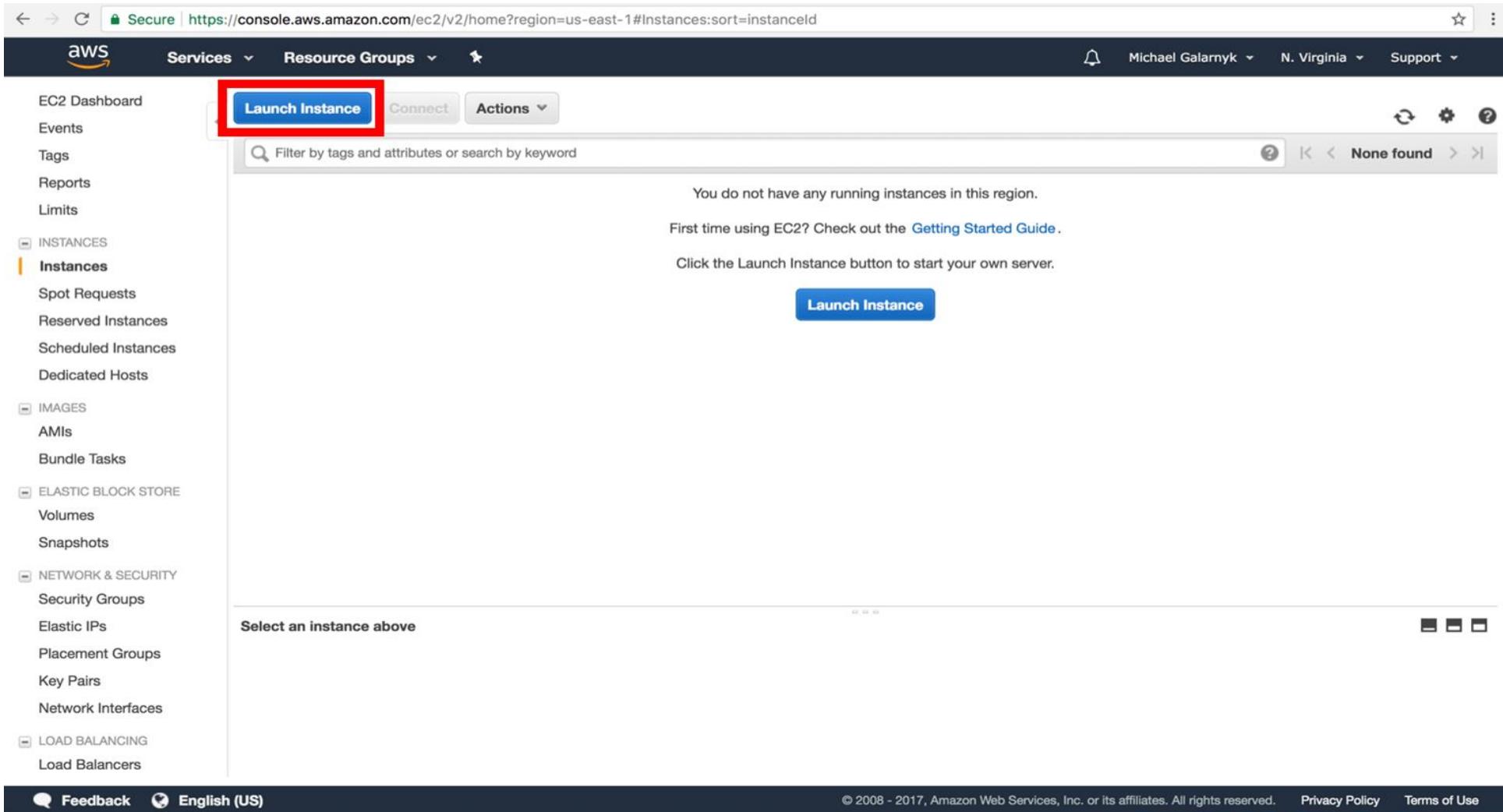
Stream and analyze real-time data, so you can get timely insights and react quickly. [Learn more](#).

Get Started with Containers on AWS

Amazon ECS helps you build and scale containers for any size application. [Learn more](#).

Demo: IaaS on AWS (Compute as a service)

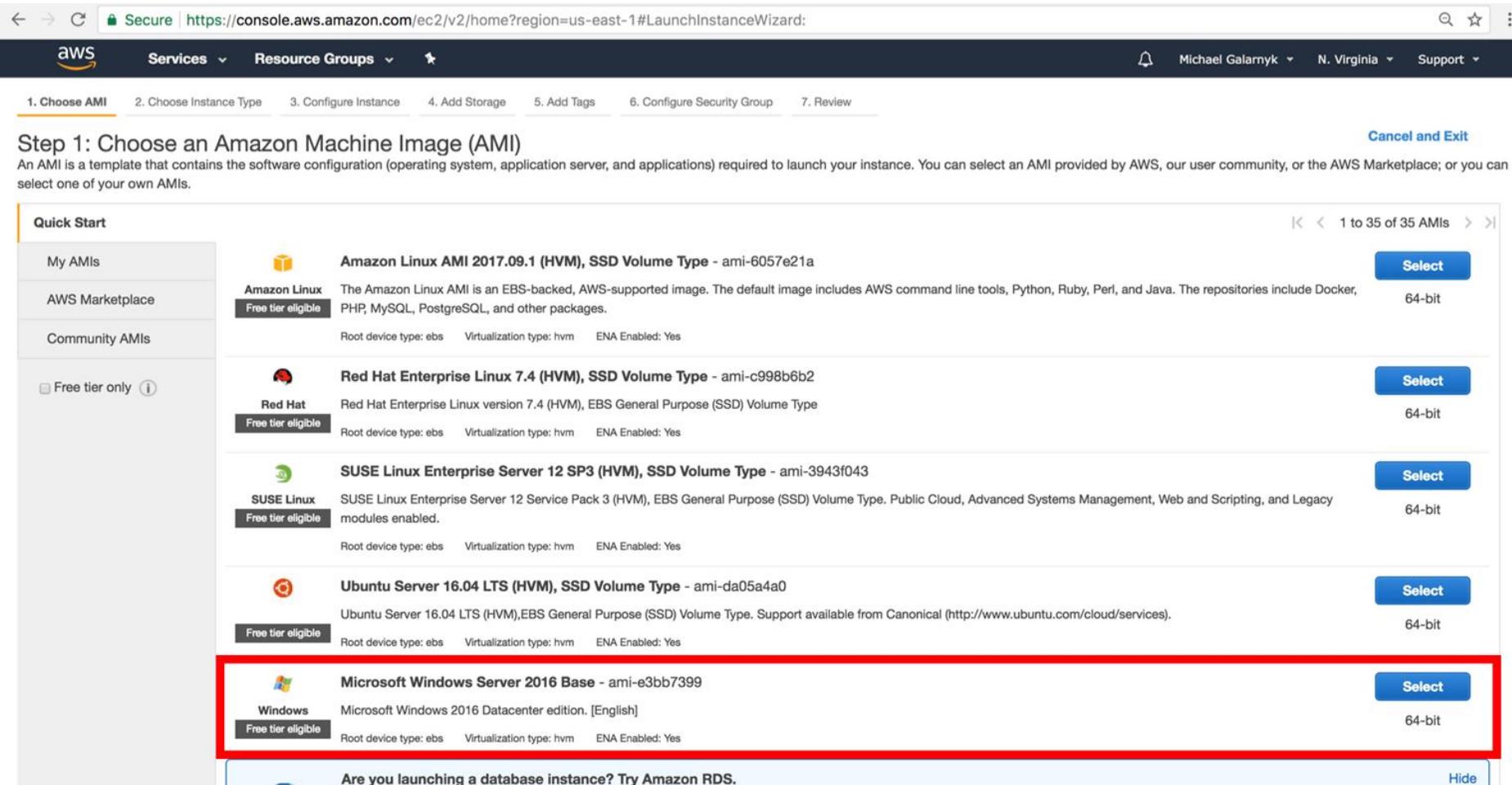
- Step 3: On the Amazon EC2 console, click on Launch Instance.



The screenshot shows the AWS EC2 Dashboard. The left sidebar menu is visible, with 'Instances' selected. The main content area displays a message: 'You do not have any running instances in this region.' It includes links for 'Getting Started Guide' and 'Click the Launch Instance button to start your own server.' A prominent blue 'Launch Instance' button is located in the center. The browser's address bar shows the URL: <https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#instances:sort=instanceId>.

Demo: IaaS on AWS (Compute as a service)

- Step 4: Select the required operating system image.



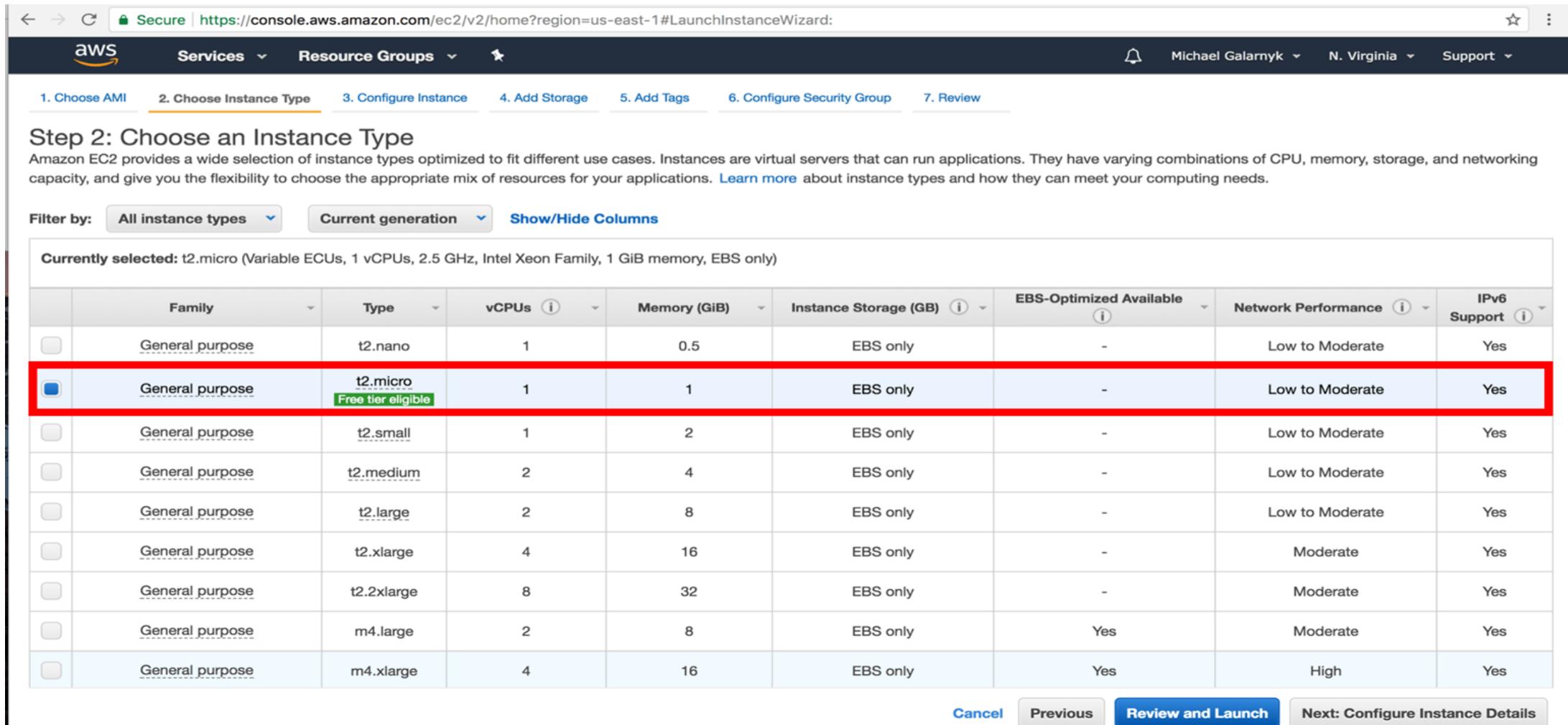
The screenshot shows the AWS Launch Instance Wizard Step 1: Choose AMI. The page title is "Step 1: Choose an Amazon Machine Image (AMI)". It explains that an AMI is a template containing software configuration required to launch an instance. The user can select an AWS-provided AMI or one from the Marketplace or their own AMIs. A sidebar on the left lists "Quick Start" options: My AMIs, AWS Marketplace, Community AMIs, and a "Free tier only" checkbox. The main content area displays a list of 35 AMIs:

Image	Name	Description	Select	Architecture
	Amazon Linux AMI 2017.09.1 (HVM), SSD Volume Type - ami-6057e21a	The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.	<button>Select</button>	64-bit
	Red Hat Enterprise Linux 7.4 (HVM), SSD Volume Type - ami-c998b6b2	Red Hat Enterprise Linux version 7.4 (HVM), EBS General Purpose (SSD) Volume Type	<button>Select</button>	64-bit
	SUSE Linux Enterprise Server 12 SP3 (HVM), SSD Volume Type - ami-3943f043	SUSE Linux Enterprise Server 12 Service Pack 3 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.	<button>Select</button>	64-bit
	Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-da05a4a0	Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).	<button>Select</button>	64-bit
	Microsoft Windows Server 2016 Base - ami-e3bb7399	Microsoft Windows 2016 Datacenter edition. [English]	<button>Select</button>	64-bit

A red box highlights the Microsoft Windows Server 2016 Base AMI. A message at the bottom encourages launching a database instance using Amazon RDS.

Demo: IaaS on AWS (Compute as a service)

- Step 5: Select the required hardware configuration



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Michael Galarnyk N. Virginia Support

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

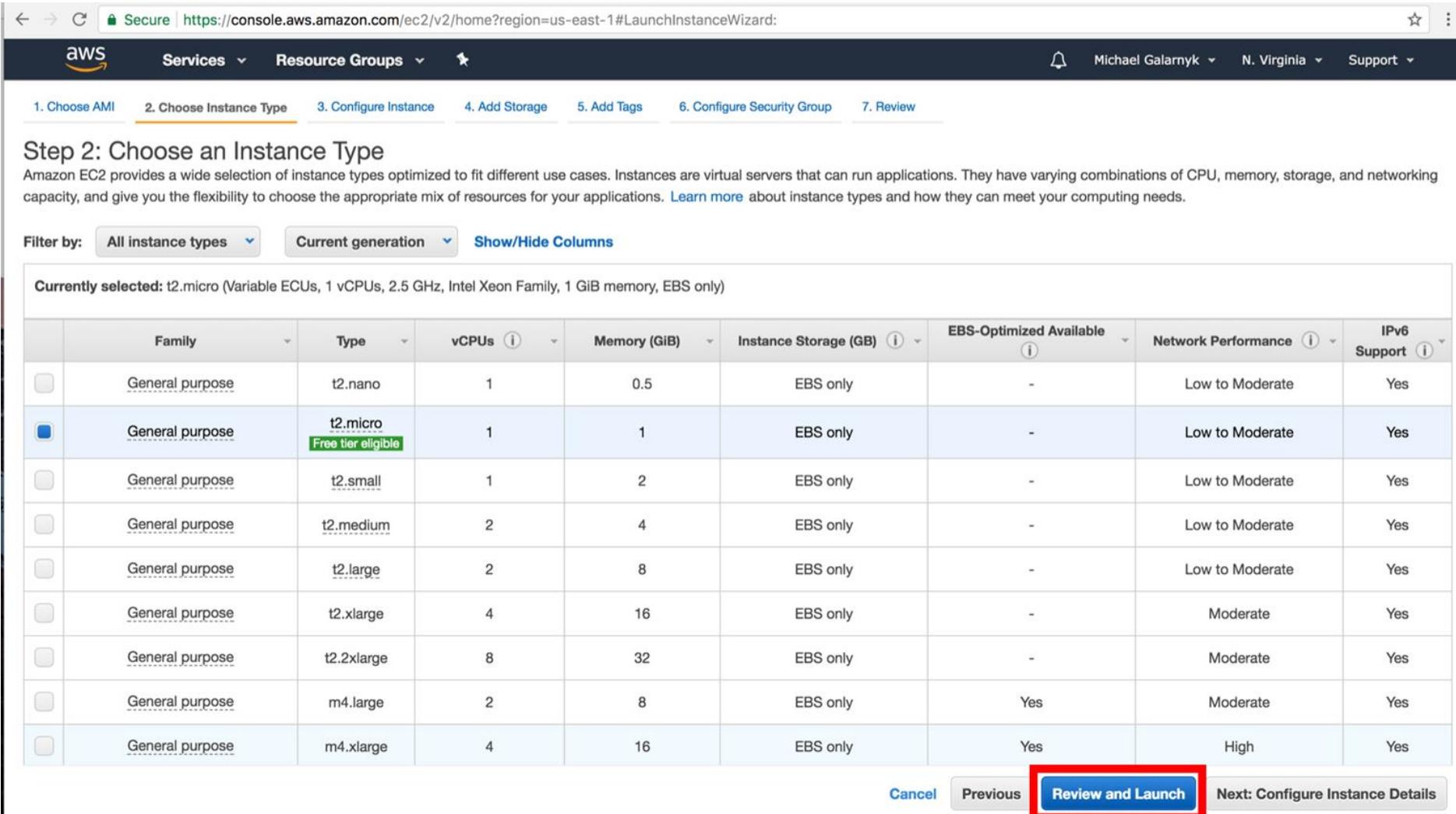
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
General purpose	m4.large	2	8	EBS only	Yes	Moderate	Yes
General purpose	m4.xlarge	4	16	EBS only	Yes	High	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Demo: IaaS on AWS (Compute as a service)

- Step 6: click on "Review and Launch"



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups Michael Galarnyk N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. [Learn more](#) about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

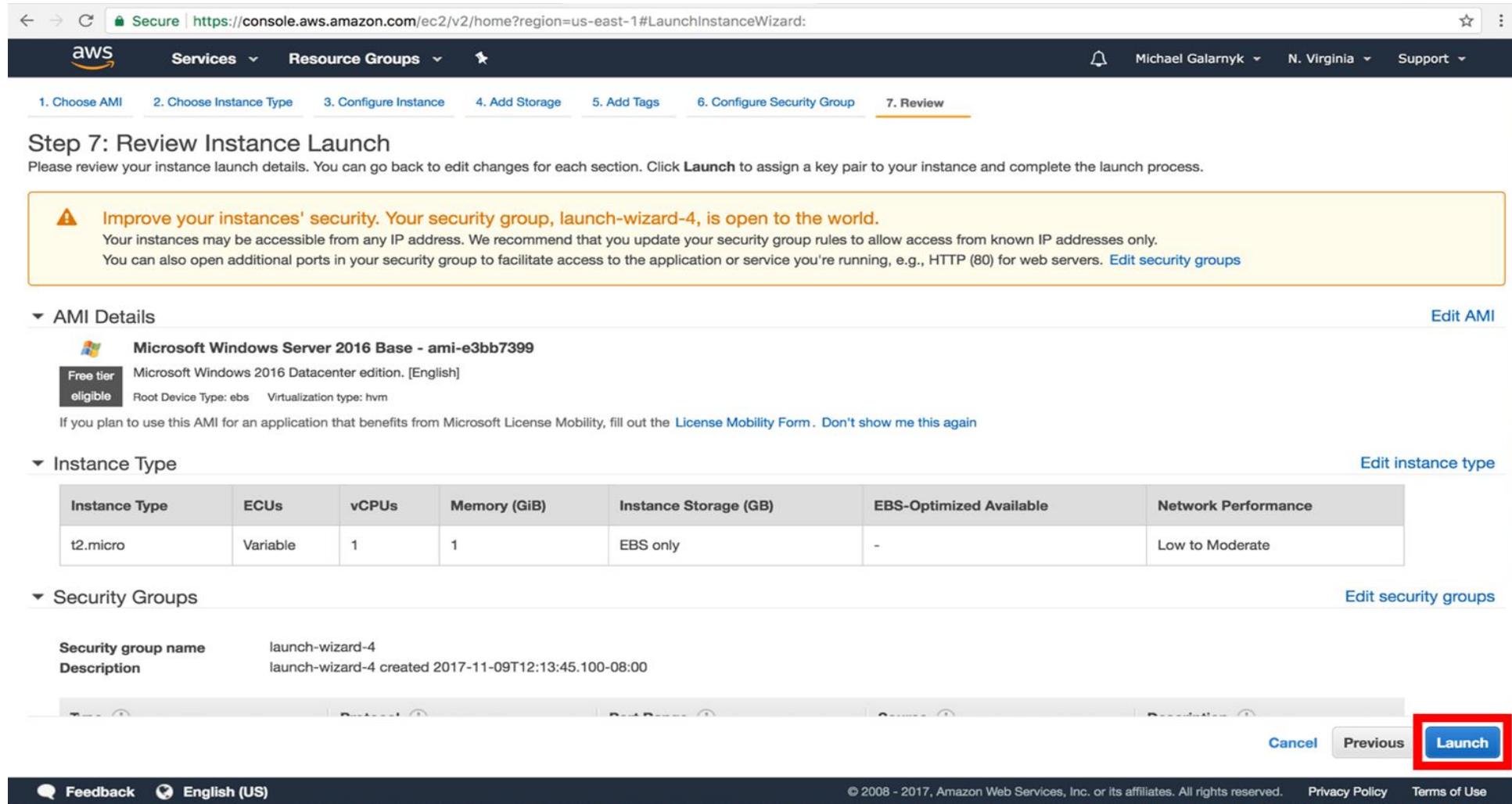
Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input checked="" type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
	General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
	General purpose	m4.large	2	8	EBS only	Yes	Moderate	Yes
	General purpose	m4.xlarge	4	16	EBS only	Yes	High	Yes

Cancel Previous **Review and Launch** Next: Configure Instance Details

Demo: IaaS on AWS (Compute as a service)

- Step 7: Click on Launch.



The screenshot shows the AWS Launch Instance Wizard Step 7: Review Instance Launch page. The URL is <https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard>. The top navigation bar includes the AWS logo, Services, Resource Groups, Michael Galarnyk, N. Virginia, and Support. The breadcrumb navigation shows 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review (the current step).

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

AMI Details

Microsoft Windows Server 2016 Base - ami-e3bb7399
Free tier eligible Microsoft Windows 2016 Datacenter edition. [English]
Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from Microsoft License Mobility, fill out the [License Mobility Form](#). Don't show me this again

Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

Security Groups

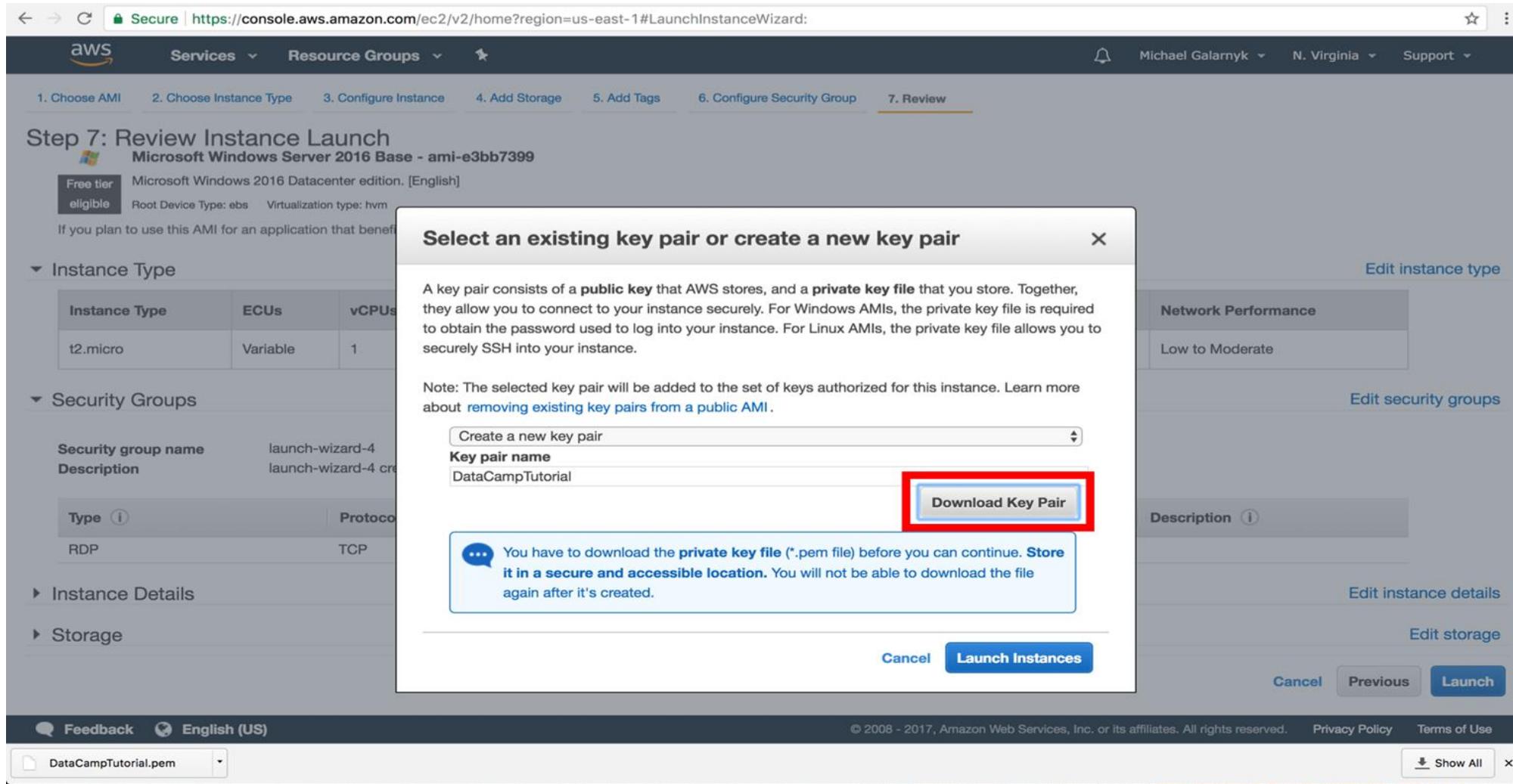
Security group name: launch-wizard-4
Description: launch-wizard-4 created 2017-11-09T12:13:45.100-08:00

At the bottom right, there are three buttons: Cancel, Previous, and Launch. The Launch button is highlighted with a red box.

CLOUD COMPUTING

Demo: IaaS on AWS (Compute as a service)

- Step 8: 7. Select "Create a new key pair". Click on "Download Key Pair". This will download the key. Keep it somewhere safe.



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

Michael Galarmy N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Microsoft Windows Server 2016 Base - ami-e3bb7399

Free tier eligible

Microsoft Windows 2016 Datacenter edition, [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from a free tier, you can use this instance type.

Instance Type

Instance Type	ECUs	vCPUs
t2.micro	Variable	1

Security Groups

Security group name: launch-wizard-4

Description: launch-wizard-4 created

Type: RDP Protocol: TCP

Network Performance: Low to Moderate

Edit security groups

Description

Edit instance details

Edit storage

Cancel Previous Launch

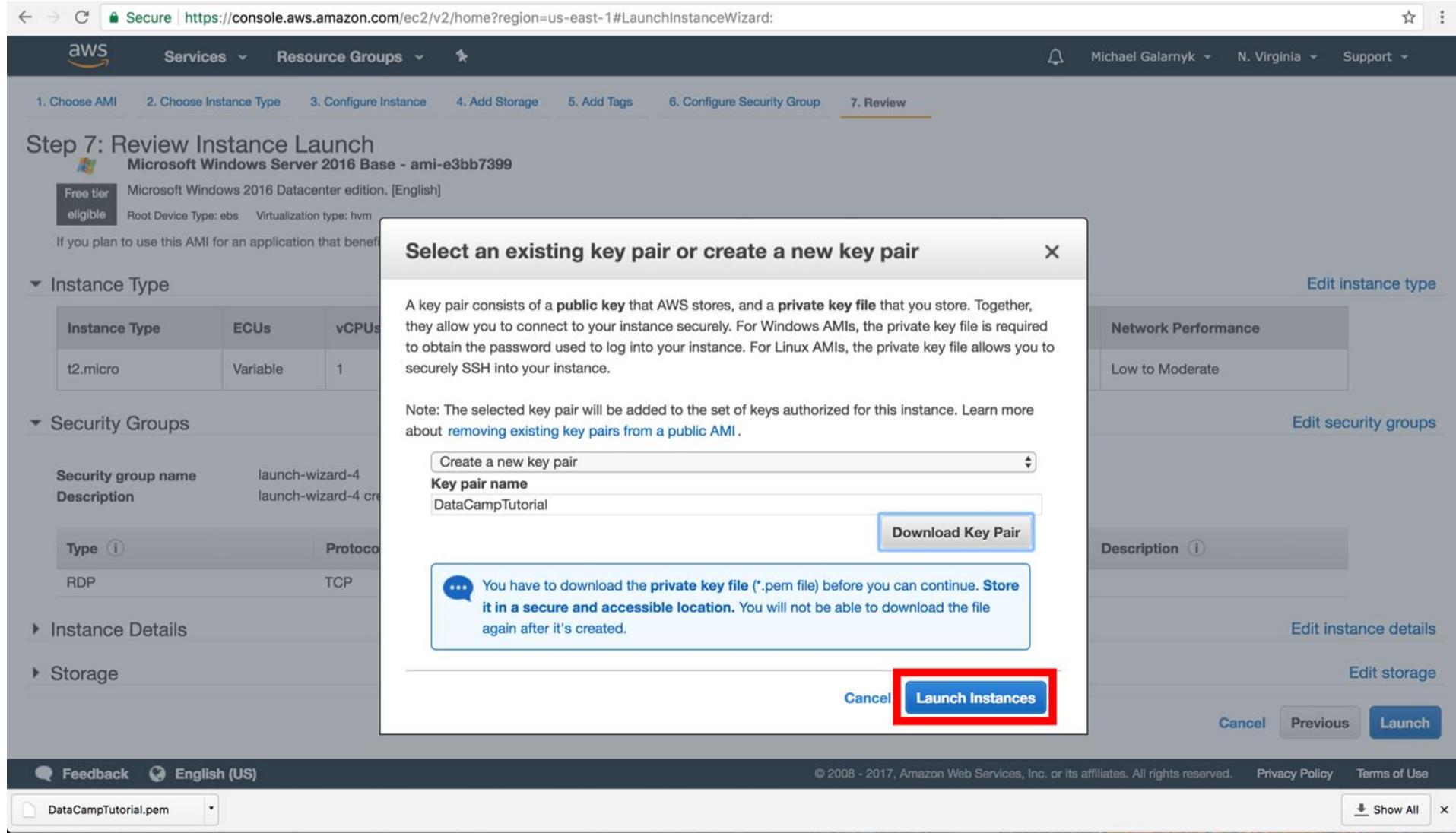
Feedback English (US)

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

DataCampTutorial.pem Show All

Demo: IaaS on AWS (Compute as a service)

- Step 9: Next, click on "Launch Instances".



Secure | https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#LaunchInstanceWizard:

aws Services Resource Groups

Michael Galarnyk N. Virginia Support

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Microsoft Windows Server 2016 Base - ami-e3bb7399

Free tier eligible Microsoft Windows 2016 Datacenter edition, [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from enhanced security, consider using a key pair.

Instance Type

Instance Type	ECUs	vCPUs
t2.micro	Variable	1

Security Groups

Security group name	Description
launch-wizard-4	launch-wizard-4 created

Type RDP Protocol TCP

Instance Details

Storage

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name DataCampTutorial

Download Key Pair

You have to download the **private key file (*.pem file)** before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel Launch Instances

Network Performance Low to Moderate

Edit instance type Edit security groups

Description

Edit instance details Edit storage

Cancel Previous Launch

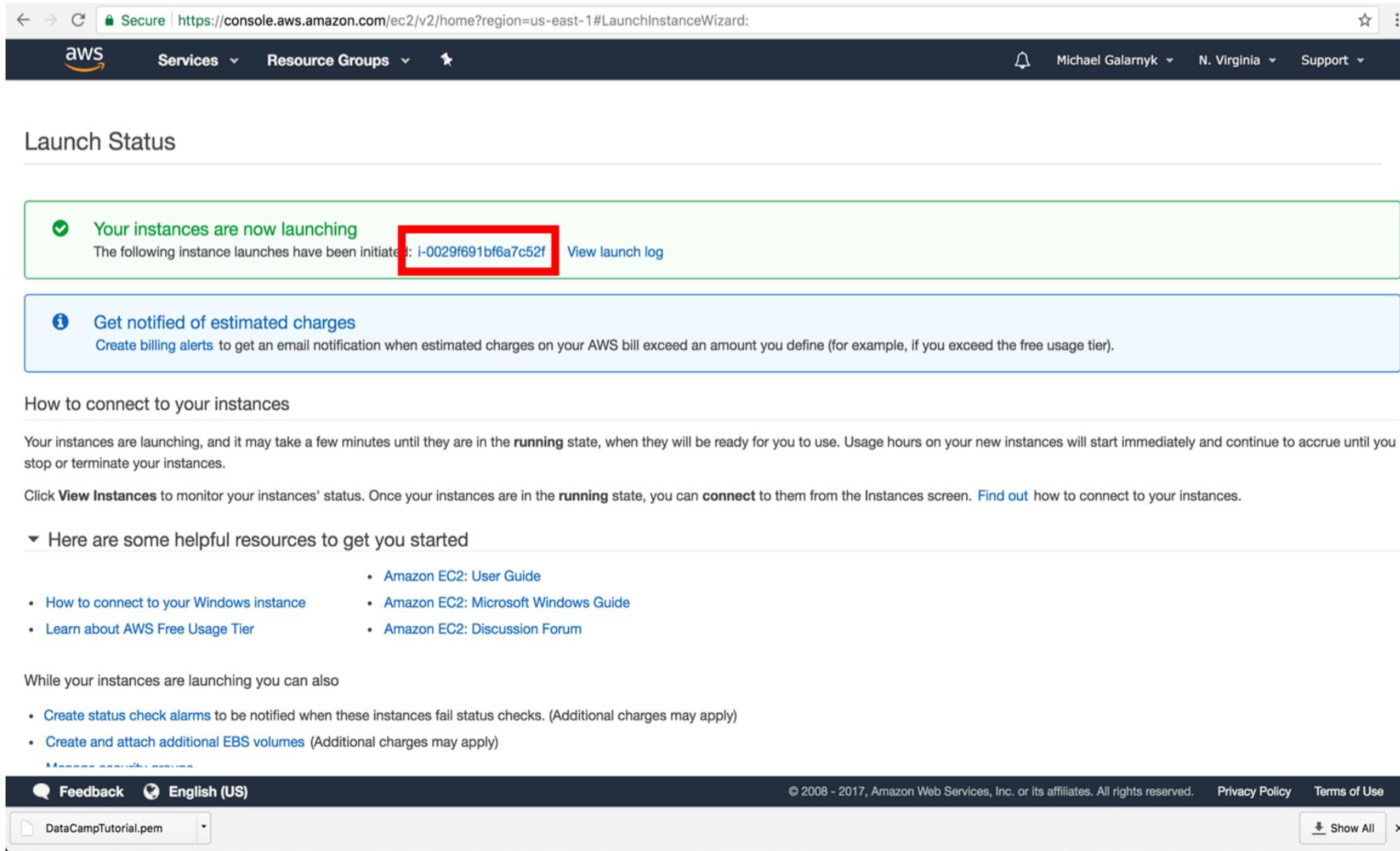
Feedback English (US)

© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use

DataCampTutorial.pem Show All

Demo: IaaS on AWS (Compute as a service)

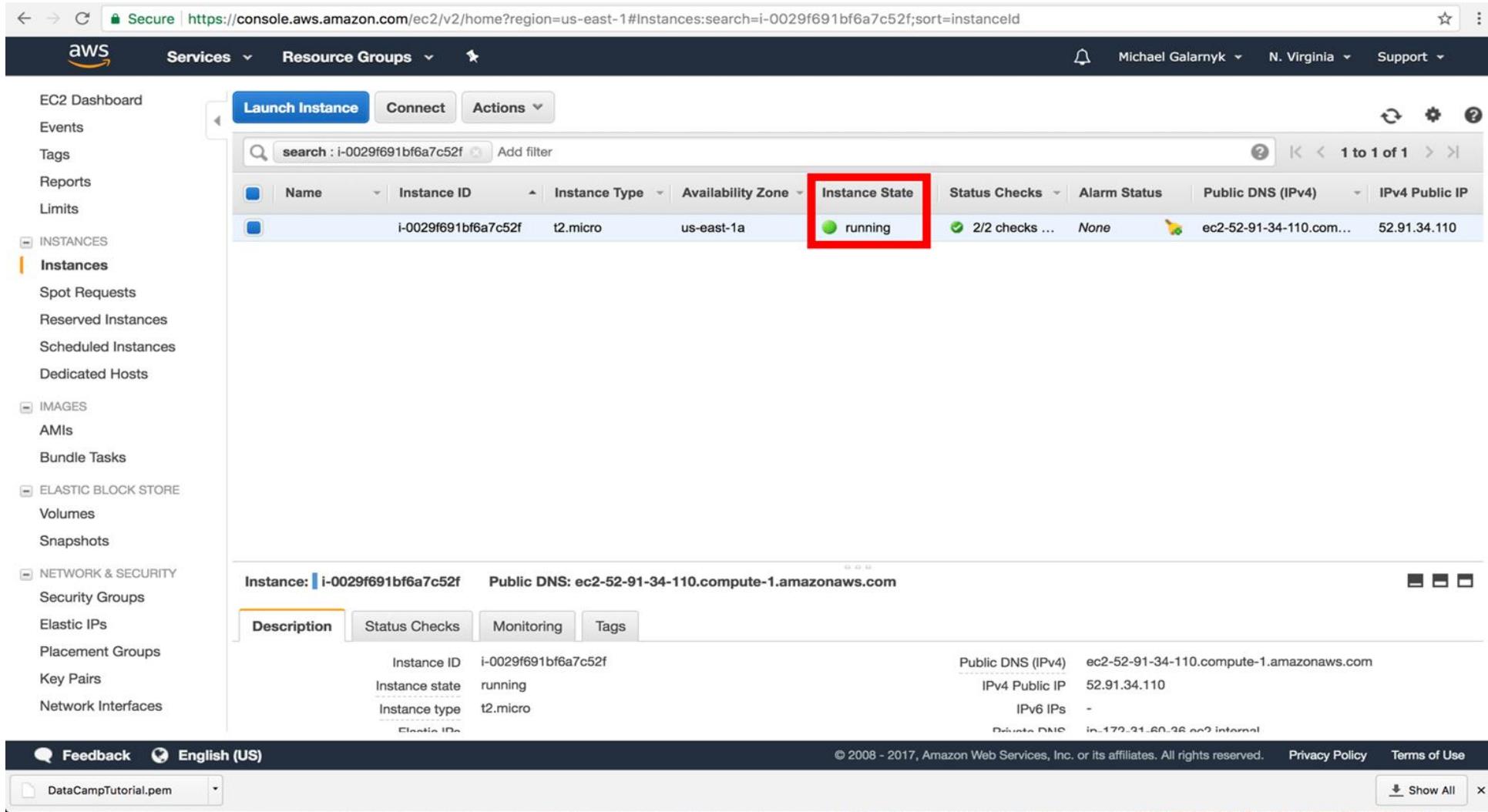
- Step 10: The instance is now launched. Go back to the Amazon EC2 console.



The screenshot shows the AWS EC2 Launch Status page. At the top, there's a navigation bar with links for Services, Resource Groups, and Support, along with user information for Michael Galarnyk and N. Virginia. Below the navigation, the main content area is titled "Launch Status". It displays a green box with a checkmark and the message "Your instances are now launching". Inside this box, it says "The following instance launches have been initiated: i-0029f691bf6a7c52f" and a "View launch log" link. A red rectangle highlights the instance ID "i-0029f691bf6a7c52f". Below this, there's a blue box with an info icon and the message "Get notified of estimated charges". It includes a link to "Create billing alerts" and a note about receiving email notifications when estimated charges exceed a defined amount. Further down, under "How to connect to your instances", it explains that instances are launching and may take a few minutes to reach the "running" state. It also suggests monitoring instances via the "View Instances" link and provides a "Find out" link for connecting. A section titled "Here are some helpful resources to get you started" lists several links, including "Amazon EC2: User Guide", "How to connect to your Windows instance", "Learn about AWS Free Usage Tier", "Amazon EC2: Microsoft Windows Guide", and "Amazon EC2: Discussion Forum". At the bottom, there are links for "Feedback", "English (US)", "DataCampTutorial.pem", and "Show All".

Demo: IaaS on AWS (Compute as a service)

- Step 11: Observe the instance is up and running



The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, INSTANCES (with 'Instances' selected), Spot Requests, Reserved Instances, Scheduled Instances, Dedicated Hosts, IMAGES, AMIs, and Bundle Tasks. Below that are sections for ELASTIC BLOCK STORE (Volumes, Snapshots) and NETWORK & SECURITY (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces). The main content area has tabs for Launch Instance, Connect, and Actions. A search bar at the top right says "search : i-0029f691bf6a7c52f". The main table has columns for Name, Instance ID, Instance Type, Availability Zone, and Instance State. The first row shows an instance named "i-0029f691bf6a7c52f" with type "t2.micro" in "us-east-1a" and the status "running". A red box highlights the "running" status. Below the table, there's a detailed view for the instance "i-0029f691bf6a7c52f" with Public DNS "ec2-52-91-34-110.compute-1.amazonaws.com". The "Description" tab is selected, showing Instance ID, Instance state (running), Instance type (t2.micro), and other details like Public DNS (IPv4), IPv4 Public IP, IPv6 IPs, and Private DNS.

- **Flexible:** The most flexible cloud computing model.
- **Control:** Clients retain complete control of their infrastructure
- **Pay-as-you-Go:** Unlike traditional IT, IaaS does not require any upfront, capital expenditures, and end users are only billed for what they use.
- **Speed:** With IaaS, users can provision small or vast amounts of resources in a matter of minutes, testing new ideas quickly or scaling proven ones even quicker.
- **Availability:** Through things like multizone regions, the availability and resiliency of cloud applications can exceed traditional approaches.
- **Scale:** With seemingly limitless capacity and the ability to scale resources either automatically or with some supervision, it's simple to go from one instance of an application or workload to many.
- **Latency and performance:** Given the broad geographic footprint of most IaaS providers, it's easy to put apps and services closer to your users, reducing latency and improving performance.

- **Startups and small companies** may prefer IaaS to avoid spending time and money on purchasing and creating hardware and software.
- **Larger companies** may prefer to retain complete control over their applications and infrastructure, but they want to purchase only what they actually consume or need.
- **Companies experiencing rapid growth** like the scalability of IaaS, and they can change out specific hardware and software easily as their needs evolve.
- **Website hosting** :Running websites using IaaS can be less expensive than traditional web hosting.
- **Storage, backup and recovery** : Organizations avoid the capital outlay for storage and complexity of storage management, which typically requires a skilled staff to manage data. IaaS is useful for handling unpredictable demand and steadily growing storage needs. It can also simplify planning and management of backup and recovery systems.
- **High-performance computing.** High-performance computing (HPC) on supercomputers, computer grids or computer clusters helps solve complex problems involving millions of variables or calculations.
- **Big data analysis.** Mining data sets to locate or tease out hidden patterns requires a huge amount of processing power, which IaaS economically provides.

Anytime you are unsure of a new application's demands, IaaS offers plenty of flexibility and scalability.

- **Security:** While the customer is in control of the apps, data, middleware, and the OS platform, security threats can still be sourced from the host or other virtual machines (VMs). Insider threat or system vulnerabilities may expose data communication between the host infrastructure and VMs to unauthorized entities.
- **Multi-tenant security:** Since the hardware resources are dynamically allocated across users as made available, the vendor is required to ensure that other customers cannot access data deposited to storage assets by previous customers. Similarly, customers must rely on the vendor to ensure that VMs are adequately isolated within the multi tenant cloud architecture.
- **Internal resources and training:** Additional resources and training may be required for the workforce to learn how to effectively manage the infrastructure. Due to inadequate control into the infrastructure however, monitoring and management of the resources may be difficult without adequate training and resources available in house.

- Amazon Web Services : Amazon EC2.
- Microsoft Azure : Azure Virtual Machines.
- Google Cloud : Google compute engine.
- IBM Cloud : IBM Cloud Private.
- Digital Ocean : Digital Ocean Droplets.



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

REST and Web Services

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Service oriented architecture (SOA)

SOA, or **service-oriented architecture**, defines a way to make software components reusable via service interfaces. These interfaces utilize common communication standards in such a way that they can be rapidly incorporated into new applications without having to perform deep integration each time.

Each service in an SOA embodies the code and data integrations required to execute a complete, discrete business function. The service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the integration is implemented underneath. The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or JSON/HTTP—to send requests to read or change data. The services are published in a way that enables developers to quickly find them and reuse them to assemble new applications.

Two major service-oriented architecture styles :

1. **REST** (REpresentational State Transfer)
2. **SOAP based WS** (Web Services)

REST(Representational State Transfer)

- Cloud architecture involves number of distributed autonomous systems or components or applications frequently communicating or interacting between themselves for performing an application request.
- REST is an architectural (sometimes also called programming) style for providing a set of rules to be used for building computer systems on the web, and making it easier for systems to communicate with each other.
- REST helps in building a client-friendly distributed systems that are simple to understand and simple to scale or following the REST principles while designing your application (distributed system), you will end up with a system that exploits the Web's architecture to your benefit
- An API which adheres to these constraints are considered RESTful and helps getting benefits of a Client Server distributed architecture by reducing the complexities of distributed systems (often called RESTful systems) and allows the benefits to be achieved more easily.
- REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web APIs. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data.

CLOUD COMPUTING

REST(REpresentational State Transfer) - Design Principles

The following are some of the mandatory constraints (principles) for designing a RESTful system.

1. Client Server Constraint (Separation of concerns)

This sets the constraint on the system to be built in such a fashion where the client can change and evolve without having to change anything on the server, and also on the contrary different aspects of the Server should also be able to be changed without breaking clients. Or

Its about how the client sends the server a message, and how the server rejects or responds to the client

2. Stateless Constraint

The communication between the client and the server must remain stateless between requests. Each request the client makes should contain all information needed for the server to answer successfully. All of the state information should be transferred back to the client as part of the response and cannot take advantage of any stored context on the server.

Session state is therefore kept entirely on the client. This is the “S” and “T” in REST; we are always giving the state back to the client. Not keeping the state in the Server impacts performance which are being addressed by the other concerns.

3. Cache Constraint

In order to improve network efficiency, cache constraints are added require that the data within a Server's response to a request, be implicitly or explicitly labeled/mark as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

CLOUD COMPUTING

REST(Representational State Transfer)

Constraints to designing a RESTful system (Cont.)

4. Uniform Interface Constraint

A uniform interface makes it generic and improves the overall visibility of interactions on how the client and server exchange requests and responses. Implementations are decoupled from the services they provide. This could impact performance for non web based data interactions. REST is defined by 4 interface constraints

- a. Resource and Resource Identification
- b. Manipulation of Resources through Representations
- c. Self-descriptive messages
- d. Hypermedia as the engine of application state

These will be discussed further.

5. Layered System Constraint

Messages between a client and server can go through a hierarchy of intermediate components (could be load balancers, proxy servers, firewalls, or other types of devices) where these intermediate components should not be able to “see” the next layer. Thus the interaction between the client and server should not be affected by these devices and thus ensures independence and bounding of complexity. All communication should remain consistent, even if one of the layers is added or removed

REST(Representational State Transfer) Functioning

- The REST architectural style is designed for network-based applications, specifically client-server applications.
- Clients can only access resources using **URIs**.
- Client requests a resource using a URI and the server responds with a **representation** of the resource.
- To ensure responses can be interpreted by the widest possible number of client **applications** a representation of the resource is sent in **hypermedia** format.
- Thus, a resource is manipulated through hypermedia representations transferred in **messages** between the clients and servers.



CLOUD COMPUTING

REST Architectural Elements – Resources and Resource Identification through URIs (T2)

Resource and Resource Identification: The key abstraction of information in REST is a resource. A resource is a “Thing”. Any information that can be named can be a resource, such as a document or image or a temporal service.

Eg. Light on board in seminar hall. If you need to control it, you name it

/pesu/b-block/groundfloor/seminarhall/board/light/1

The RESTful web service exposes a set of resources which identify targets of interaction with its clients. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability.

URI – Uniform Resource Identifier	URL – Uniform Resource Locator
<ul style="list-style-type: none">▪ The name of the object on the web▪ Identifies a resource by<ul style="list-style-type: none">▫ Name▫ Location▫ Or both	<ul style="list-style-type: none">▪ Subset of an URI▪ Specifies where to find a resource – the location▪ How to retrieve the resource

Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) Interfaces which supports operations as below :

GET — retrieve a specific resource or a collection of resources

POST — Create or partial update of a resource

PUT — Create or update resource by replacement.

DELETE — remove a resource

So typically in your application clients say for a book lending library something like

- GetListOfBooks()
- AddBookToShoppingCart()

Need to define these operations. No standard style exists. These operations are implemented using the Interfaces described earlier.

- REST Operations

GET

- Retrieve representation of a resource

PUT

- Create or update resource by replacement.

POST

- Create or partial update of a resource

DELETE

- Remove a resource

- Usage

GET

- <https://bblock/seminarhall/board/light1.xml>

- To check if light is ON

POST to

- <https://bblock/seminarhall/board/light1.xml>

- To turn on/off the light

- Put ON in the body of the message

CLOUD COMPUTING : Operations

- When we treat resources as things to manipulate via a URL and we restrict ourselves to a small set of operations, there is less need to extensively describe a service.
- It's not about the 50 things you can do with a particular service (which is how SOAP developers view the world). It's about using a standard set of things you can do (PUT, GET, etc.) with a set of resources (robots, spaceships, flightpaths, etc.)
- So how do you FlyShip (`id="ship-123"`, `destination=Planets.Mars`) ?
 - PUT to `/ships/ship-123/flightpath` and send "destination=mars" in the body of the message
 - Now you can poll that URI using GET for flight updates.
 - or
 - POST to `/ships/ship-123/flightpaths` and send "/planets/mars" in the body of the message. The server might return something like `/ships/ship-123/flightpaths/456` to indicate that the ship is now flying to Mars and that a new flightpath resource was created to track it. You can poll that URI using GET for flight updates.
- Instead of defining a new operation such as `StopFlight (id)` to stop the flight, do this:
 - DELETE `/ships/ship-123/flightpaths/456`
- How does a web browser know what to do with a URL? All resources support the same operations, and GET is one of them. The browser GETs until everything has been gotten!

CLOUD COMPUTING : Characteristics of these Operations: Safe - Idempotent

- Safe
 - Does not modify the resources
 - Example
 - GET
- Idempotent
 - Idempotent has no additional effect if it is called more than once with same input parameters
 - Can repeatedly perform operations.
 - No effect on servers
 - How would you like to pay for a seat multiple times?

CLOUD COMPUTING : Operations with characteristics

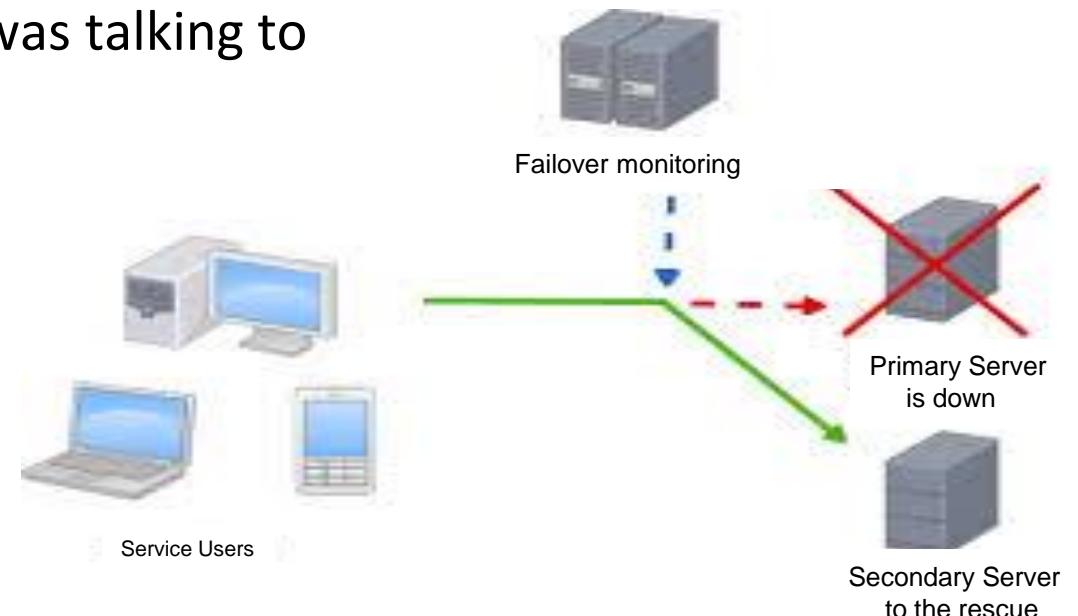
Operation	Safe	Idempotent	When to use
GET	Yes	Yes	Mostly for retrieving resources. Can call multiple times.
PUT	No	Yes	Modifies a resource but no additional impact if called multiple times
POST	No	No	Modifies resources, multiple calls will cause additional effect if called with same parameter
DELETE	No	Yes	Removing a resource.

Stateless Interactions: Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa. In this way, both the server and the client can understand any message received, even without seeing previous messages. Each client request is treated independently.

Statelessness benefits :

1. Clients isolated against changes on server
2. Promotes redundancy - unlocks performance:
 - a. don't really need to know which server client was talking to
 - b. No synchronization overhead

No state saved on server,
so even if server fails, the
client connects to another
server and continue



CLOUD COMPUTING

REST Principles - Representations

Representation : A representation is a snapshot in time of the state of a given resource. It's a sequence of bytes made up of the data, plus representation metadata to describe those bytes.

- This captures the current or intended state of that resource and helps transferring that representation between the interacting components.
- The metadata could be in the form of binary or textual key-value pairs.
- The data format of a representation is known as a media type and this type identifies a specification that defines how a representation is to be processed.
- The message type is ***hypermedia***, which refers to any content that contains links to other forms of media such as images, movies, and text. hypermedia links in the API response contents. It allows the client to dynamically navigate to the appropriate resources by traversing the hypermedia links.
- Navigating hypermedia links is conceptually the same as browsing through web pages by clicking the relevant hyperlinks to achieve a final goal.
- All the response representation need to be self-descriptive

A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents.

In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.) i.e. the resources can have **multiple representations**. RESTful systems empowers client to ask for data in a form they understand.

Example:

```
GET /articles/23 HTTP/1.1
Accept: text/html, application/xhtml
```

Client request

```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

Server Response

Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

1. Rest is not a standard :

It is a design and architectural style for large scale distributed systems

2. Rest is not same as http:

Http can also be used in non-RESTful way. Example : SOAP services that use http to transport data.

Web Service: a software system designed to support interoperable machine-to-machine interaction over a network.

The term “web service” is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software applications across the web.

Once a web service is deployed, other applications and other web services can discover and invoke the deployed service. Other systems interact with the web service in a manner prescribed by its description.

A web service is one of the most common instances of an SOA implementation.

The two prominent ways of implementing Web Services. This could be using the approach of :

1. Simple Object Access Protocol (SOAP)
2. REST (we have discussed this at length)

Simple Object Access Protocol (SOAP)

SOAP is a protocol which was designed before REST came into the picture. The main idea behind creating SOAP was to ensure that programs built on different platforms and programming languages could securely exchange data.

SOAP provides a structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP.

A SOAP message consists of element called **envelope**, which is used to encapsulate all of the data in the SOAP message which contains a **Header** element that contains header information such as **authentication credentials** which can be used by the calling application, and a **body** element that carries the payload of the message.

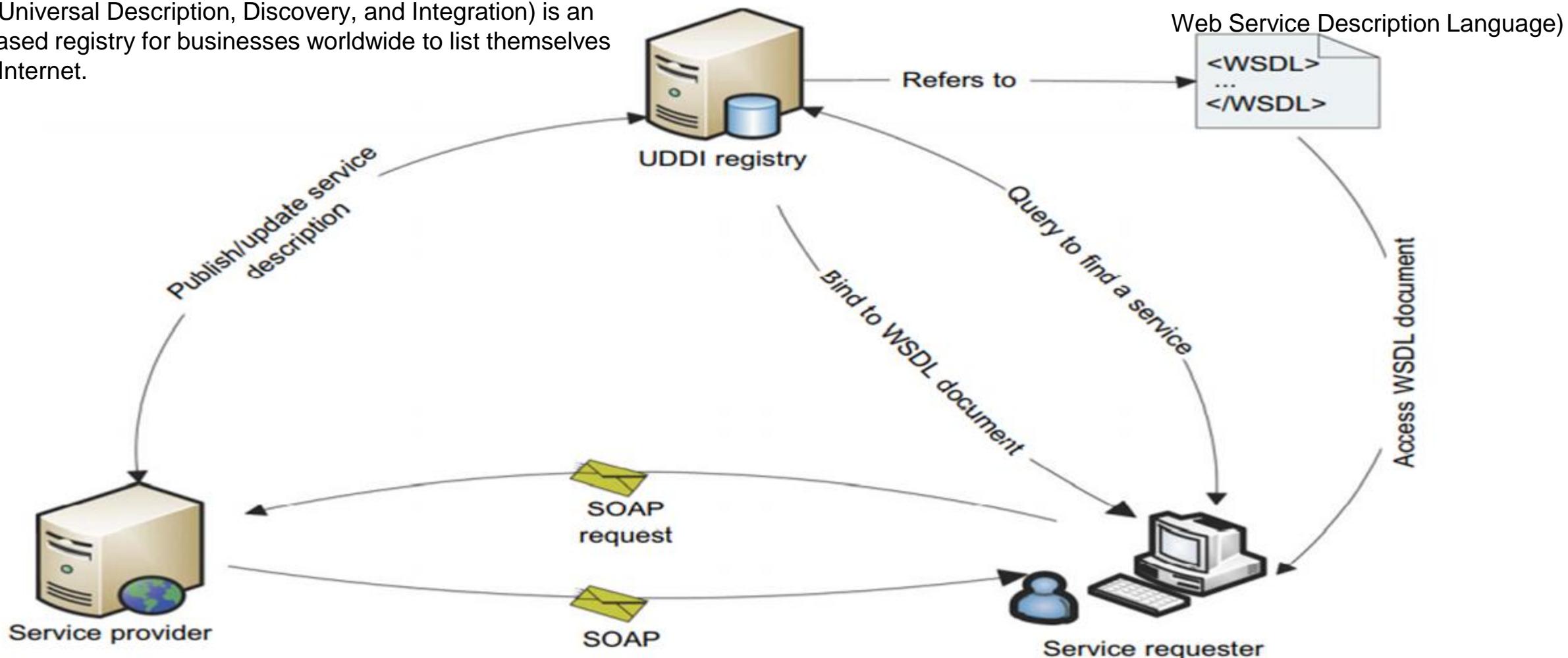
The content of the payload will be marshaled by the sender's SOAP engine and un-marshaled at the receiver side.

WSDL or web service description language describes the functionality of the SOAP based webservice

Web Services in action say using SOAP

(T2)

UDDI (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet.



Refer to Figure 5.2(previous slide):

Step 1 : The service provider publishes its location and description to the UDDI registry.

Step 2: The service requester queries the UDDI registry using names, identifiers, or the specification. The UDDI registry finds the corresponding service and returns the WSDL document of the service.

Step 3: The service requester reads the WSDL document and forms a SOAP message binding to all constraints in the WSDL document.

Step 4: This SOAP message is sent to the web service as the body of an HTTP/SMTP/FTP request.

Step 5: The web service unpacks the SOAP request and converts it into a command that the application can understand and executes the command.

Step 6: The web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP/SMTP/FTP request

Step 7: The client program unpacks the SOAP message to obtain the results



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Platform as a Service(PaaS) Programming Model

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

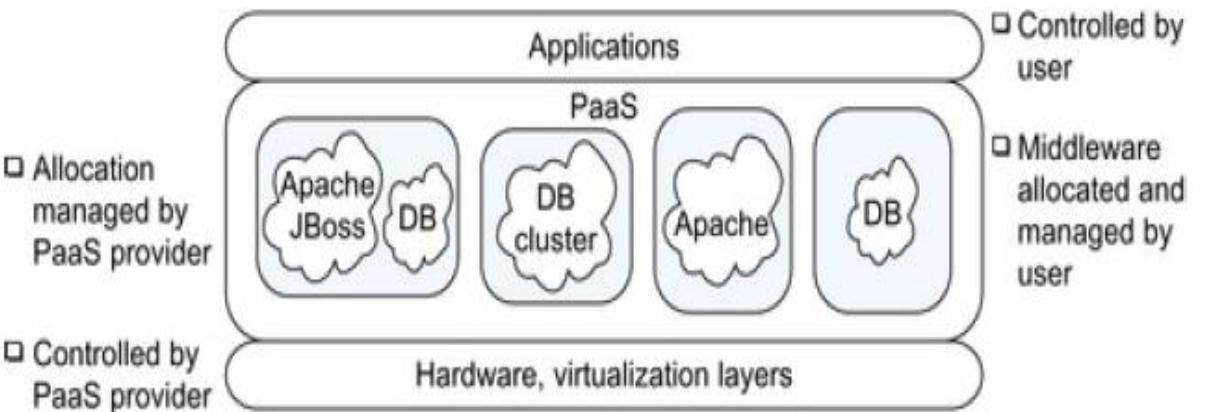
Acknowledgements:

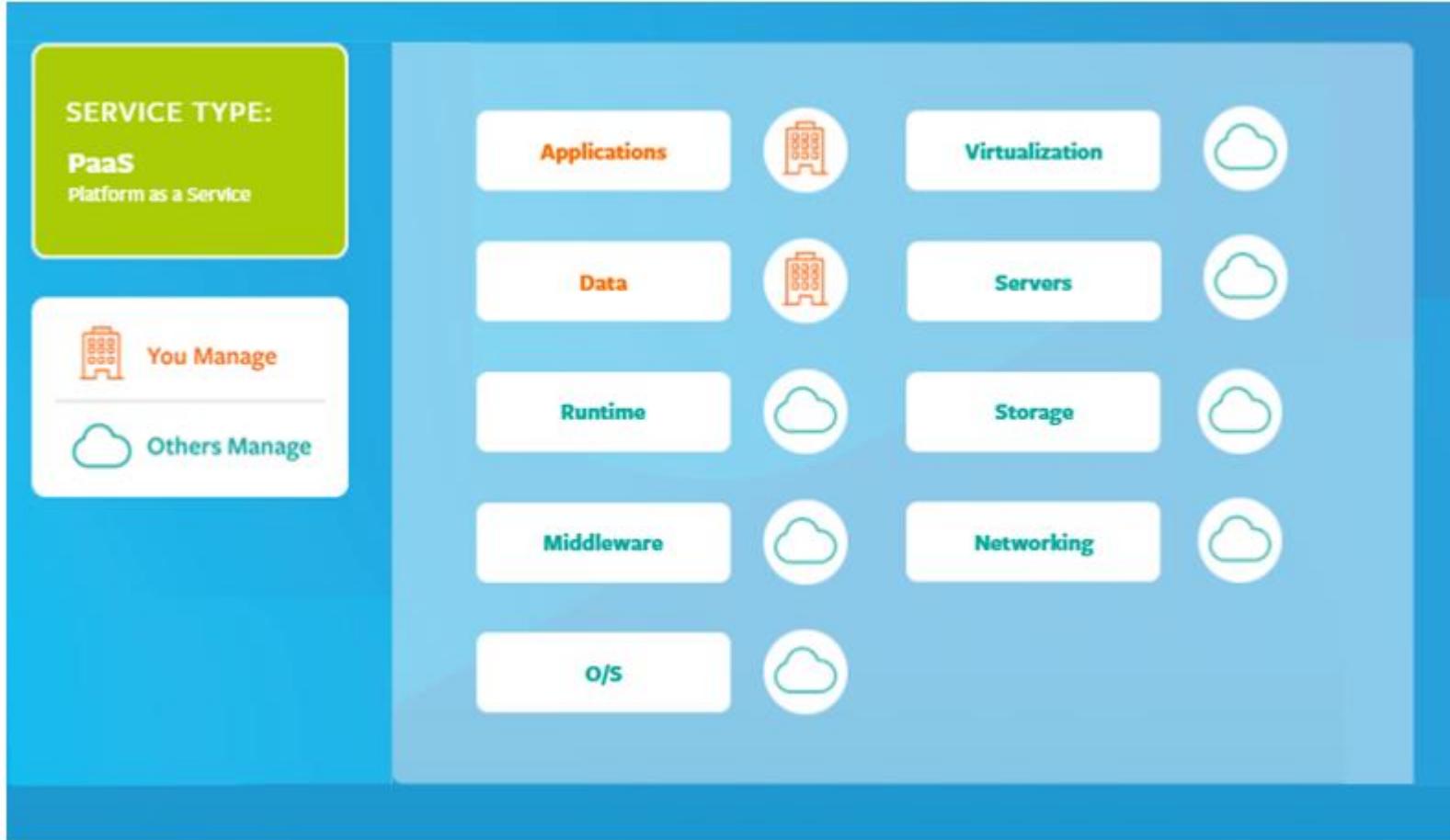
Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

Platform as a Service(PaaS) Programming Model (Recap)

Platform as a service (PaaS)

- It's a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications.
- You purchase the resources you need from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection.
- Like IaaS, PaaS includes not only the abstracted infrastructure—servers, storage and networking—but also middleware, development tools, business intelligence (BI) services, database management systems and more. PaaS is designed to support a Cloud application like a web application across the complete lifecycle: building, testing, deploying, managing and updating.
- This platform is delivered via the web, giving developers the freedom to concentrate on building the software without having to worry about operating systems, software updates, storage, or infrastructure.
- The hardware, as well as any mapping of hardware to virtual resources, such as virtual servers, is controlled by the PaaS provider. The cloud user can configure and build on top of this middleware, like defining a new database table in a database





Ref - <https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/#ref3>

PaaS Advantages - Reiteration

- 1. Faster time to market:** With PaaS, there's no need to purchase and install the hardware and software you'll use to build and maintain your application development platform and no need for development teams to wait while you do this. You simply tap into the cloud service provider's PaaS resources and begin developing immediately.
- 2. Faster, easier, less-risky adoption of a wider range of resources:** PaaS platforms typically include access to a greater variety of choices up and down the application development stack—operating systems, middleware, and databases, and tools such as code libraries and app components—than you can affordably or practically maintain on-premises. It also lets you test new operating systems, languages, and tools without risk—that is, without having to invest in the infrastructure required to run them.
- 3. Develop for multiple platforms—including mobile—more easily.** Some service providers give you development options for multiple platforms, such as computers, mobile devices and browsers making cross-platform apps quicker and easier to develop.
- 4. Easy, cost-effective scalability:** If an application developed and hosted on-premises starts getting more traffic, you'll need to purchase more computing, storage, and even network hardware to meet the demand, which you may not be able to do quickly enough and can be wasteful (since you typically purchase more than you need). With PaaS, you can scale on-demand by purchasing just the amount of additional capacity you need.

PaaS Advantages - Reiteration (Cont.)

5. **Support geographically distributed development teams:** Because the development environment is accessed over the Internet, development teams can work together on projects even when team members are in remote locations.
6. **Efficiently manage the application lifecycle :** PaaS provides all of the capabilities that you need to support the complete web application lifecycle: building, testing, deploying, managing and updating within the same integrated environment.
7. **Lower costs:** Because there's no infrastructure to build, your upfront costs are lower. Costs are also lower and more predictable because most PaaS providers charge customers based on usage.
8. **Development framework:** PaaS provides a framework that developers can build upon to develop or customize cloud-based applications. PaaS lets developers create applications using built-in software components. Cloud features such as scalability, high-availability and multi-tenant capability are included, reducing the amount of coding that developers must do.
9. **Analytics or business intelligence:** Tools provided as a service with PaaS allow organizations to analyze and mine their data, finding insights and patterns and predicting outcomes to improve forecasting, product design decisions, investment returns and other business decisions.

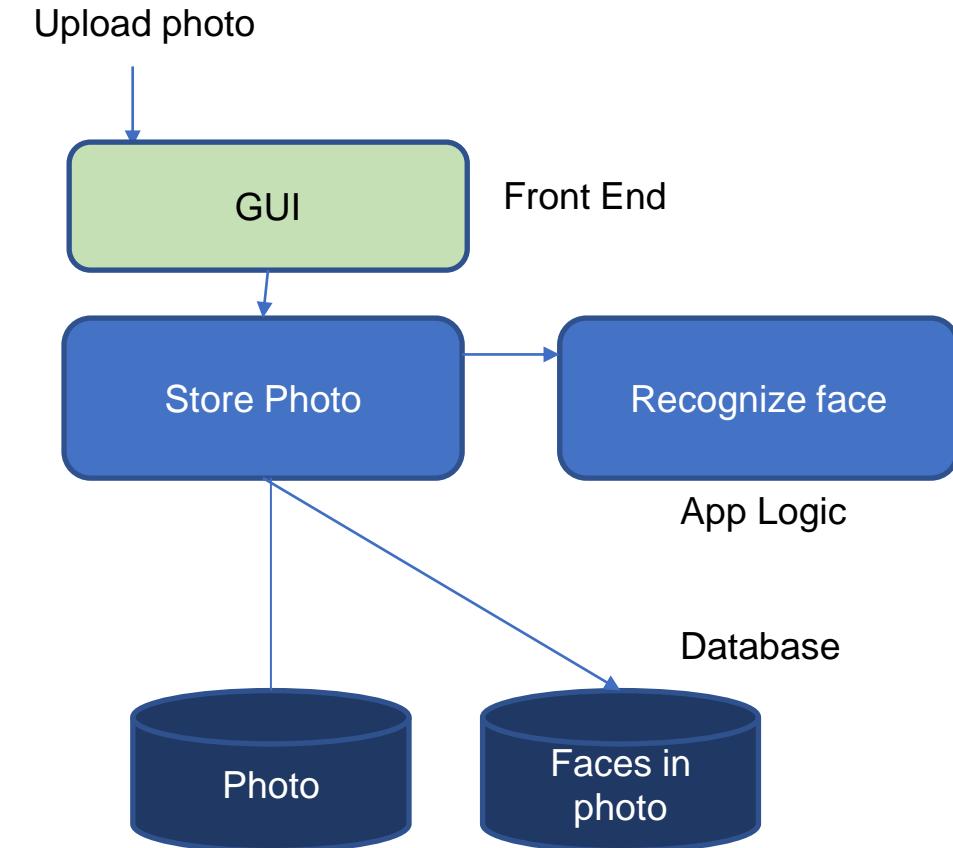
Consider you are building a photo sharing app

- User uploads Photo
- App recognizes faces in photo
- App Stores photos + faces in photo into database

If we are looking at a 3 tier application model, the layers would have something like shown.

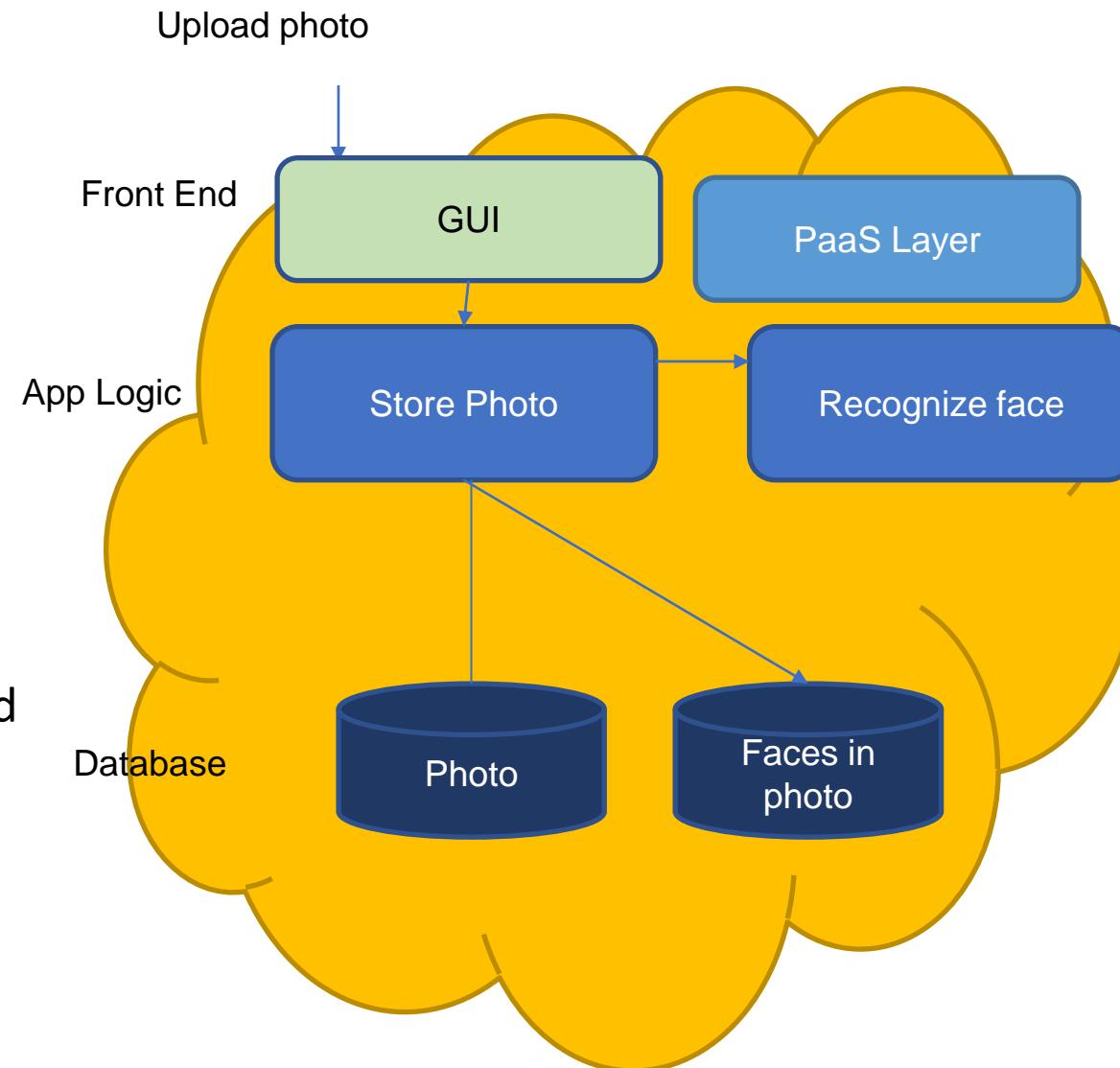
The activities in an IaaS programming model would be

- Start a VM
- Connect a block storage device
- Install Web Server
- Install App Server
- Install Database
- Upload application
- Run Application



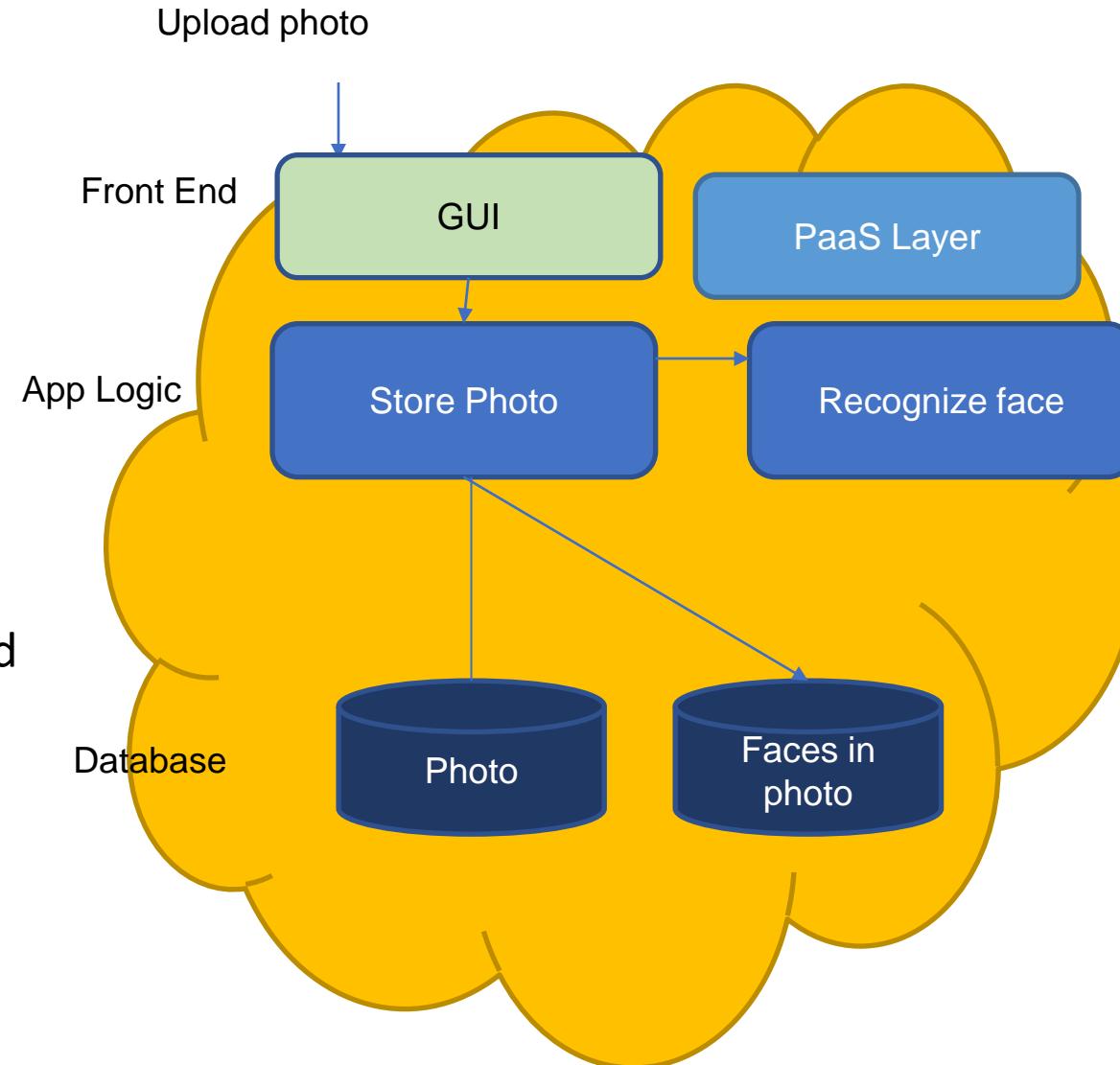
In the PaaS programming model, this would be

- Application is uploaded
 - This will be made up of components/sections
 - Each component specifies environment (e.g., part of code/actions to run under web server or it could be part of the code to run under app server)
 - Specifies initial number of components, scale up / scale down policy
- In Windows Azure
 - Web roles – front end type tasks
 - Worker roles – those that run tasks in the background (Different colours on the figure)
- PaaS Layer
 - Deploy
 - Allocate resources
 - Number of instances



In the PaaS programming model, this would be

- Application
 - Made up of components
 - Each component specifies environment (e.g., run under web server or app server)
 - Specifies initial number of components, scale up / scale down policy
- In Windows Azure
 - Web roles – front end type tasks
 - Worker roles – those that run tasks in the background (Different colours on the figure)
- PaaS Layer
 - Deploy
 - Allocate resources
 - Number of instances

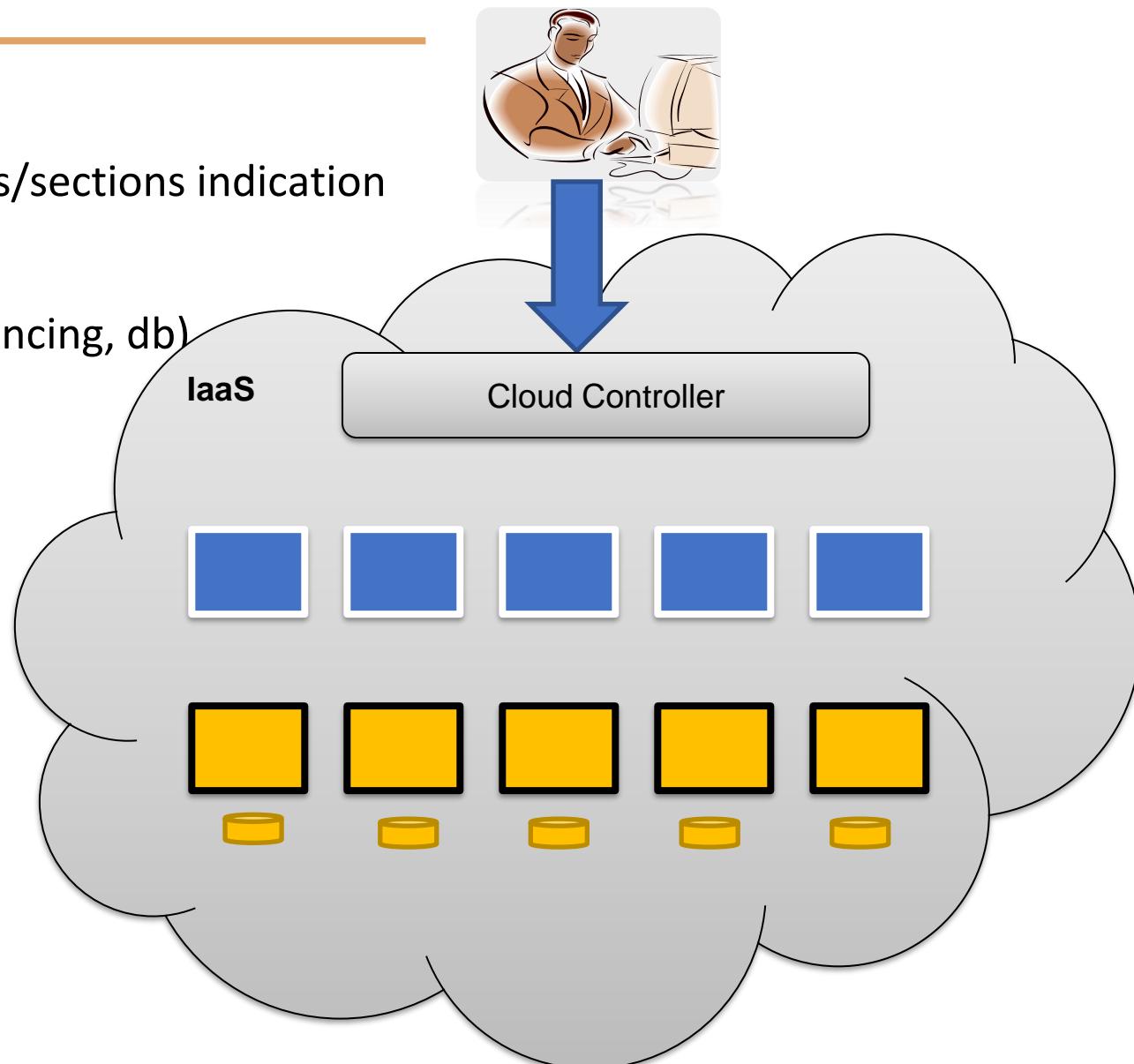


How is this built with PaaS as the Service Model

- What do we want to do?
 - User pushes into cloud
 - Application (with the different components/sections indication where it needs to done)
 - Specification of S/W (e.g., OS, Web server)
 - Services needed from cloud (e.g., load balancing, db)

How do we do this?

- Cloud controller
 - Automatically creates VMs and containers
 - Deploys applications
 - Connects to services
 - Automatically scales up or down
- Storage Services
 - Object
 - NoSQL
 - Relational
 - Block storage
- Applications store state in Storage Services
 - Simpler to scale applications
 - Easier recovery from failure



When to Use PaaS

PaaS can streamline workflows when multiple developers are working on the same development project. If other vendors must be included, PaaS can provide great speed and flexibility to the entire process. PaaS is particularly beneficial if you need to create customized applications.

API development and management: You can use PaaS to develop, run, manage, and secure application programming interfaces (APIs) and microservices.

Internet of Things (IoT): PaaS can support the broad range of application environments, programming languages, and tools used for IoT deployments.

PaaS Limitations & Concerns (Recap)

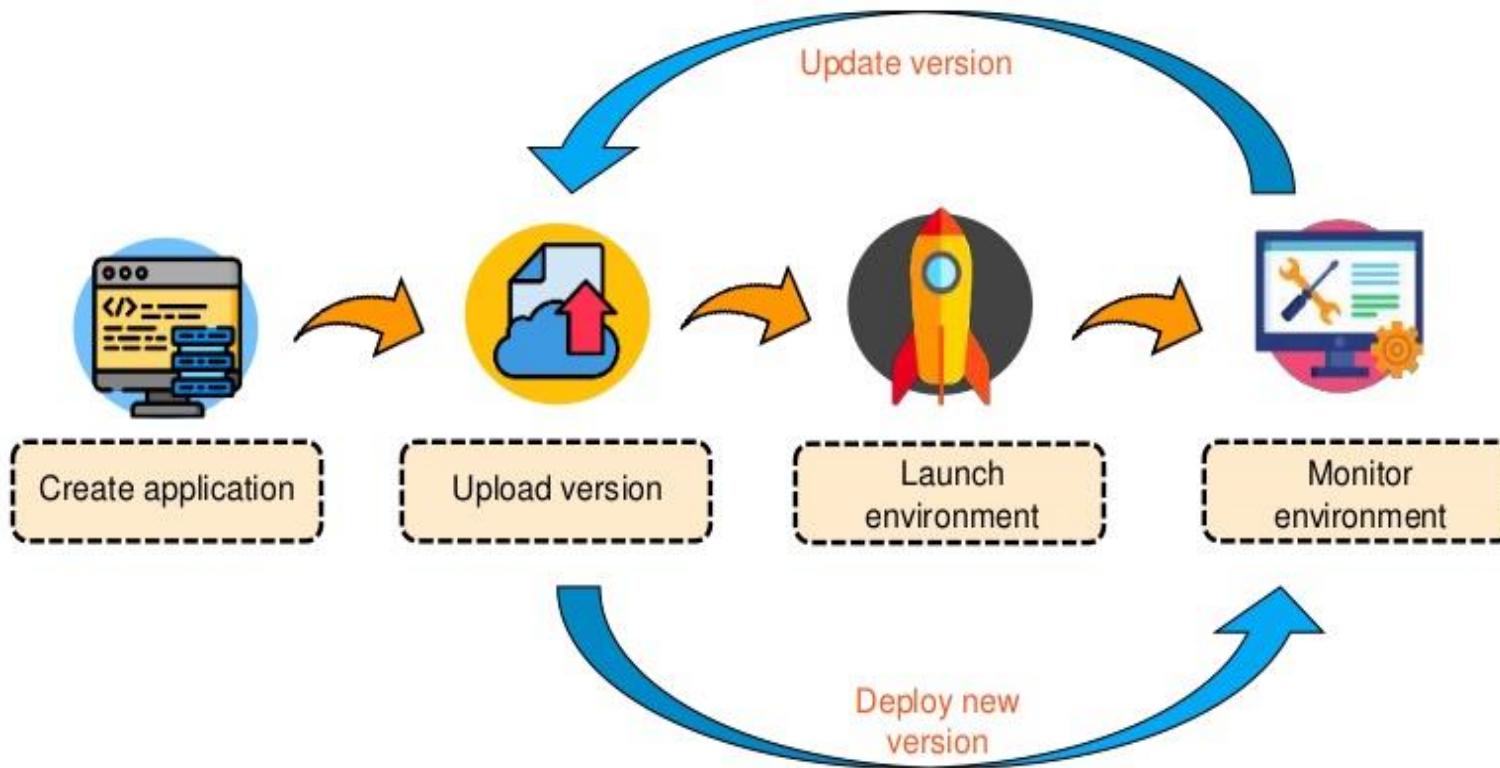
1. **Operational limitation(Lack of control):** Customized cloud operations with management automation workflows may not apply to PaaS solutions, as platform tends to limit operational capabilities for end users. Although this is intended to reduce the operational burden on end users, the loss of operational control may affect how PaaS solutions are managed, provisioned, and operated.
2. **Vendor lock-in:** Business and technical requirements that drive decisions for a specific PaaS solution may not apply in the future. If the vendor has not provisioned convenient migration policies, switching to alternative PaaS options may not be possible without affecting the business.
3. **Runtime issues:** In addition to limitations associated with specific apps and services, PaaS solutions may not be optimized for the language and frameworks of your choice. Specific framework versions may not be available or perform optimally with the PaaS service. Customers may not be able to develop custom dependencies with the platform.
4. **Data security:** Organizations can run their own apps and services using PaaS solutions, but the data residing in third-party, vendor-controlled cloud servers poses security risks and concerns. Your security options may be limited as customers may not be able to deploy services with specific hosting policies.
5. **Integrations:** The complexity of connecting the data stored within an onsite data center or off-premise cloud is increased, which may affect which apps and services can be adopted with the PaaS offering. Particularly when not every component of a legacy IT system is built for the cloud, integration with existing services and infrastructure may be a challenge.
6. **Customization of legacy systems:** PaaS may not be a plug-and-play solution for existing legacy apps and services. Instead, several customizations and configuration changes may be necessary for legacy systems to work with the PaaS service. The resulting customization can result in a complex IT system that may limit the value of the PaaS investment altogether.

- Amazon Web Services : Elastic Beanstalk.
- Microsoft Azure : Azure DevOps.
- Google Cloud : Google App Engine.

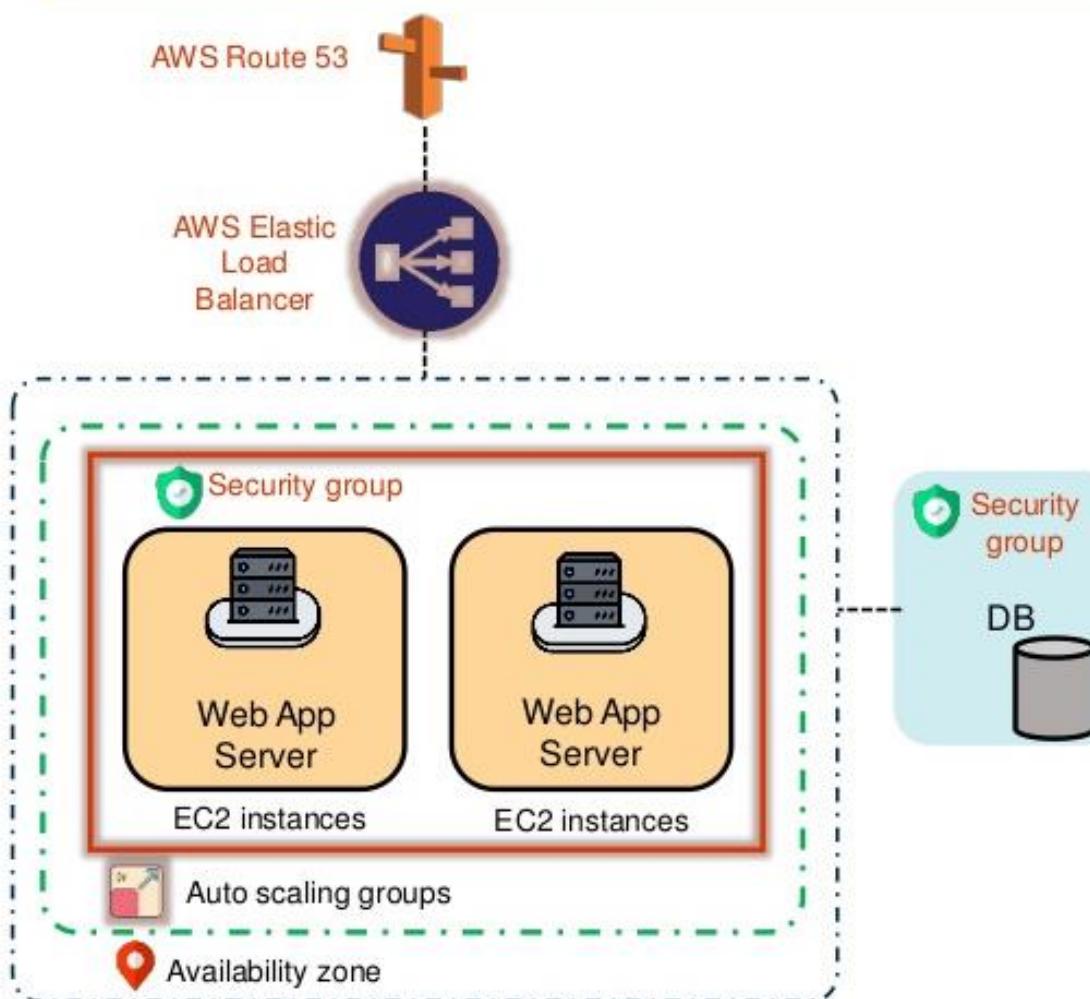
Table 4.2 Five Public Cloud Offerings of PaaS [10,18]

Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

How does Elastic Beanstalk in AWS work?



PaaS Demo : Architecture of AWS Elastic Beanstalk

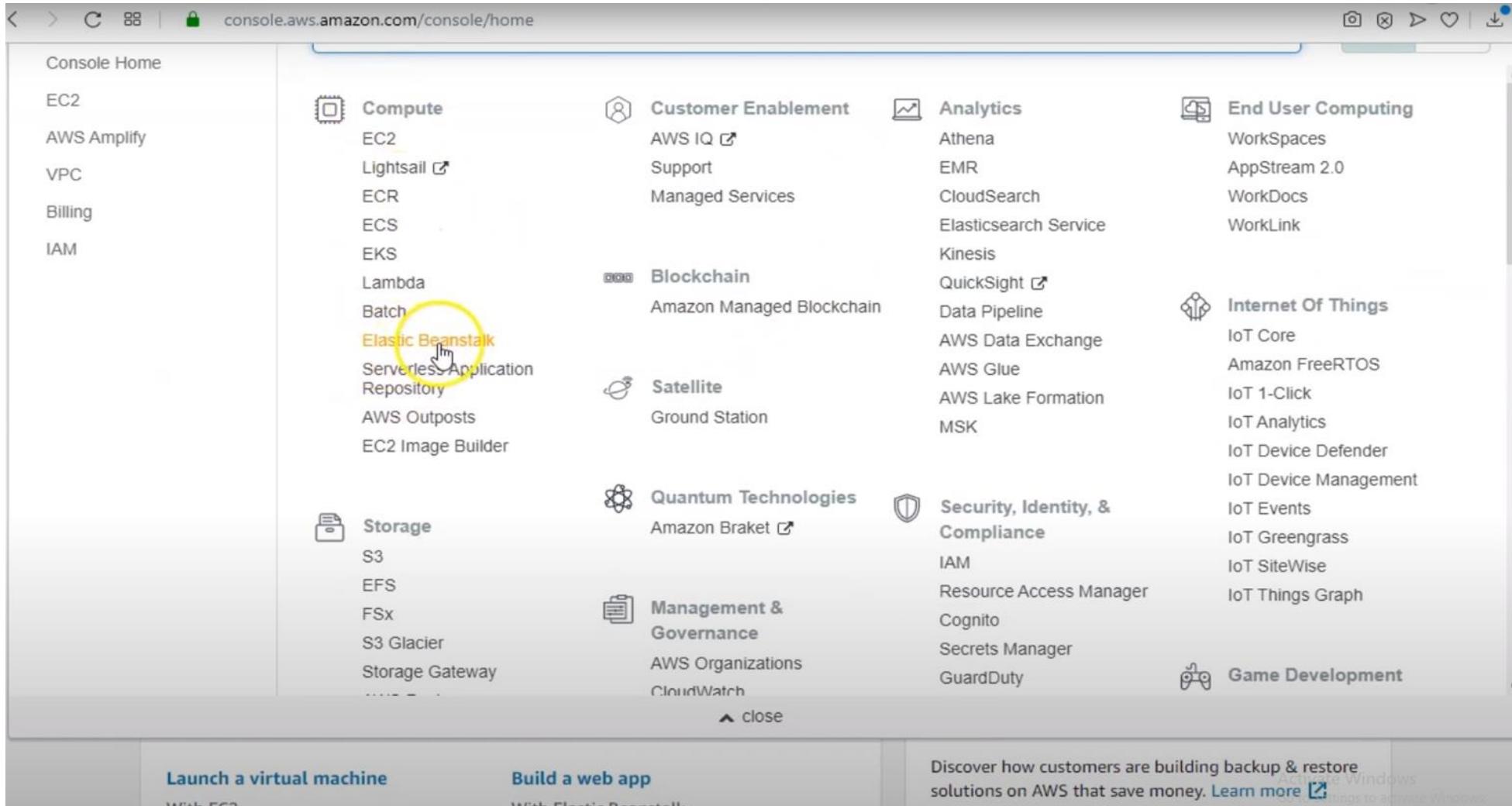


Web Server environment

- If the application receives client requests, Amazon Route53 sends these requests to the AWS Elastic Load Balancer.
- Later, AWS load balancer shares the requests among EC2 instances
- Every EC2 instances have a security group
- The load balancer is connected to Amazon EC2 instances, which are part of an Auto Scaling group

PaaS Demo : AWS Elastic Beanstalk

Step 1: Log into AWS, go to services and select Elastic Beanstalk



The screenshot shows the AWS Console Home page. On the left, there's a sidebar with links to EC2, AWS Amplify, VPC, Billing, and IAM. The main area is a grid of service icons. In the first row, 'Compute' is expanded, showing EC2, Lightsail, ECR, ECS, EKS, Lambda, Batch, and Elastic Beanstalk. 'Elastic Beanstalk' is highlighted with a yellow circle and a cursor icon pointing at it. Other services in this row include Customer Enablement (AWS IQ, Support, Managed Services), Analytics (Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis), and End User Computing (WorkSpaces, AppStream 2.0, WorkDocs, WorkLink). The second row includes Blockchain (Amazon Managed Blockchain), QuickSight, Data Pipeline, AWS Data Exchange, AWS Glue, AWS Lake Formation, and MSK. The third row includes Satellite (Ground Station), Internet Of Things (IoT Core, Amazon FreeRTOS, IoT 1-Click, IoT Analytics, IoT Device Defender, IoT Device Management, IoT Events, IoT Greengrass, IoT SiteWise, IoT Things Graph). The fourth row includes Quantum Technologies (Amazon Braket), Security, Identity, & Compliance (IAM, Resource Access Manager, Cognito, Secrets Manager, GuardDuty), and Game Development.

Console Home

EC2

AWS Amplify

VPC

Billing

IAM

Compute

Customer Enablement

Analytics

End User Computing

Blockchain

Satellite

Quantum Technologies

Management & Governance

Security, Identity, & Compliance

Game Development

Elastic Beanstalk

Serverless Application Repository

AWS Outposts

EC2 Image Builder

Storage

S3

EFS

FSx

S3 Glacier

Storage Gateway

Launch a virtual machine With EC2

Build a web app With Elastic Beanstalk

Discover how customers are building backup & restore solutions on AWS that save money. Learn more

PaaS Demo : AWS Elastic Beanstalk

Step 2: Click on Get started and then create new application

Welcome to AWS Elastic Beanstalk

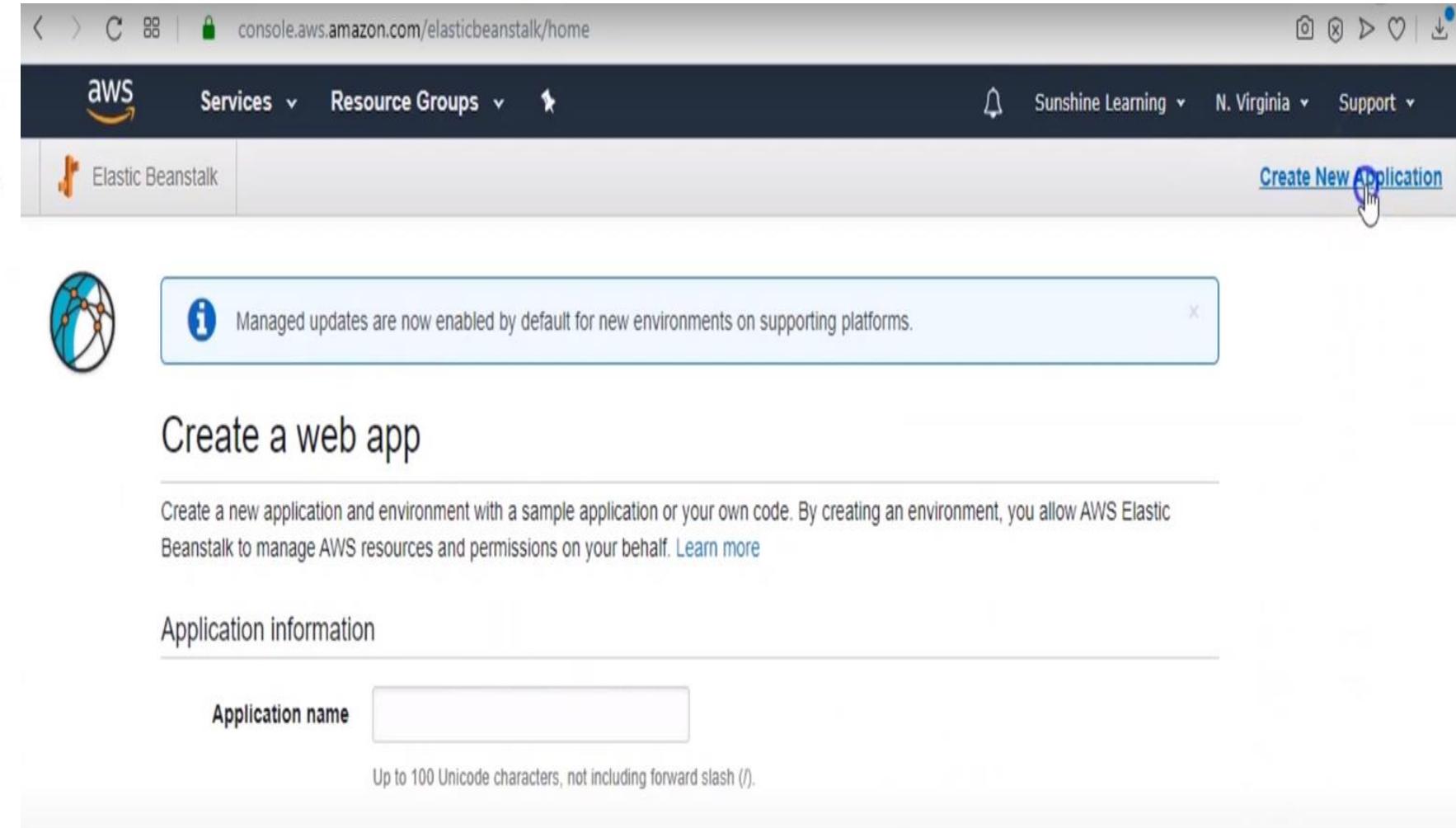
With Elastic Beanstalk, you can **deploy**, **monitor**, and **scale** an application quickly and easily. Let us do the heavy lifting so you can focus on your business.

To deploy your **existing web application**, create an **application source bundle** and then **create a new application**. If you're using **Git** and would prefer to use it with our command line tool, please see [Getting Started with the EB CLI](#).

To deploy a **sample application**, click **Get started**, choose a name, select a platform and click **Create app**.

By launching the sample application, you allow AWS Elastic Beanstalk to administer AWS resources and necessary permissions on your behalf. [Learn more](#)

Get started



console.aws.amazon.com/elasticbeanstalk/home

aws Services Resource Groups Sunshine Learning N. Virginia Support

Elastic Beanstalk Create New Application

Managed updates are now enabled by default for new environments on supporting platforms.

Create a web app

Create a new application and environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Application information

Application name

Up to 100 Unicode characters, not including forward slash (/).

PaaS Demo : AWS Elastic Beanstalk

Step 3: Enter application name, description and click on create

Create New Application

Application Name XYZ
Maximum length of 100 characters, not including forward slash (/).

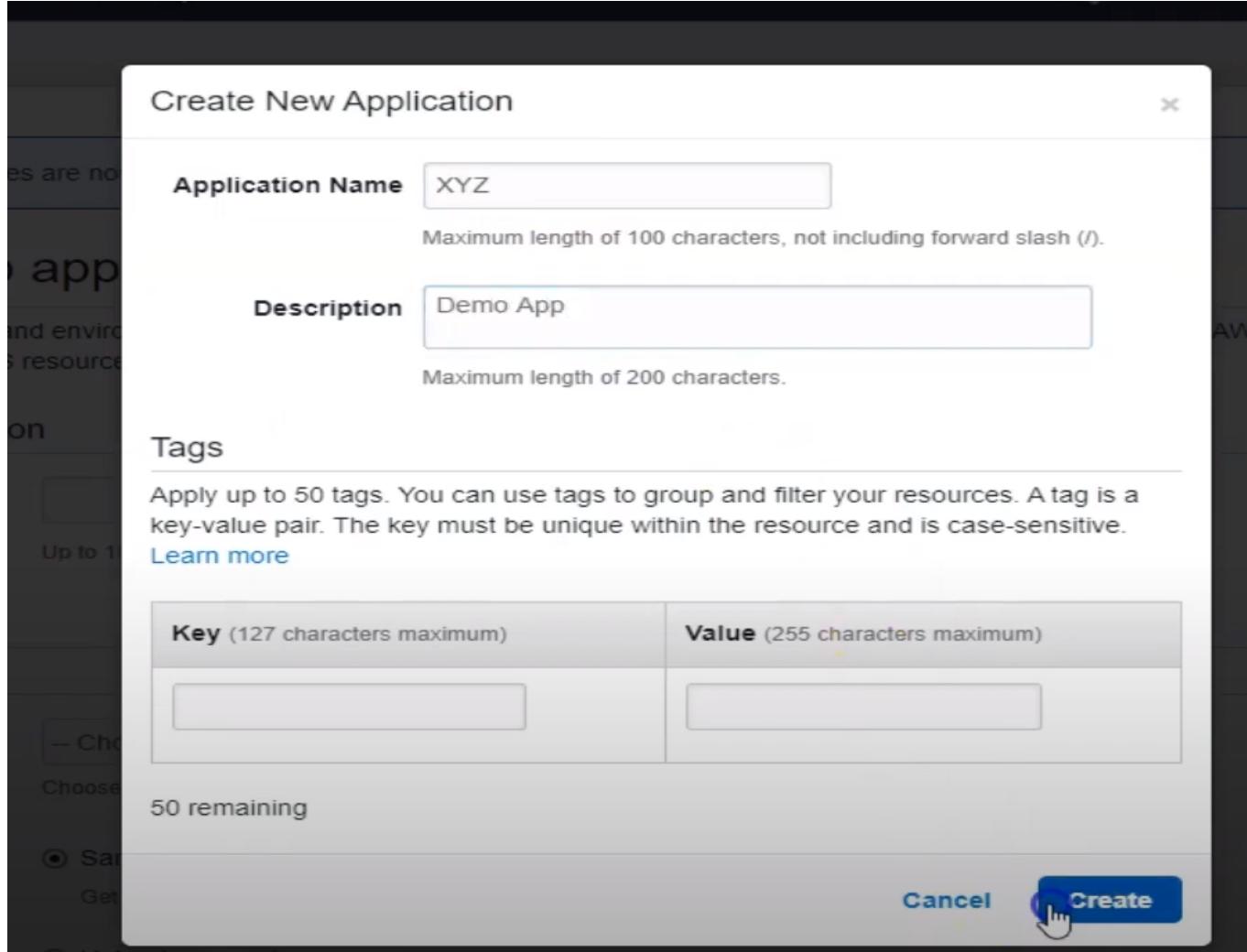
Description Demo App
Maximum length of 200 characters.

Tags
Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive.
[Learn more](#)

Key (127 characters maximum)	Value (255 characters maximum)
<input type="text"/>	<input type="text"/>

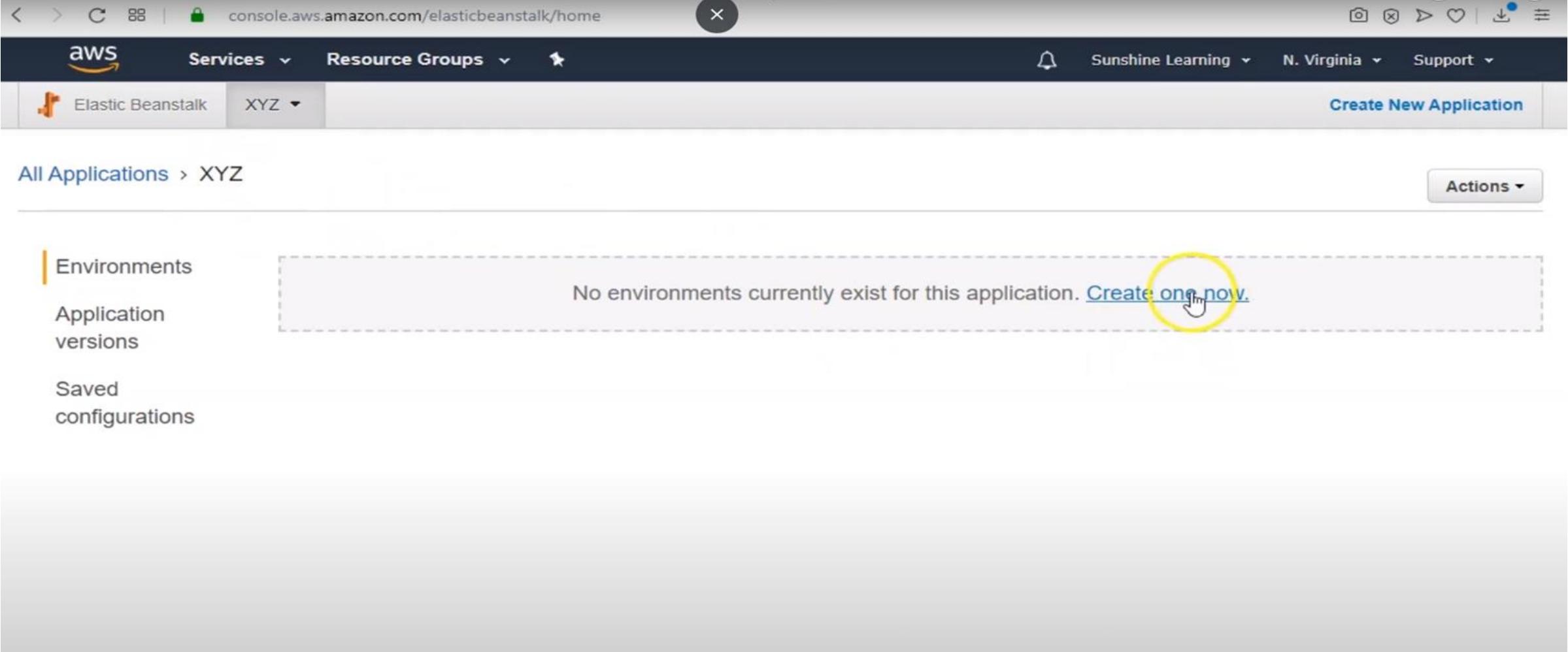
50 remaining

Cancel **Create**



PaaS Demo : AWS Elastic Beanstalk

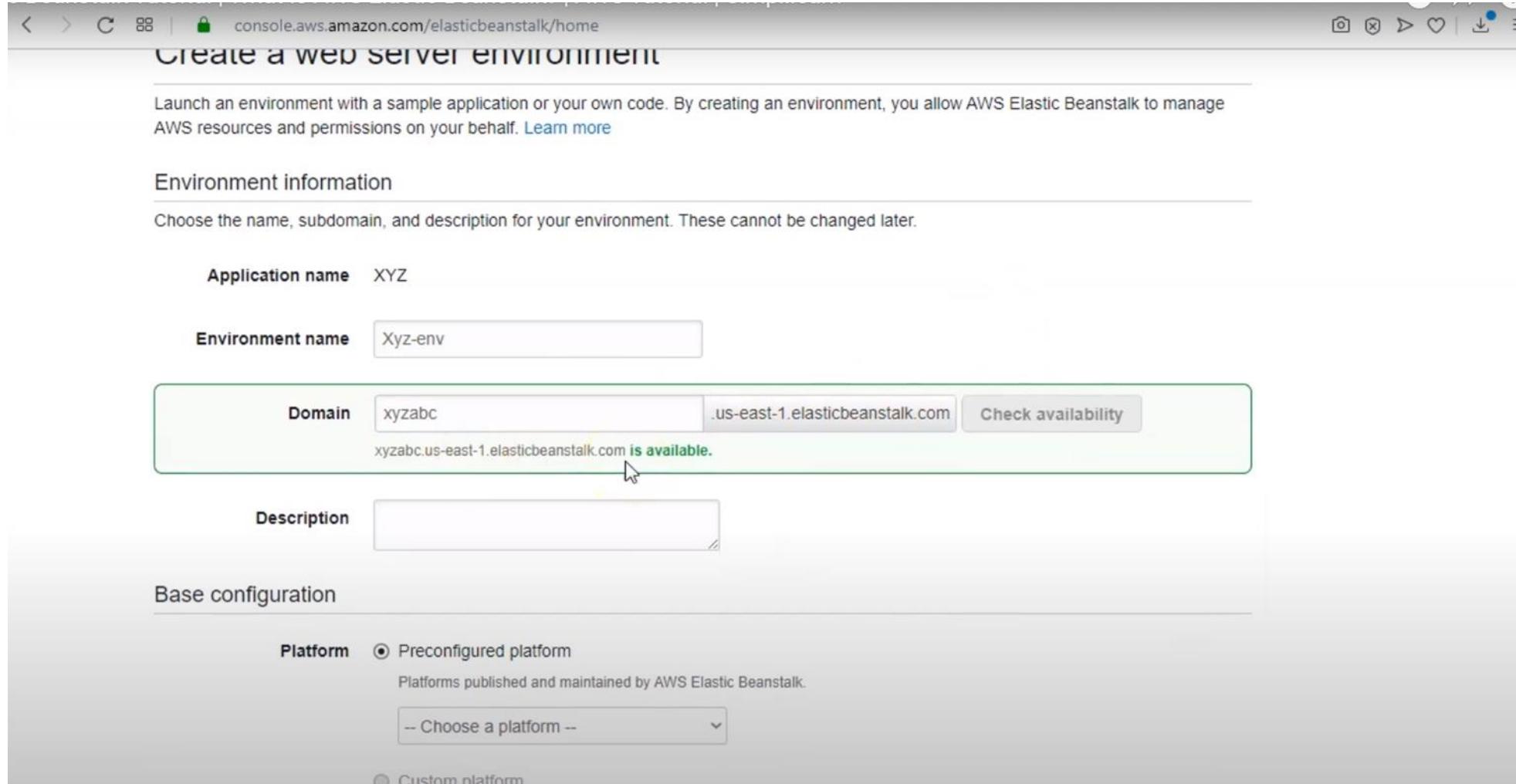
Step 4: Observer we don't have an environment for application, click on create one now.



The screenshot shows the AWS Elastic Beanstalk console interface. At the top, the URL is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, a bell icon, Sunshine Learning, N. Virginia, and Support. Below the navigation bar, there are tabs for 'Elastic Beanstalk' and 'XYZ'. On the right, there is a 'Create New Application' button. The main content area shows the 'XYZ' application under 'All Applications'. On the left, there is a sidebar with three options: 'Environments' (which is selected and highlighted in orange), 'Application versions', and 'Saved configurations'. The main content area displays a message: 'No environments currently exist for this application. [Create one now.](#)' A yellow circle with a cursor icon is drawn around the 'Create one now.' link.

PaaS Demo : AWS Elastic Beanstalk

Step 5: Enter the domain name for the application



console.aws.amazon.com/elasticbeanstalk/home

Create a web server environment

Launch an environment with a sample application or your own code. By creating an environment, you allow AWS Elastic Beanstalk to manage AWS resources and permissions on your behalf. [Learn more](#)

Environment information

Choose the name, subdomain, and description for your environment. These cannot be changed later.

Application name XYZ

Environment name Xyz-env

Domain xyzabc.us-east-1.elasticbeanstalk.com [Check availability](#)

xyzabc.us-east-1.elasticbeanstalk.com **is available.**

Description

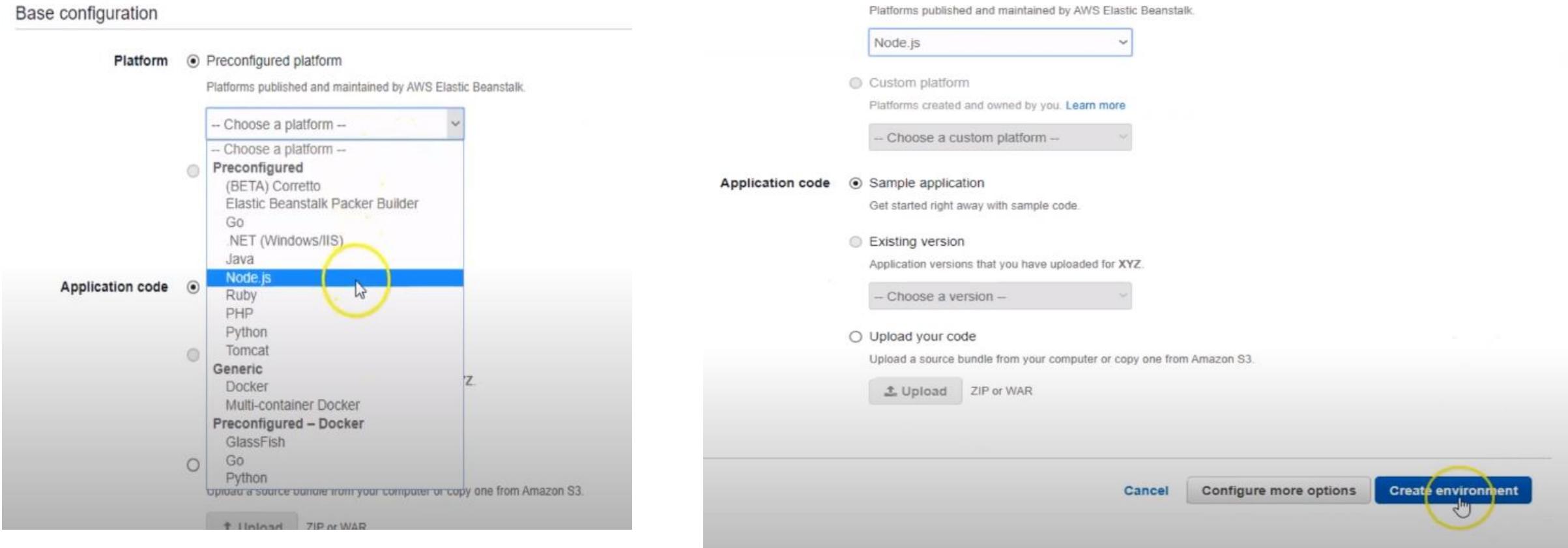
Base configuration

Platform Preconfigured platform
Platforms published and maintained by AWS Elastic Beanstalk.

Custom platform

PaaS Demo : AWS Elastic Beanstalk

Step 6: Enter the platform(Node js in this case) and click on create environment



The screenshot shows the 'Create New Environment' wizard in the AWS Elastic Beanstalk console. The first step, 'Base configuration', is displayed.

Platform: Preconfigured platform (selected). A dropdown menu shows 'Node.js' highlighted with a yellow circle and a cursor icon.

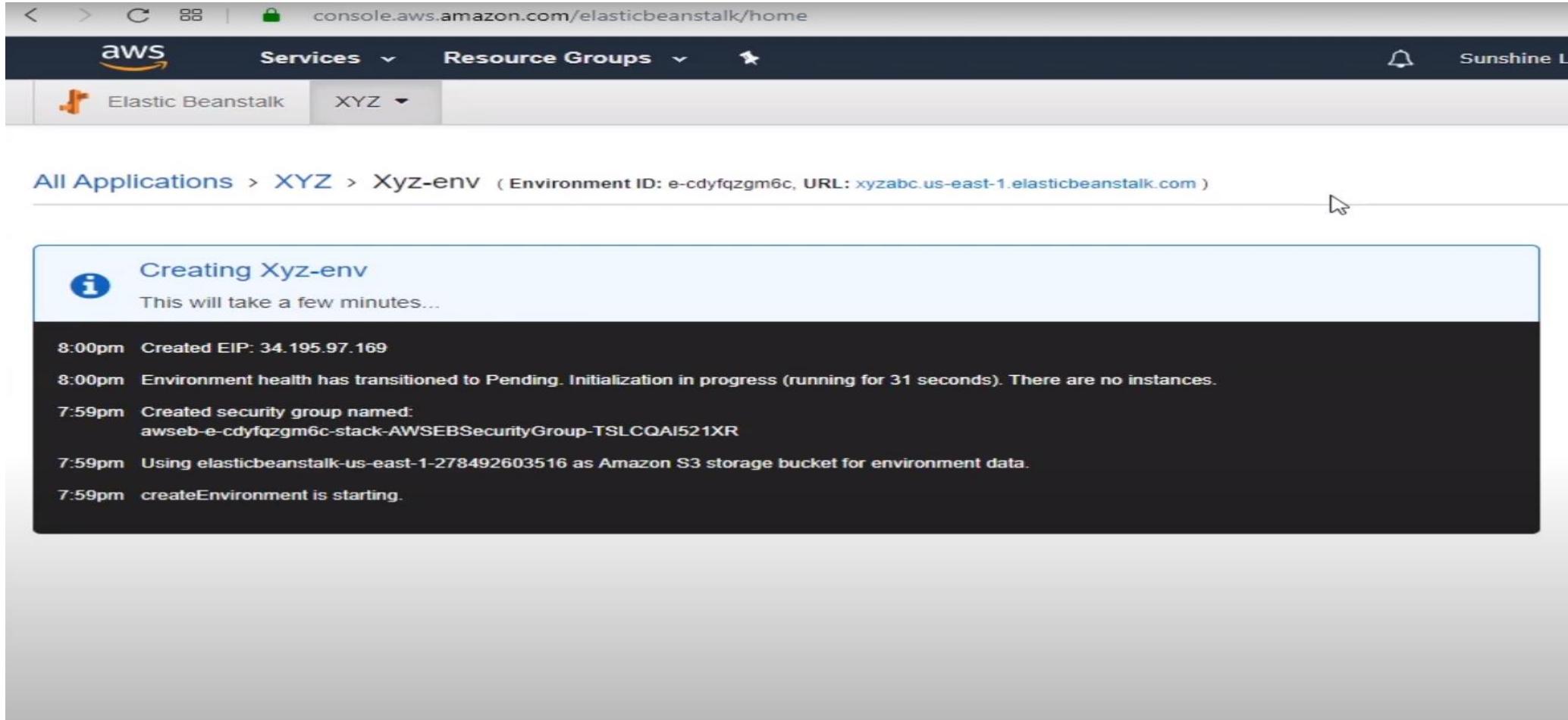
Application code: Node.js (selected). Other options include Ruby, PHP, Python, Tomcat, Generic, Docker, Multi-container Docker, and Preconfigured – Docker. A note at the bottom says 'Upload a source bundle from your computer or copy one from Amazon S3.' Below the application code section are 'Upload' and 'ZIP or WAR' buttons.

Platform selection: A dropdown menu titled 'Platforms published and maintained by AWS Elastic Beanstalk' shows 'Node.js' selected. Other options include 'Custom platform' (selected), 'Sample application' (selected), 'Existing version', and 'Upload your code'. A note at the bottom says 'Platforms created and owned by you. Learn more'. Below the platform selection are 'Choose a custom platform' and 'Upload' buttons.

Create environment button: At the bottom right of the form, there are three buttons: 'Cancel', 'Configure more options', and a large blue 'Create environment' button, which is also circled in yellow.

PaaS Demo : AWS Elastic Beanstalk

Step 7: Elastic Beanstalk will create the environment and we can observe the logs on the dashboard.
Click on the application name(XYZ in this case).



The screenshot shows the AWS Elastic Beanstalk console interface. At the top, the URL is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, Resource Groups dropdown, and a Sunshine icon. Below the navigation bar, there are tabs for **Elastic Beanstalk** and **XYZ**. The main content area displays the following information:

All Applications > XYZ > Xyz-env (Environment ID: e-cdyfqzgm6c, URL: xyzabc.us-east-1.elasticbeanstalk.com)

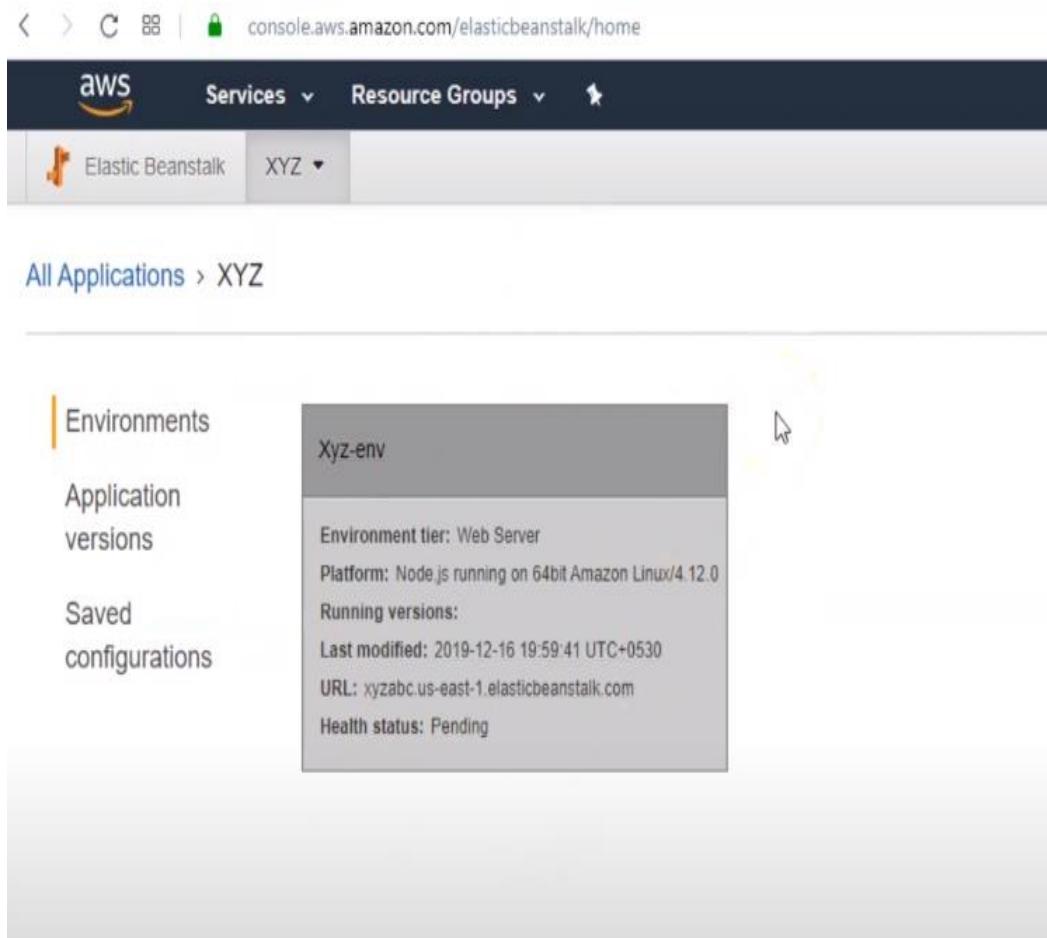
Creating Xyz-env
This will take a few minutes...

Logs (Timeline):

- 8:00pm Created EIP: 34.195.97.169
- 8:00pm Environment health has transitioned to Pending. Initialization in progress (running for 31 seconds). There are no instances.
- 7:59pm Created security group named:
awseb-e-cdyfqzgm6c-stack-AWSEBSecurityGroup-TSLCQAI521XR
- 7:59pm Using elasticbeanstalk-us-east-1-278492603516 as Amazon S3 storage bucket for environment data.
- 7:59pm createEnvironment is starting.

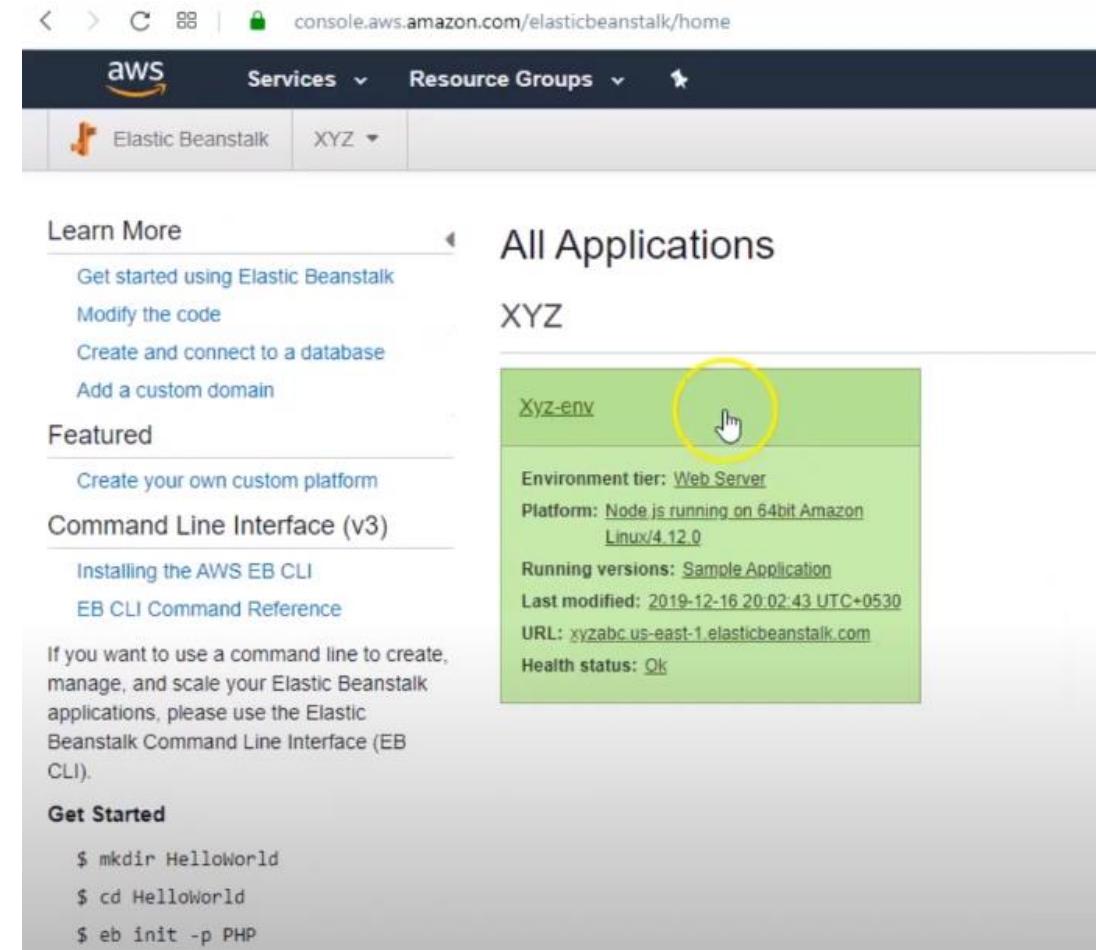
PaaS Demo : AWS Elastic Beanstalk

Step 8: Observe the colour of the box depicting the environment. Grey colour indicates the build is in progress. The green colour indicates the environment is up and running. Once the environment is ready click on it.



The screenshot shows the AWS Elastic Beanstalk console. The URL in the browser is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, and Resource Groups dropdown. Under the Services dropdown, 'Elastic Beanstalk' is selected. A dropdown menu shows 'XYZ' is currently selected. The main content area displays the 'All Applications > XYZ' section. On the left, there's a sidebar with 'Environments', 'Application versions', and 'Saved configurations'. The 'Environments' section lists 'Xyz-env' which is highlighted with a grey background. The details for 'Xyz-env' are as follows:

- Environment tier: Web Server
- Platform: Node.js running on 64bit Amazon Linux/4.12.0
- Running versions:
- Last modified: 2019-12-16 19:59:41 UTC+0530
- URL: xyzabc.us-east-1.elasticbeanstalk.com
- Health status: Pending

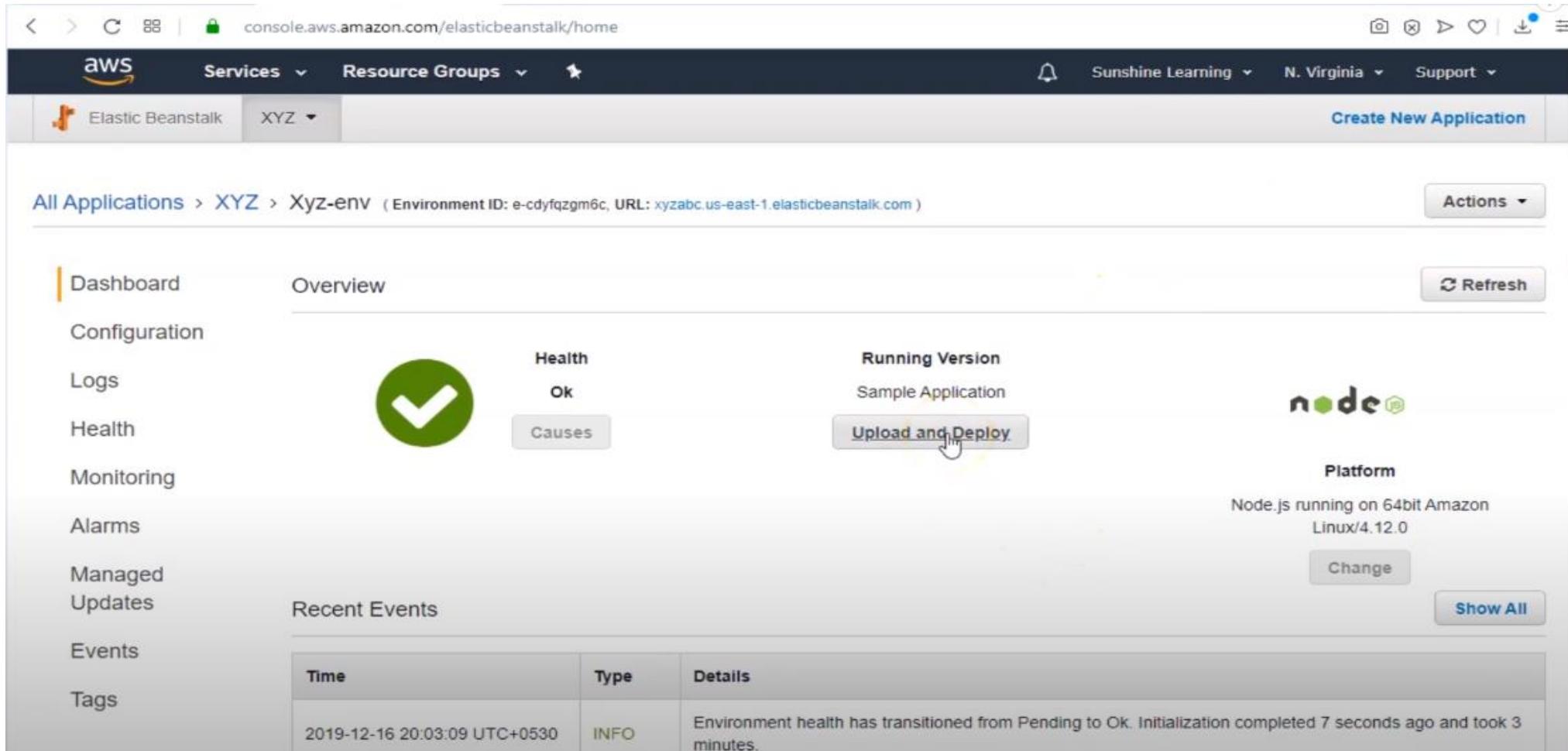


The screenshot shows the AWS Elastic Beanstalk console. The URL in the browser is `console.aws.amazon.com/elasticbeanstalk/home`. The navigation bar includes the AWS logo, Services dropdown, and Resource Groups dropdown. Under the Services dropdown, 'Elastic Beanstalk' is selected. A dropdown menu shows 'XYZ' is currently selected. The main content area displays the 'All Applications > XYZ' section. On the left, there's a sidebar with 'Learn More' links: 'Get started using Elastic Beanstalk', 'Modify the code', 'Create and connect to a database', and 'Add a custom domain'. Below that is a 'Featured' section with 'Create your own custom platform', 'Command Line Interface (v3)', 'Installing the AWS EB CLI', and 'EB CLI Command Reference'. The right side shows the 'XYZ' application details. The environment 'Xyz-env' is highlighted with a green background and has a yellow circle around its name. The details for 'Xyz-env' are as follows:

- Environment tier: Web Server
- Platform: Node.js running on 64bit Amazon Linux/4.12.0
- Running versions: Sample Application
- Last modified: 2019-12-16 20:02:43 UTC+0530
- URL: xyzabc.us-east-1.elasticbeanstalk.com
- Health status: Ok

PaaS Demo : AWS Elastic Beanstalk

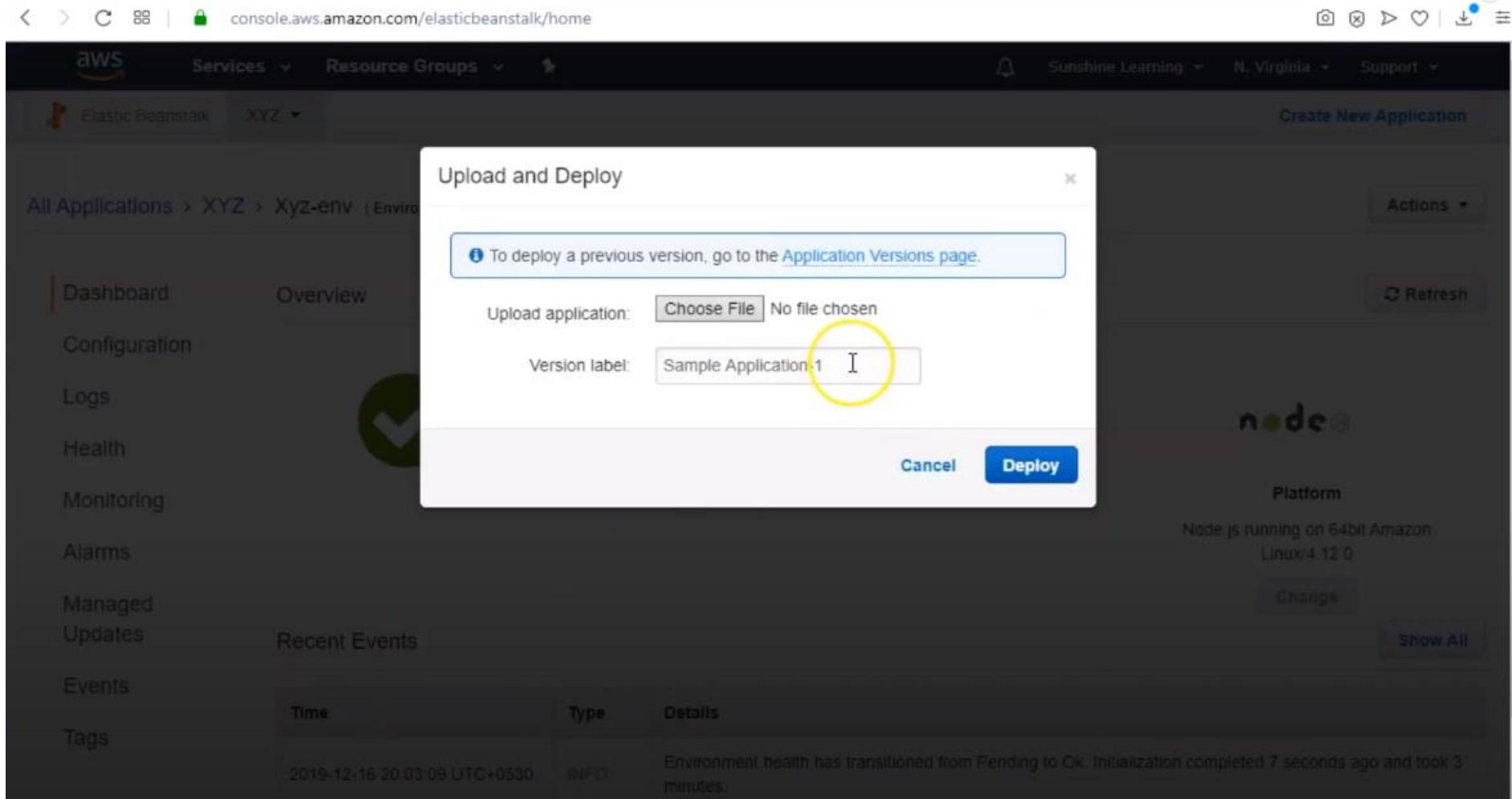
Step 9: We can see a dashboard giving an overview of the environment. Observe the health status and the selected platform. Click on Upload and Deploy



The screenshot shows the AWS Elastic Beanstalk Overview page for the 'XYZ' application environment 'Xyz-env'. The left sidebar lists navigation options: Dashboard (selected), Configuration, Logs, Health, Monitoring, Alarms, Managed Updates, Events, and Tags. The main content area displays the 'Overview' tab. It features a large green circle with a white checkmark, labeled 'Health' and 'Ok'. Below it is a 'Causes' link. To the right, there's a section for the 'Running Version' labeled 'Sample Application' with a 'node.js' logo. A large, prominent blue button labeled 'Upload and Deploy' is centered below the health status. To the right of the application details, a 'Platform' section indicates 'Node.js running on 64bit Amazon Linux/4.12.0' with a 'Change' link and a 'Show All' link. At the bottom, a 'Recent Events' table shows one entry: '2019-12-16 20:03:09 UTC+0530' (INFO) - 'Environment health has transitioned from Pending to Ok. Initialization completed 7 seconds ago and took 3 minutes.'

PaaS Demo : AWS Elastic Beanstalk

Step 10: Upload your application file and deploy.



The screenshot shows the AWS Elastic Beanstalk console. In the center, a modal window titled "Upload and Deploy" is open. It contains a message: "To deploy a previous version, go to the [Application Versions page](#)". Below this, there is a "Upload application:" section with a "Choose File" button and a message "No file chosen". Underneath is a "Version label:" section with an input field containing "Sample Application 1", which is circled in yellow. At the bottom of the modal are "Cancel" and "Deploy" buttons. The background shows the main Elastic Beanstalk dashboard for the "XYZ" application environment, featuring tabs for Dashboard, Overview, Configuration, Logs, Health, Monitoring, Alarms, and Managed Updates. The "Overview" tab is selected. On the right side of the dashboard, there is information about the platform: "node.js running on 64bit Amazon Linux/4.12.0".



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

COMMUNICATION USING MESSAGE QUEUES

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

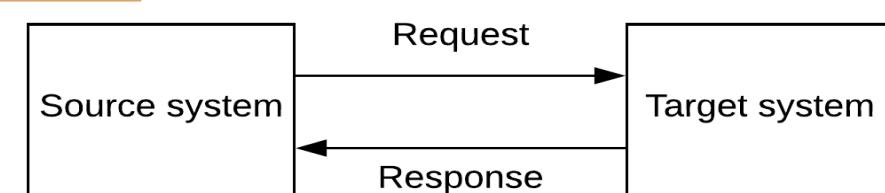
- In a distributed computing environment like the Cloud, where the hardware infrastructure typically configured as a cluster with different system architectures like a client-server or master-slave or say a P2P architecture, there exists a significant interactions between the components or machines involving data flows from one system to another which could be through messages or events.
- These data flows could happen using different protocols such as HTTP, AMQP (Advanced Message Queuing Protocol, or a binary protocol like TCP and with different data formats.
- There are also different styles or natures through which these communication may happen like **synchronous** or **asynchronous** or based on the **request** and **number of processors** of the service
- If the application is a monolith with multiple processes, simple intra-system IPC mechanism may work to communicate with processes which need to interact, but when built as a set of say microservices maybe running on different systems, communication mechanisms which can support Inter-service communication would be essential

CLOUD COMPUTING

Interaction styles

1. Synchronous (request-response) :

- Synchronous messaging means that the system which is sending the message expects an immediate response from the target system.
- Source system sends a message (request) to the target system and waits (blocks) until it receives a response from the target system.
- Target system may process the message within itself or send the message to another system and generate a response within a short time period.
- To make this communication successful and make sure source system does not waste its resources in case of a target system failure, there is a timeout configured at the source side so that it will stop waiting for the response if the timeout is elapsed without receiving a response from the target system.
- There are variants of this like with call-back and full-duplex .. which you could go through



Source system waits(blocks) for the response from target system

2. Asynchronous :

- The client doesn't block, and the response, if any, isn't necessarily sent immediately
- This supports high rates of data flow.
- The expectations of these type of messages are
 - Guaranteed delivery
 - Extensive processing
 - Correlation and time series analysis
 - Decoupling of source and target systems
- These could be of the types
 - Publish Subscribe (based on topics)
 - Message Queues
 - *Event based real time processing (not planned for discussion)*
 - *Batch Processing*
 - *Store and Forward*

Asynchronous messaging : Advantages

- **Reduced coupling:** The message sender does not need to know about the consumer.
- **Multiple subscribers:** Using a pub/sub model, multiple consumers can subscribe to receive events.
- **Failure isolation:**
 - If the consumer fails, the sender can still send messages.
 - The messages will be picked up when the consumer recovers.
 - Asynchronous messaging can handle intermittent downtime. Synchronous APIs, on the other hand, require the downstream service to be available or the operation fails.
- **Load leveling:** A queue can act as a buffer to level the workload, so that receivers can process messages at their own rate.

Asynchronous messaging : Disadvantages

Coupling with the messaging infrastructure: Using a particular messaging infrastructure may cause tight coupling with that infrastructure. It will be difficult to switch to another messaging infrastructure later.

- **Latency:** End-to-end latency for an operation may become high if the message queues fill up.
- **Complexity:** Handling asynchronous messaging is not a trivial task. For example, handling of duplicated messages, or correlating request and response messages using a separate response queue.
- **Throughput:** If message queues are used, each message requires at least one queue operation and one dequeue operation. Moreover, queue semantics generally require some kind of locking inside the messaging infrastructure.

Interaction styles (Synchronous or Asynchronous)

	one-to-one	one-to-many
Synchronous	Request/response	—
Asynchronous	Asynchronous request/response One-way notifications	Publish/subscribe Publish/async responses

One-to-one interaction : Each client request is processed by exactly one service

The following are the different types of one-to-one interactions

1. ***Synchronous Request/response*** — A service client makes a request to a service and waits for a response. The client expects the response to arrive in a timely fashion. It might even block while waiting. This is an interaction style that generally results in services being tightly coupled.
2. ***Asynchronous Request/response*** — A service client sends a request to a service, which replies asynchronously. The client doesn't block while waiting, because the service might not send the response for a long time.
3. ***One-way notifications*** — A service client sends a request to a service, but no reply is expected or sent.

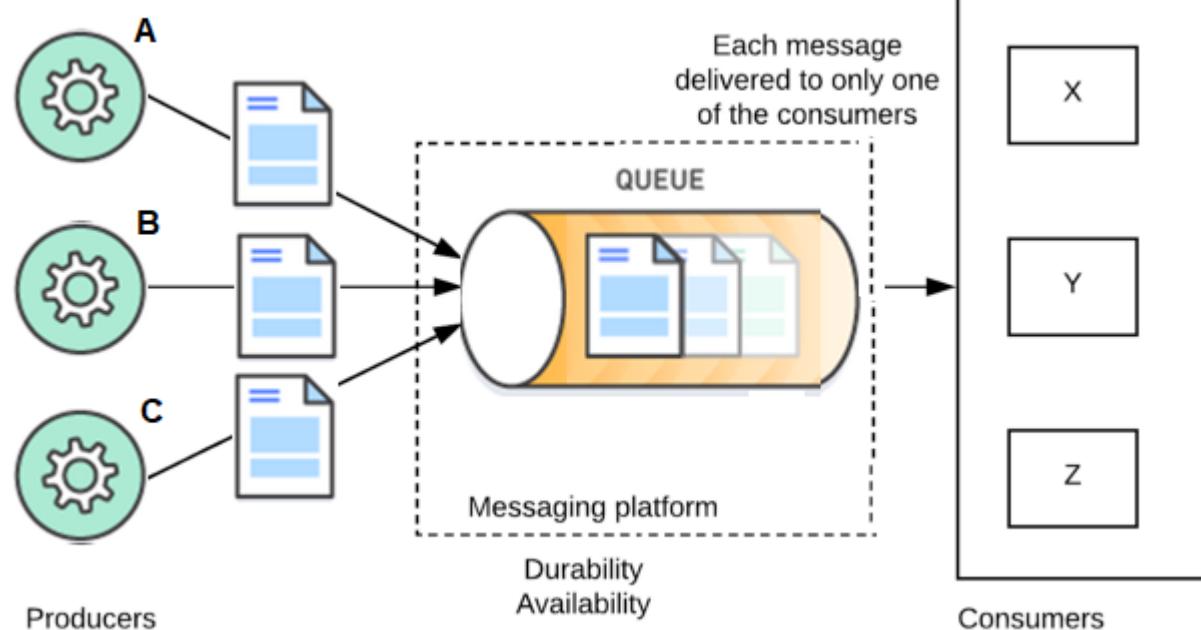
One-to-many interaction : Each request is processed by multiple services

The following are the different types of one-to-one interactions

1. ***Publish/subscribe***— A client publishes a notification message, which is consumed by zero or more interested services.
2. ***Publish/async responses***— A client publishes a request message and then waits for a certain amount of time for responses from interested services.

- A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures.
- Messages are stored on the queue until they are processed and deleted. Each message is processed only once, by a single consumer. Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.

Examples - Apache ActiveMQ, RabbitMQ



- A message queue provides a lightweight buffer which temporarily stores messages, and endpoints that allow software components to connect to the queue in order to send and receive messages.
- The messages are usually small, and can be things like requests, replies, error messages, or just plain information.
- To send a message, a component called a producer adds a message to the queue. The message is stored on the queue until another component called a consumer retrieves the message and does something with it.
- Many producers and consumers can use the queue, but each message is processed only once, by a single consumer. For this reason, this messaging pattern is often called one-to-one, or point-to-point, communications.
- When a message needs to be processed by more than one consumer, message queues can be combined with Pub/Sub messaging in a fanout design pattern.

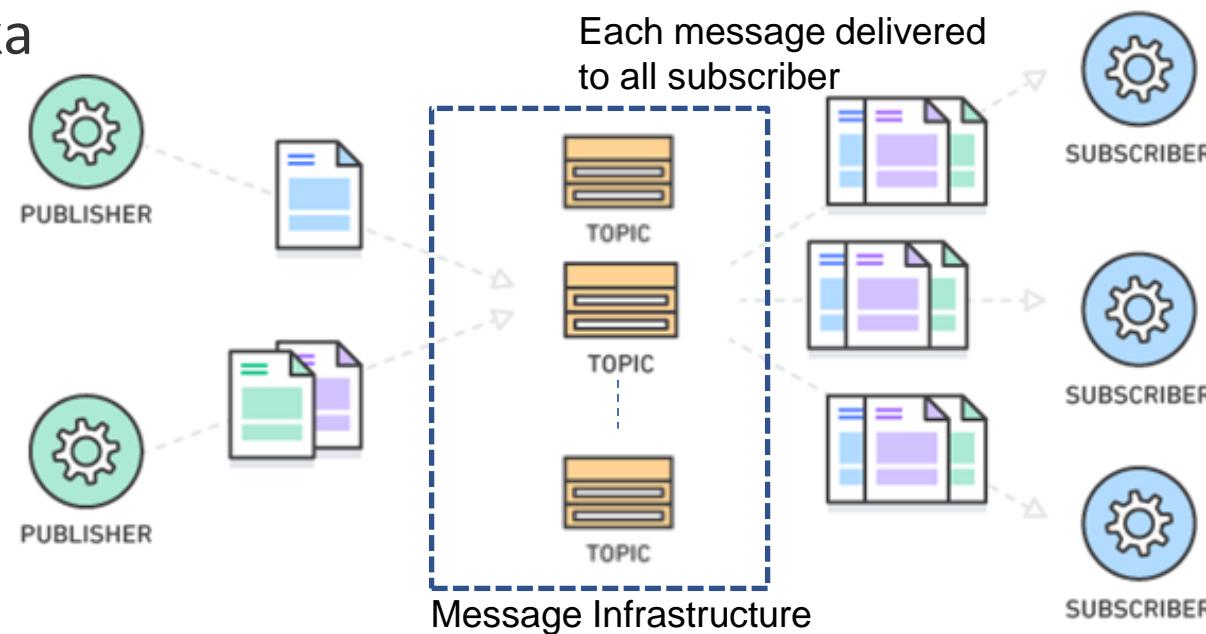
Asynchronous Communication : Publish-Subscribe

Publish/subscribe messaging is another asynchronous service-to-service communication mechanism

In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic.

Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability.

Eg. Apache Kafka



Four core concepts make up the pub/sub model:

- 1. Topic** – An intermediary channel that maintains a list of subscribers to relay messages to that are received from publishers

There can be different topic channels & different subscribers can look at their interested topics

- 1. Message** – Serialized messages sent to a topic by a publisher, has no knowledge of the subscribers
- 2. Publisher** – The application that publishes a message to a topic
- 3. Subscriber** – An application that registers itself with the desired topic in order to receive the appropriate messages

Advantages

1. Loosing coupling

Publishers are never aware of the existence of subscribers so that both systems can operate independently of each other. This methodology removes service dependencies that are present in traditional coupling.

For example, a client generally cannot send a message to a server if the server process is not running. With pub/sub, the client is no longer concerned whether or not processes are running on the server.

2. Scalability

Pub/sub messaging can scale to volumes beyond the capability of a single traditional data center. This level of scalability is primarily due to parallel operations, message caching, tree-based routing, and multiple other features built into the pub/sub model.

Advantages (Cont.)

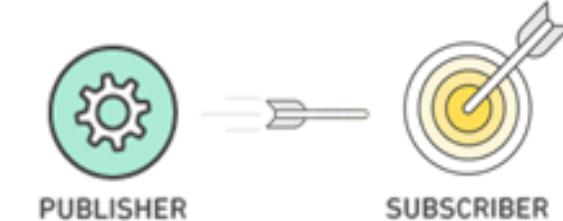
3. Eliminate Polling:

- Message topics allow instantaneous, push-based delivery, eliminating the need for message consumers to periodically check or “poll” for new information and updates.
- This promotes faster response time and reduces the delivery latency that can be particularly problematic in systems where delays cannot be tolerated.



4. Dynamic Targeting

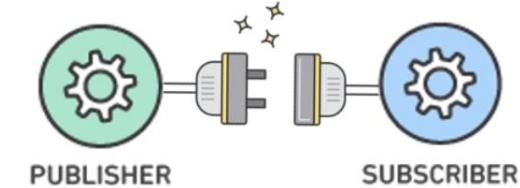
- Instead of maintaining a roster of peers that an application can send messages to, a publisher will simply post messages to a topic.
- Then, any interested party will subscribe its endpoint to the topic, and start receiving these messages. Subscribers can change, upgrade, multiply or disappear and the system dynamically adjusts.



Advantages (Cont.)

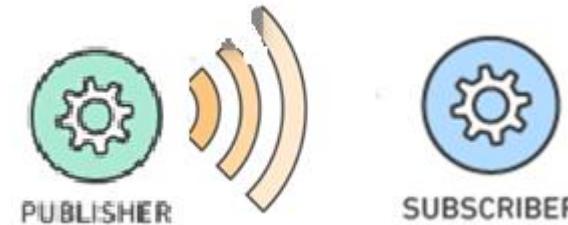
5. Decouple and Scale Independently:

- Publishers and subscribers are decoupled and work independently from each other, which allows you to develop and scale them independently.
- Adding or changing functionality won't send ripple effects across the system, because Pub/Sub allows you to flex how everything uses everything else.



6. Simplify Communication

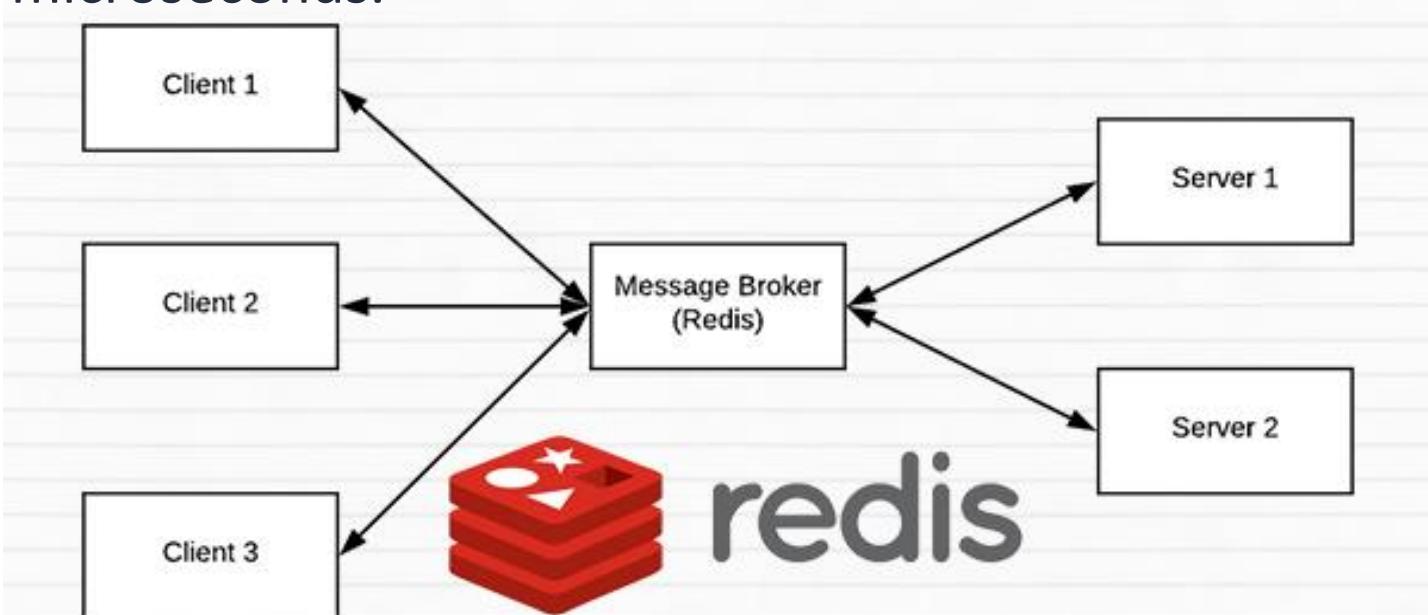
- Communications and integration code is some of the hardest code to write. The Publish Subscribe model reduces complexity by removing all the point-to-point connections with a single connection to a message topic, which will manage subscriptions to decide what messages should be delivered to which endpoints. Fewer callbacks results in looser coupling and code that's easier to maintain and extend.



Redis, which stands for **Remote Dictionary Server**, is a fast, open-source, in-memory key-value data store for use as a database, cache, **message broker**, and **queue**.

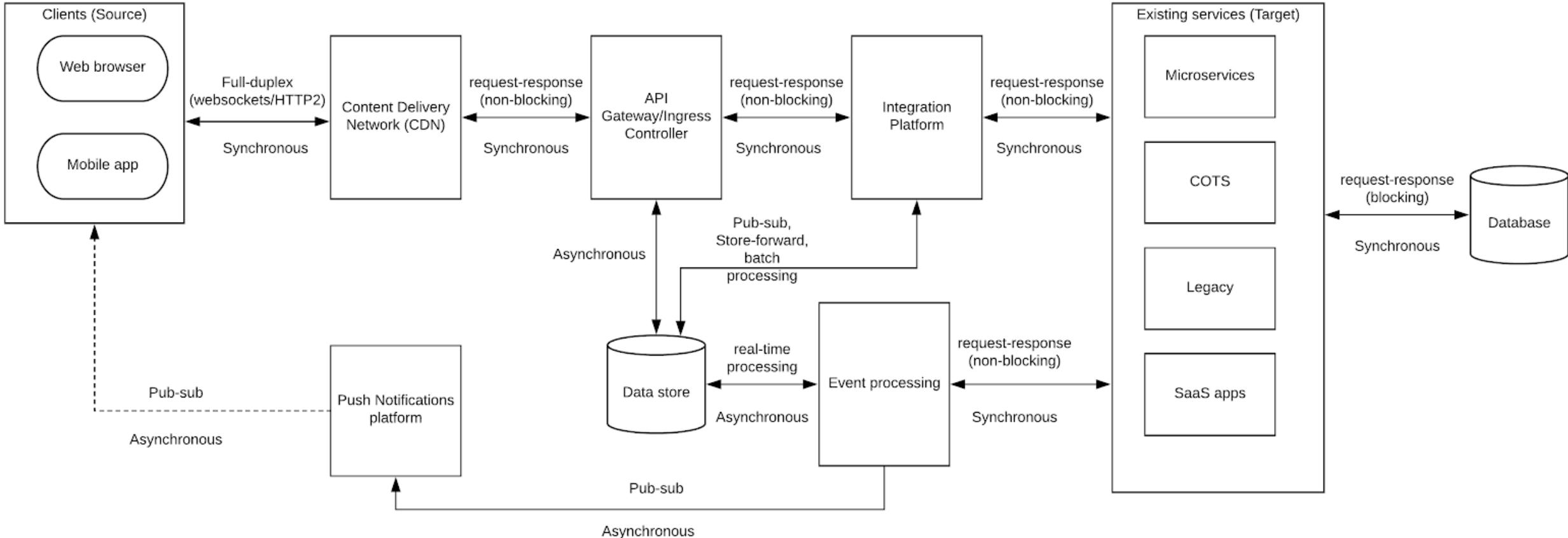
Why use Redis?

- All Redis data resides in-memory, in contrast to databases that store data on disk or SSDs. By eliminating the need to access disks, in-memory data stores such as Redis avoid seek time delays and can access data in microseconds.



CLOUD COMPUTING

A Distributed System with Messaging at Focus





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

SaaS Programming Model

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

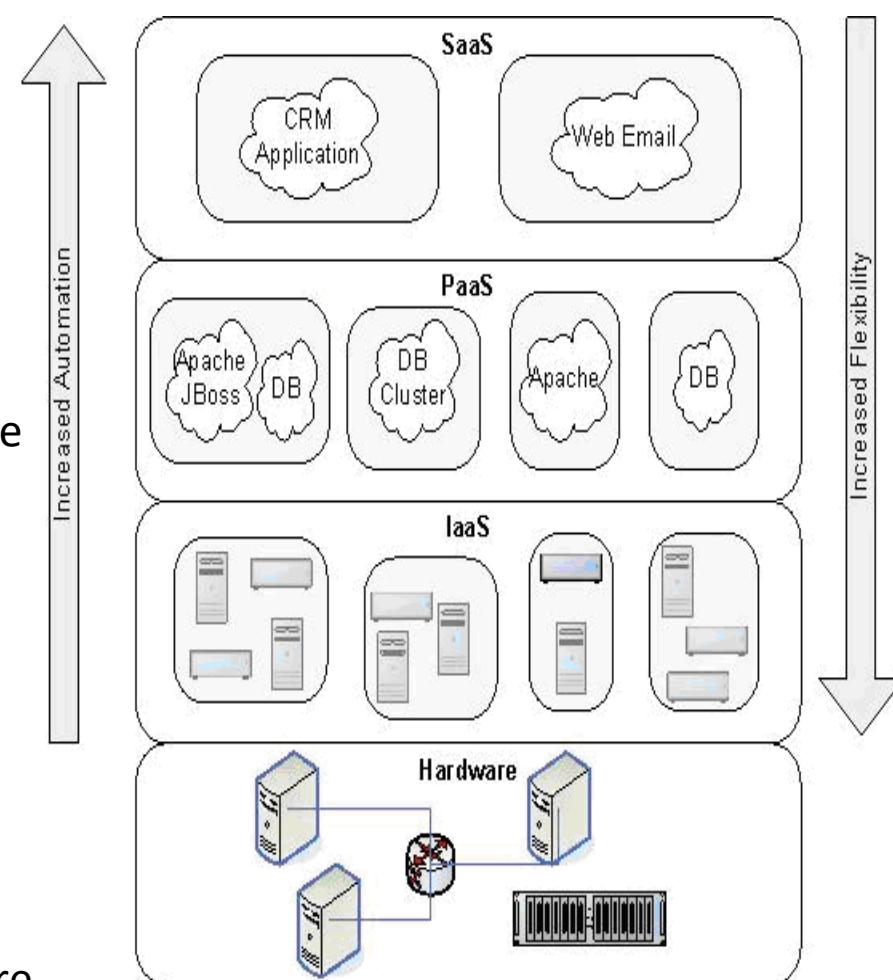
Software as a Service (Recap)

What is SaaS?

Software as a service (or SaaS) is a way of delivering applications as a service over the Internet. Instead of installing and maintaining software, you simply access it via the Internet from any thin client, freeing yourself from complex software and hardware management.

SaaS functioning

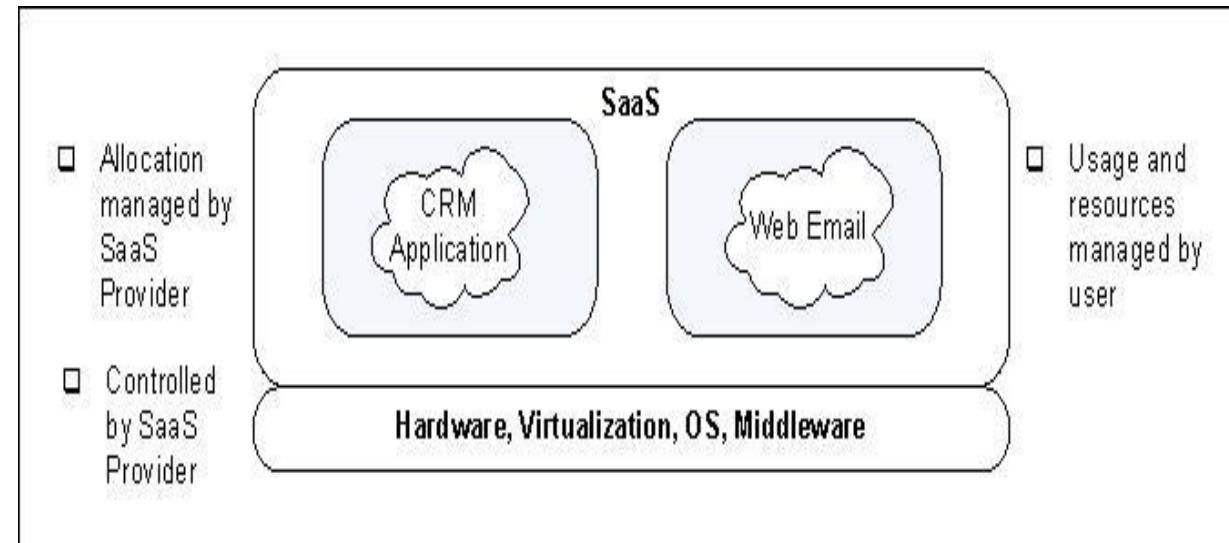
- Typically, a cloud service provider (like AWS, Azure, or IBM Cloud) manages the cloud environment on which the software is hosted.
- Users interact with the software through a web browser on their computer or mobile devices or they may use application programming interfaces (APIs) like REST to connect the software to other functions.
- SaaS applications take advantage of multitenant architecture to make use of pooled resources.
- Software updates, bug fixes, and other general app maintenance are taken care of by the SaaS provider.



Nearly every software that runs in your browser and is targeted towards the end user can be categorized as a Software as a Service product.

Popular SaaS Applications

1. Office apps - Google Docs, Sheets, Office 365
2. Email client - Gmail, Outlook
3. File Storage - DropBox, OneDrive
4. Social Media - Facebook, Snapchat
5. Customer Relationship Management - Salesforce
6. Customer support - Zendesk



1. Multi-tenant Architecture

- A multi-tenant architecture, in which all users and applications share a single, common infrastructure and code base that is centrally maintained.
- SaaS providers can make upgrades more often, with less customer risk and much lower adoption cost.
- The nature of SaaS makes it easier for providers to roll out new versions of software and features to their customers and reduce time previously spent on maintaining numerous versions of outdated code.

2. Easy Customization

- The ability for each user to easily customize applications to fit their business processes without affecting the common infrastructure.
- These customizations are unique to each company or user and are always preserved through upgrades. That means. Most SaaS applications are preconfigured plug-and-play products where the SaaS provider manages everything behind the app, including:

Lower up-front costs

Eliminate the need for additional hardware and middleware.
Reduce installation and implementation costs.
Validate and correct errors before making updates to your master data.

Predictable ongoing costs

Eliminate unpredictable costs of managing, patching, and updating software and hardware.
Turn capital expenses into operational expenses.
Reduce risk with experts managing software and overseeing cloud security.

Rapid deployment

Get up and running in hours instead of months.
Turn on and use the latest innovations and updates.
Automated software patching.

On-demand scalability.

Scale instantly to meet growing data or transactional demands.
Reduce disruptions while maintaining service levels.

Subscription

- Pay only monthly rental

Tiered Pricing

- Basic, Moderate and Superuser

Usage based pricing

- Pay for how much you are using, #API calls made

Per (Active) user pricing

- Pay for #users (or active users)

Per Feature pricing

- Each feature is priced separately

Ad-based revenue (pay per click)

- Every time there is a click, you pay

1. Security

Data is stored in the cloud, so security may be an issue for some users. There are number of approaches taken by Service providers to address this, and it could be as secure or more than in-house deployment.

2. Latency issue

Since data and applications are stored in the cloud at a variable distance from the end-user, there is a possibility that there may be greater latency when interacting with the application compared to local deployment.

Therefore, the SaaS model may not be suitable for applications whose demand response time is in milliseconds.

3. Dependency on Internet

Without an internet connection, most SaaS applications are not usable.

4. Switching between SaaS vendors is difficult

Switching SaaS vendors involves the difficult and slow task of transferring the very large data files over the internet and then converting and importing them into another SaaS.

In a SaaS Architecture, typically you will see

- Front-end or GUI for SaaS application is a browser
- Business Logic runs on the backend cloud infrastructure
- Business Logic frequently implemented as *microservices*

Examples: Google Docs, Gmail, Salesforce.com

Example of creating a SaaS application : Consider building an application say for a Book Mart which lets you browse, search, select and order, pay and maybe request for books.

Activities for the same would be

1. This SaaS application will need an say web-browser based GUI for the application (needs to factor in how the GUI screen would looks like, what technology to use ..)
2. Identify the features which would need to built into the system (authentication, listing of books, search, shopping cart, payment, refresh inventory ..)
3. Identify the workflows, customization of order of display of books .. which could happen in Backend
4. Identify and list say the features which could be built as microservices as a RESTful API for this application (Login, List books, Search, Shopping cart, purchase ..)
5. Identify those non-functional requirements (like availability, scaling, multi-tenancy, security ...) needed

Application Architectures - Monolithic

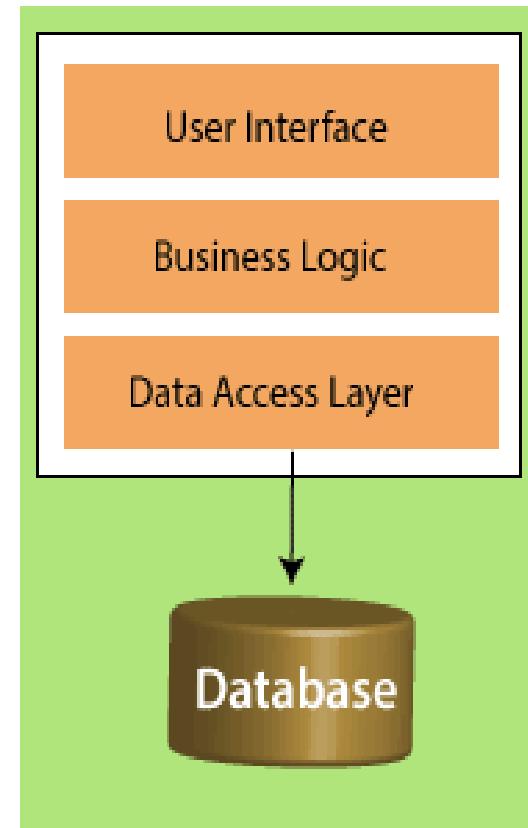
An application architecture describes the approach or patterns and techniques which are used to design and build an application.

Monolithic Architecture

- Monolith Architecture is the approach of building the application wherein all the features of the system are put on a single codebase and are typically with a single database
- In a monolith, all the components share the same resources and memory space
- It is considered to be a traditional way of building applications.
- A monolithic application is built as a single and indivisible unit. This could be visualized as in the figure with a client-side UI, Business logic and data access layer together with a database. If distributed you could have Server side application and a database access there.
- They can be characterized by being large, having long release cycles and having large teams

Typical Challenges

- Changes are not easily integrated (whole system may need to be rebuilt)
- Scalability challenges
- New Technology Adoption
- Difficult to adapt to newer practices, devices
- Reliability can be a question



Monolithic Architecture

Strengths of Monolithic Architecture

1. Development is quite simple.
2. Testing is very simple - Just launch the application and start end-to-end testing. We can also do test automation using Selenium without any difficulty.
3. Deploying the monolithic application is straightforward - Just copy the packaged application to the server.
4. Scalability is simple - We only need to have a new instance of the monolithic application and ask the load balancer to distribute load to the new instance as well. However, as the monolithic application grows in size, scalability becomes a serious issue.

Monolithic architecture worked successfully for many decades. In fact, many of the most successful and largest applications were initially developed and deployed as a monolith.

But serious issues like flexibility, reliability and scalability have led to the emergence of microservices architecture.

What are Microservices

Microservices are small services that work together but are independent, and are components or processes of an application. They have benefits of dynamic scalability, Reliability etc. as discussed later.

What is an Microservice Architecture

" The microservice is an architectural approach of developing a single application as a suite of small collaborating services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable by fully automated deployment machinery.

As an architectural framework, microservices are distributed and loosely coupled, so one team's changes won't break the entire app.

There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. "

- Martin Fowler

Benefits of Microservices Architecture

Benefits

1. Flexibility:

Microservices architecture is quite flexible. Different microservices can be developed in different technologies. Since a microservice is smaller, the code base is quite less, so it's not that difficult to upgrade the technology stack versions. Also, we can incrementally adopt a newer technology without much difficulty.

2. Reliability:

Microservices architecture can be very reliable. If one feature goes down, the entire application doesn't go down. We can fix the issue in the corresponding microservice and immediately deploy it. This makes the application to be more resilient.

3. Development speed:

Development is pretty fast in microservices architecture. Since the volume of code is much less for a microservice, it's not difficult for new team members to understand and modify the code. They become productive right from the start. Code quality is maintained well. The IDE is much faster. A microservice takes much less time to start up. All these factors considerably increase developers' productivity.

Benefits of Microservices Architecture (contd.)

4. Building complex applications:

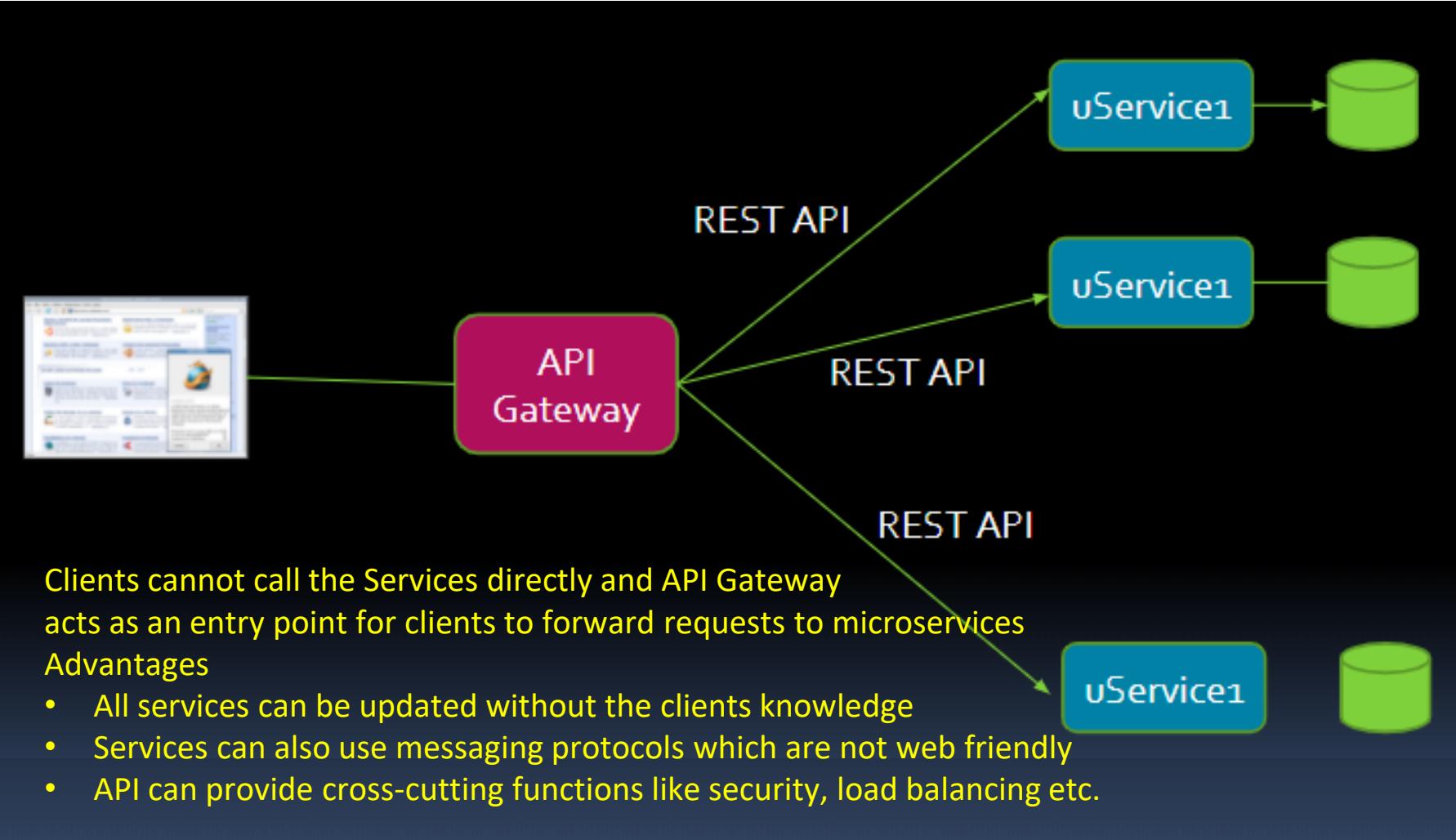
With microservice architecture, it's easy to build complex applications. If the features of the application are analyzed properly, we can break it down into independent components which can be deployed independently. Then, even the independent components can be further broken down into small independent tasks which can be deployed independently as a microservice. Deciding the boundaries of a microservice can be quite challenging. It's actually an evolutionary process, but once we decide on a microservice, it's easy to develop, as there are no limitation in technologies.

5. Dynamic Scalability:

Scalability is a major advantage in microservice architecture. Each microservice can be scaled individually. Since individual microservices are much smaller in size, caching becomes very effective.

6. Continuous deployment:

Continuous deployment becomes easier. In order to update one component, we have to redeploy only that particular microservice.



There are the following principles of Microservices:

- **Single Responsibility principle**

The single responsibility principle states that a class or a module in a program should have only one responsibility. Any microservice cannot serve more than one responsibility, at a time.

- **Modelled around business domain**

Microservice never restrict itself from accepting appropriate technology stack or database. The stack or database is most suitable for solving the business purpose.

- **Isolate Failure**

The large application can remain mostly unaffected by the failure of a single module. It is possible that a service can fail at any time. So, it is important to detect failure quickly, if possible, automatically restore failure.

- **Infrastructure automation**

The infrastructure automation is the process of scripting environments. With the help of scripting environment, we can apply the same configuration to a single node or thousands of nodes. It is also known as configuration management, scripted infrastructures, and system configuration management.

- **Deploy independently**

Microservices are platform agnostic. It means we can design and deploy them independently without affecting the other services.

Microservices Limitations

Building: Upfront time needs to be spent in terms of identifying dependencies between your services. Be aware that completing one build might trigger several other builds, due to those dependencies. You also need to consider the effects that microservices have on your data.

Testing: Integration testing, as well as end-to-end testing, can become more difficult as a failure in one part of the architecture could cause something a few hops away to fail, depending on how you've architected your services to support one another.

Versioning: When you update to new versions, keep in mind that you might break backward compatibility. You can build in conditional logic to handle this, but that gets unwieldy and nasty, fast. Alternatively, you could stand up multiple live versions for different clients, but that can be more complex in maintenance and management.

Deployment: Needs initial investment towards automation as the complexity of microservices becomes overwhelming for human deployment. Think about how you're going to roll services out and in what order.

Logging: With distributed systems, you need log management as the scale makes it hard to manage.

Monitoring: This may need to have a centralized view of the system to pinpoint sources of problems.

Debugging: Remote debugging through your local integrated development environment (IDE) isn't an option and it won't work across dozens or hundreds of services. Its difficult with no single answer to how to debug.

Connectivity: Consider service discovery, whether centralized or integrated

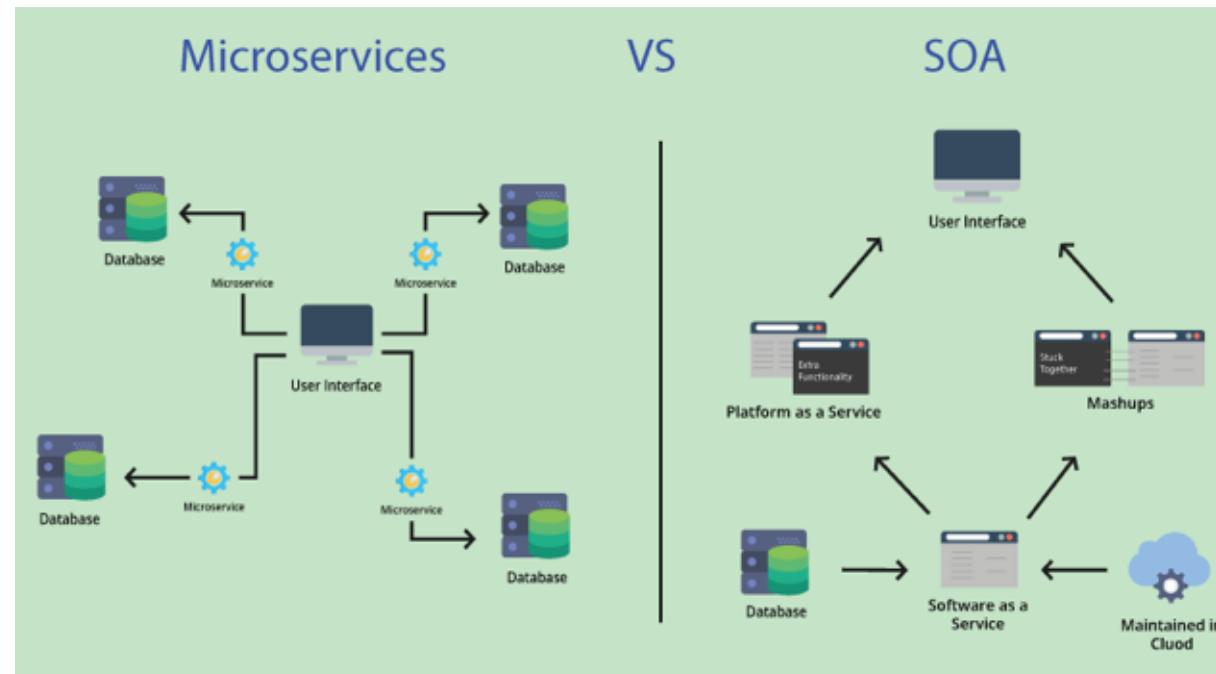
Microservices Limitations (others)

- Quick setup needed: You cannot spend a month setting up each microservice. You should be able to create microservices quickly.
- Automation: Because there are a number of smaller components instead of a monolith, you need to automate everything - Builds, Deployment, Monitoring, etc.
- Visibility: You now have a number of smaller components to deploy and maintain, maybe 100 or maybe 1000 components. You should be able to monitor and identify problems automatically. You need great visibility around all the components.
- Configuration Management: You need to maintain configurations for hundreds of components across environments. You would need a Configuration Management solution

What is SOA?

Service-oriented architecture was largely created as a response to traditional, monolithic approaches to building applications. SOA breaks up the components required for applications into separate service modules that communicate with one another to meet specific business objectives.

Microservice architecture is generally considered an evolution of SOA as its services are more fine-grained, and function independently of each other

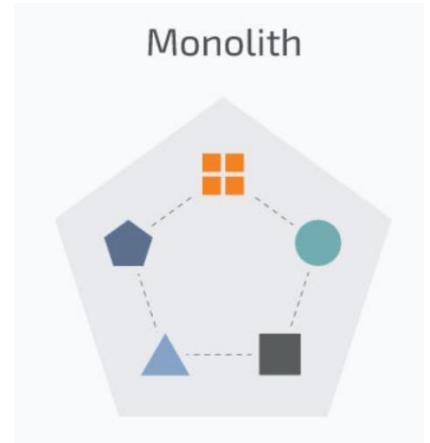


Comparison with SOA

SOA	Microservice Architecture
Follows “ share-as-much-as-possible ” architecture approach	Follows “ share-as-little-as-possible ” architecture approach
Importance is on business functionality reuse	Importance is on the concept of “ bounded context ” or Single Responsibility
They have common governance and standards	They focus on people, collaboration and freedom of other options
Uses Enterprise Service bus (ESB) for communication	Simple messaging systems
They support multiple message protocols	They use lightweight protocols such as HTTP/REST etc.
Multi-threaded with more overheads to handle I/O	Single-threaded usually with the use of Event Loop features for non-locking I/O handling
Maximizes application service reusability	Focuses on decoupling
Traditional Relational Databases are more often used	Modern Relational Databases are more often used
A systematic change requires modifying the monolith	A systematic change is to create a new service
DevOps / Continuous Delivery is becoming popular, but not yet mainstream	Strong focus on DevOps / Continuous Delivery

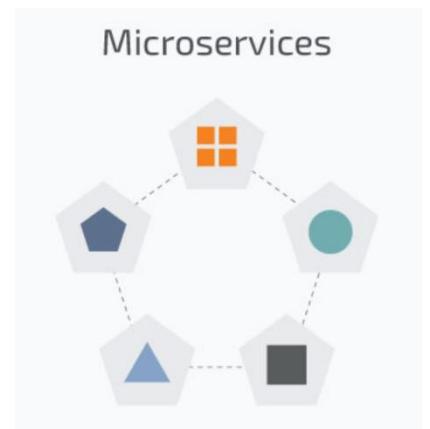
Monolithic Architecture

It is considered to be a traditional way of building applications. A monolithic application is built as a single unit. Usually, such a solution comprises a client-side user interface, a server side-application, and a database.



Microservices Architecture

While a monolithic application is a single unified unit, a microservices architecture breaks it down into a collection of smaller independent units. These units carry out every application process as a separate service. So all the services have their own logic and the database as well as perform the specific functions.





THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu



CLOUD COMPUTING

Challenges of migrating monolithic applications to microservices

Dr. H.L. Phalachandra

Department of Computer Science and Engineering

Acknowledgements:

Significant information in the slide deck presented through the Unit 1 of the course have been created by **Prof. K.S. Srinivas** and would like to acknowledge and thank him for the same. There have been some information which I might have leveraged from the content of **Dr. K.V. Subramaniam's** lecture contents too. I may have supplemented the same with contents from books and other sources from Internet and would like to sincerely thank, acknowledge and reiterate that the credit/rights for the same remain with the original authors/publishers only. These are intended for class room presentation only.

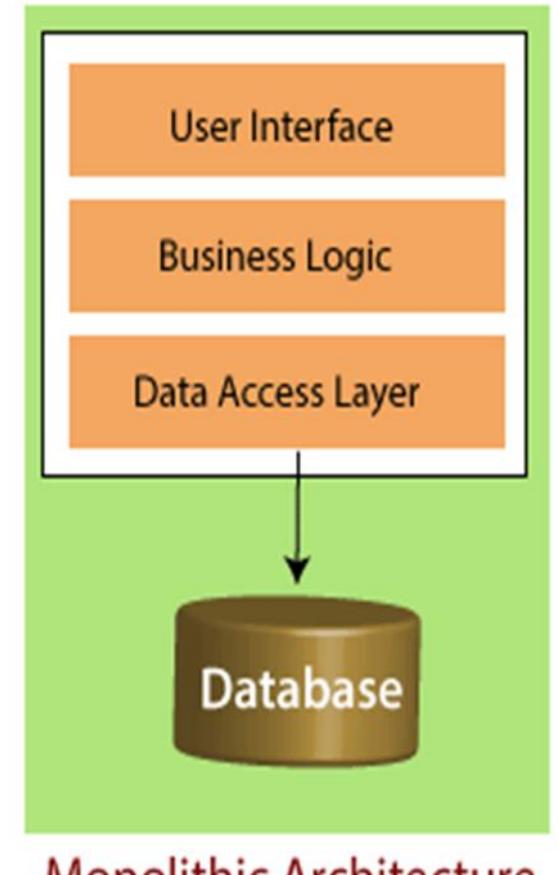
Need for Migration of applications to the Cloud

There are many benefits for applications to be on the cloud as seen earlier. Some of these could be

- **Scalability** : Applications which are based on the cloud, have a better ability to scale up or down based on your IT requirements and business plan.
- **Cost**: Reduce operational costs while improving IT processes. Applications in the cloud only pay for what they use, and have no need to maintain costly data centers when your important information is hosted in the cloud
- **Integration** : Cloud allows business to seamlessly connect systems together and improve efficiency with all your services. This alleviates risks like refresh of hardware etc.
- **Access** : all data is stored in the cloud, it can be accessed no matter what happens to your physical machinery.
- **Security** : In the cloud environment cloud infrastructure companies have built their cloud offering with privacy and security in mind making the applications to be secure. Most cloud providers also take care of issues like keeping unwanted traffic outside a specific scope from accessing the machines on which your business data and apps reside and ensuring automatic security updates are applied to their systems to keep from being vulnerable to the latest known security threats.

Need for Migration of applications to the Cloud

- Applications are designed and built with different approaches or patterns or techniques or have different Application Architectures
 - These could be built with Architectures which are not designed in a manner which can utilize all the benefits provided by the Cloud
- Eg. Monolithic Architectures discussed in the previous session which is built as a single and indivisible unit.
- This brings back the challenges like not easily integrated, scalability etc.



What is Migration

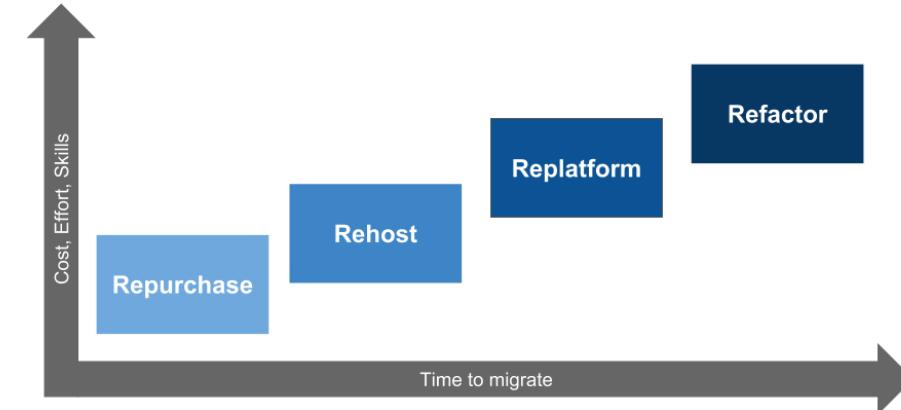
- Migrating to the cloud involves moving critical services and applications from on-premise and co-located hardware to Amazon Web Services, Google Cloud Platform, Microsoft Azure, or other cloud providers to take advantage of benefits of being on the cloud.
- These services allow you to manage IT infrastructure entirely remotely without the security risk, inconvenience, and cost of maintaining on-premise hardware.
- **Application migration** is the process of moving software applications from one computing environment to another. This can include migrating applications from one data center to another, such as from a public to a private cloud, or from a company's on-premises server to a cloud provider's environment.
- Organizations migrate applications to the cloud to take advantage of an improved cost structure, responsive scalability, and the ability to quickly update apps to meet changing demands etc.

CLOUD COMPUTING

Migration strategies

There are many ways to migrate applications to the cloud. The 6R framework proposes 6 different strategies for application migration depending on the amount of cloud nativity, business needs and level of effort. These 6Rs are:

- Re-host
- Re-platform
- Re-architecting
- Re-purchase
- *Retire*
- *Retain*



Of these 6 strategies, retire and retain are design strategies used post-migration. Retire means to remove features of the application that are no longer in use while retain means to only keep those features that are critical to the application. Re-purchase strategy is the method of moving perpetual licenses to a software-as-a-service model. For example, move from a customer relationship management (CRM) to Salesforce.com.

Ref - <https://cloudsoft.io/blog/a-practical-guide-to-understanding-the-6rs-for-migration-to-aws>

<https://www.heptabit.at/blog/6rs-application-migration-strategies#:~:text=6R%20framework%20proposes%20a%20separation,purchase%2C%20Retire%2C%20and%20Retain.&text=It%20is%20essential%20to%20prioritize,immediate%20value%20from%20the%20process.>

- Re-hosting: This is also known as the lift-and-shift migration strategy. Applications are migrated to the cloud without changes. It is most suitable for organizations that need to meet a business objective quickly. It is the fastest and easiest migration strategy. However, applications will not be able to fully reap the benefits of the cloud and have limited scalability.
- Re-platforming: This migration strategy is also known as lift-tinker-and-shift. The basic architecture of the application still remains the same but some optimizations may be made to achieve some of the benefits of the cloud. One of the most common alterations done to these applications is by managing database instances to a database-as-a-service platform like Amazon RDS.

Re-architecting

Re-architecting is one of the migration strategies that results in migrating non cloud native applications to the cloud, this leads to the highest return of investment and exploits complete cloud nativity. SOA or microservices are few of the architectures that can be considered while re-architecting applications. This strategy involves systematic and aggressive remodeling of existing architecture. It requires extensive modification of the application's codebase. There is no standard set of steps to be followed for re-architecting an application as it is entirely dependent on the application's architecture, logical components and functionalities.

The different application architectures are -

monolithic, service oriented architecture(SOA), microservice.

Therefore in re-architecting we can convert the existing architecture of the application to another.

Example: 1) monolithic to SOA

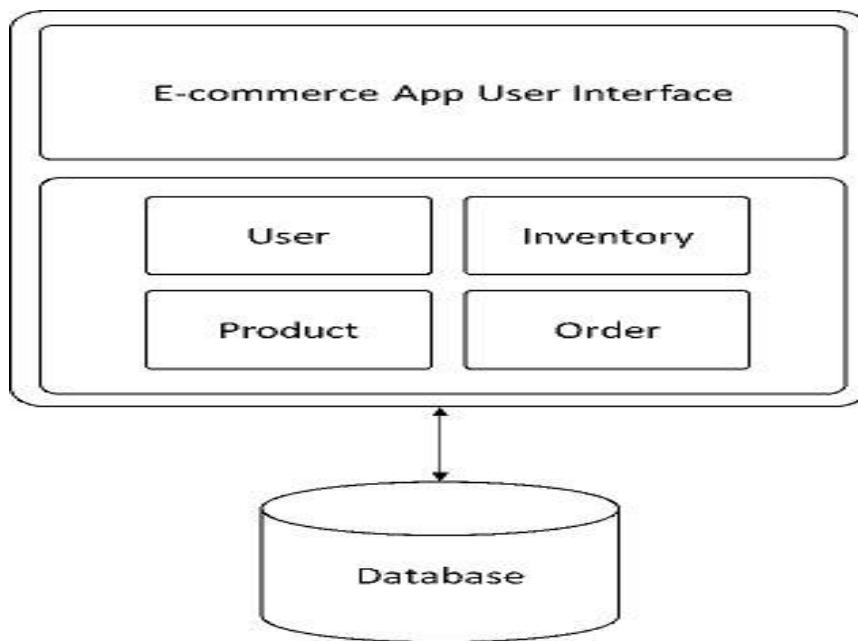
- 2) monolithic to microservice
- 3) SOA to microservice

Microservice architecture reaps all the features of cloud. Hence can be called cloud-native (built for cloud).

Migration is not easy

Microservices architecture promises to solve the shortcomings of monolithic applications, so many enterprises are interested in migrating their applications to be microservices. This migration is a worthwhile journey, but not an easy one.

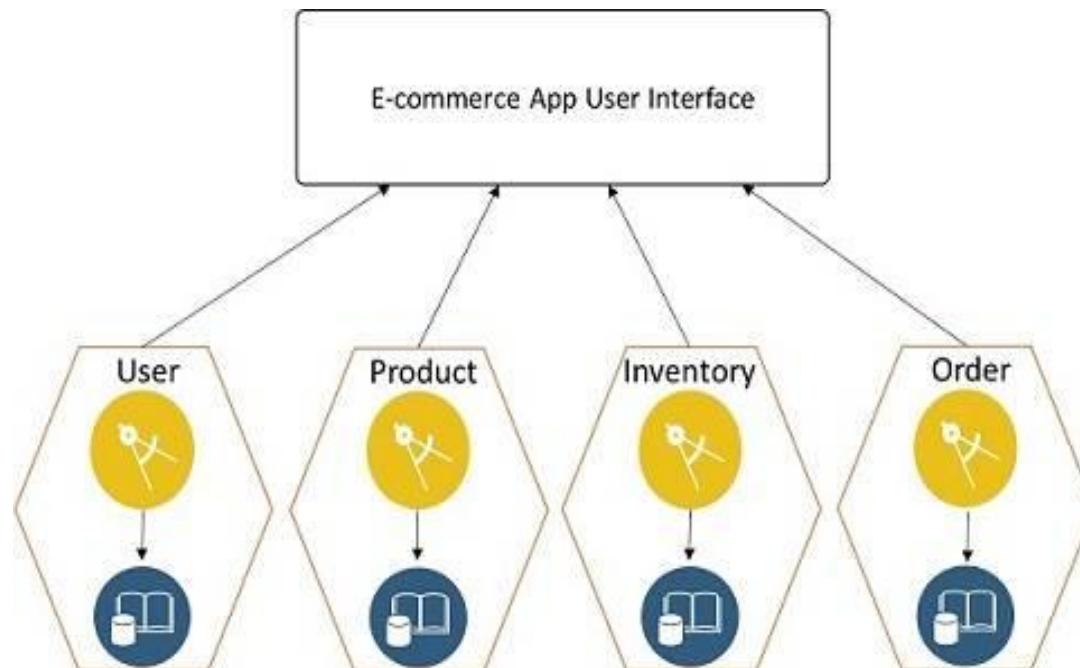
Most enterprises have a large application in place for their business use cases. Take the example of an e-commerce application used by an online retailer. The diagram below shows its monolithic architecture, where all services interact with a single database.



Monolithic architecture seems like a good idea to quickly get the application started, it becomes a problem over time. As the business and user base grows, customers expect newer user experiences, and integration requirements increase, the monolithic approach becomes a bottleneck to growth.

In contrast, Figure below shows how the same e-commerce application can be broken down into a microservices architecture.

Each service has its own database.



Migration is not easy

The idea seems simple but is sometimes difficult to perceive. The architecture team should consider the following challenges:

1. Service decomposition : Possible candidates that could represent a microservice.
2. Persistence : Monolithic database decomposition
3. Tackling transaction boundaries
4. Performance
5. Testing
6. Inter-service communication

In transitioning an existing monolith to a microservices, you would typically need to decompose the existing application into granular microservices. This is a challenging task as we need to identify possible independent components in a monolithic application.

Although breaking down a monolithic application can seem like an uphill task to start, with certain guidelines even this mammoth task is possible:

1. Stop adding more things to the monolithic app. Fix what is broken and accept only small changes.
2. Find out the so-called seams in your monolithic app. Identify components that are more loosely coupled than others and use them as a starting point.
3. Identify low hanging fruits, such as the components for which business units want to add more advanced features.

During this transitioning process, you would typically need to decompose the monolith to building more and more granular microservices to suit the business needs.

Once this is accomplished, there would be more moving parts in the application. As a result, this would lead to operational and infrastructural overheads, i.e., configuration management, security, provisioning, integration, deployment, monitoring, etc.

One of the ways to reduce these complexities is in using containerization. In using containerization, provisioning, configuration, and deployment of microservices would be simplified.

Persistence : Monolithic database decomposition

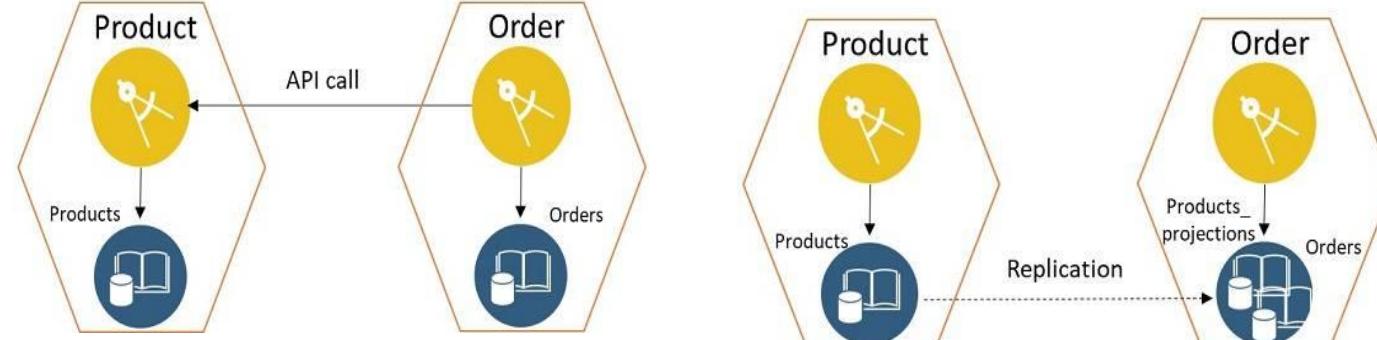
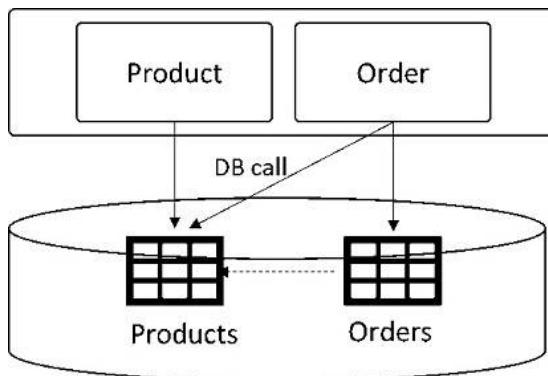
A monolithic application typically contains a single data store. In contrast, a microservices-based application contains multiple databases, typically one for each service. One of the key challenges in transitioning to microservices-based architecture is in understanding and handling decentralized data management.

Splitting the data model of a monolithic application to fit the autonomous data models of a microservices-based application is challenging. Breaking a monolith data model into separate autonomous data models that are local to each microservice is a daunting task, the following challenges need to be considered:

- A. Reference tables
- B. Shared mutable data
- C. Shared table

A. Reference tables :

- The most common type of pattern that is seen within monolithic applications is a reference table.
- In this pattern, the function or module accesses a table that belongs to another function or module.
- The joins between a module's own table and another table are used to retrieve the required info.
Using the previous example of an e-commerce application, the Order module uses a reference to the Products table to retrieve product information.
- In a microservices architecture, the Products table and Order module become separate microservices. Here are two options for you to consider to segregate the database objects :
 1. Data as an API
 2. Projection/Replication of data

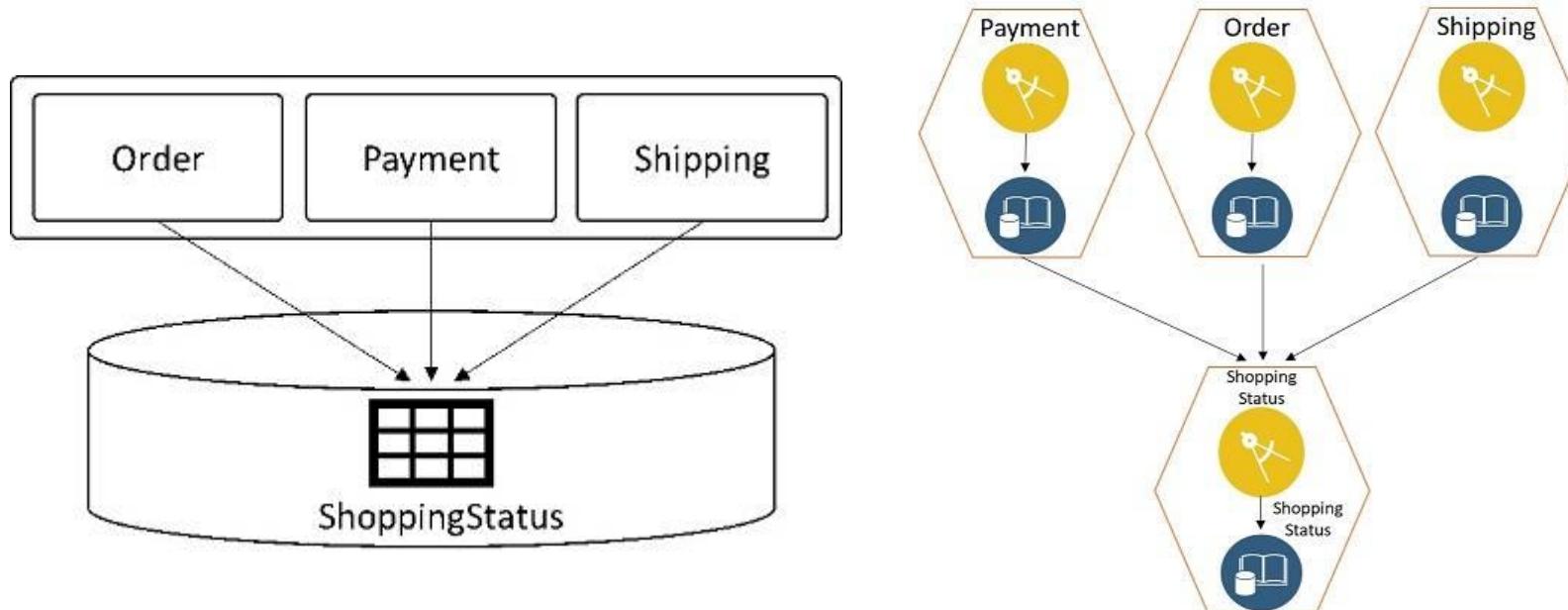


Persistence : Monolithic database decomposition

B. Shared mutable data :

In monolithic applications, there is a common pattern that is known as shared mutable state. This pattern represents certain domain realities that are modeled in a database and accessed by different functionalities or modules of the application. This is mainly done for convenience. For example, in the e-commerce application, the Order, Payment, and Shipping functionalities use the same ShoppingStatus table to maintain the customer's order status throughout the shopping journey.

While moving to microservice architecture they should eventually be modeled as a separate microservice.

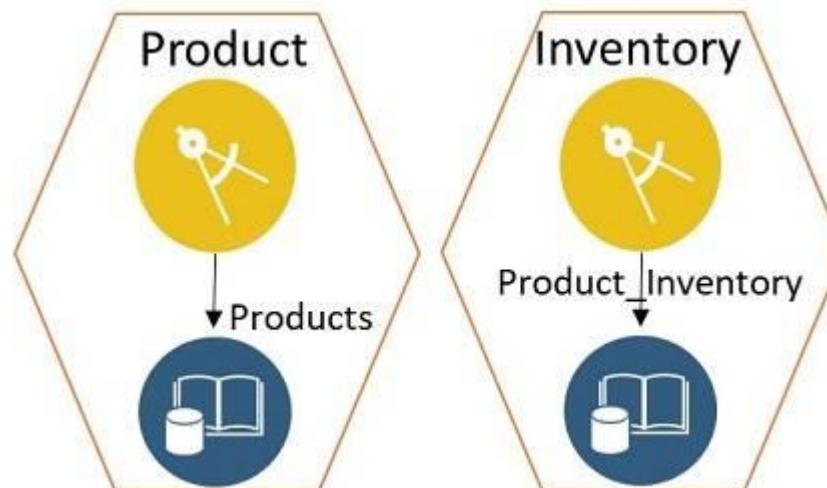
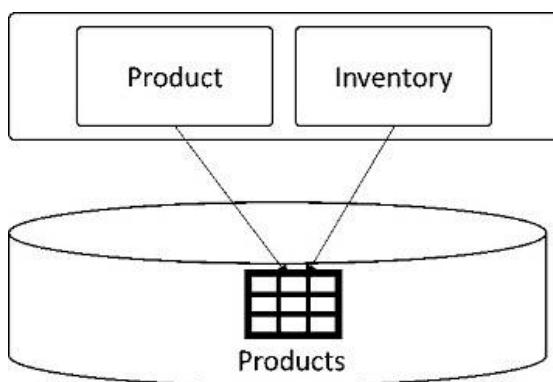


Persistence : Monolithic database decomposition

C. Shared table:

The shared table pattern is very similar to the shared mutable state pattern and is the result of erroneous domain modeling. In this scenario, a database table is modeled with attributes that are needed by two or more functionalities or modules. Again, this is done mostly for convenience reasons. To explain this, Figure below shows a scenario where the Products table is modeled to cater to the Product and Inventory functionalities.

While moving to a microservices architecture, Product and Inventory are modeled as separate services. The Products table is split into individual entities that belong to the specific bounded context of the separate Product and Inventory microservices, as shown in Figure.

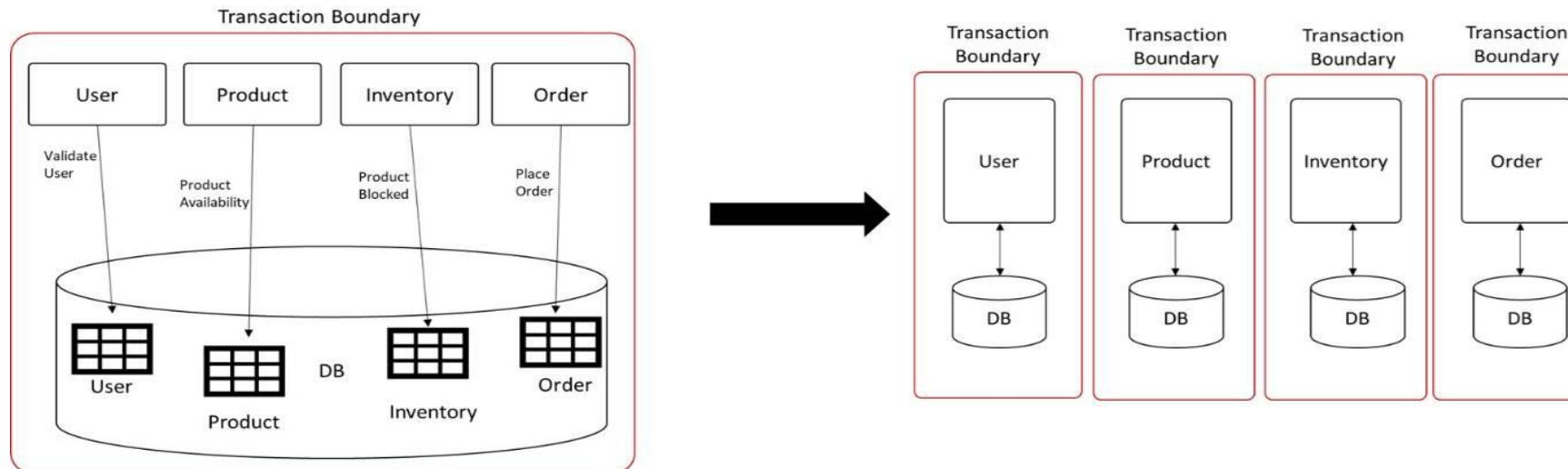


Tackling transaction boundaries

With a monolithic application that has a single database, you could guarantee the ACID (Atomicity, Consistency, Isolation, Durability) properties and ability to perform transactions by keeping locks on rows.

Things changed drastically when enterprises moved from service-oriented architecture to microservices architecture. They now must deal with database per service eventually leading to distributed transaction trouble. Following are the ideal ways to deal with distributed transactions :

- A. Two-phase commit (2PC)
- B. Compensating transactions (Saga transactions)

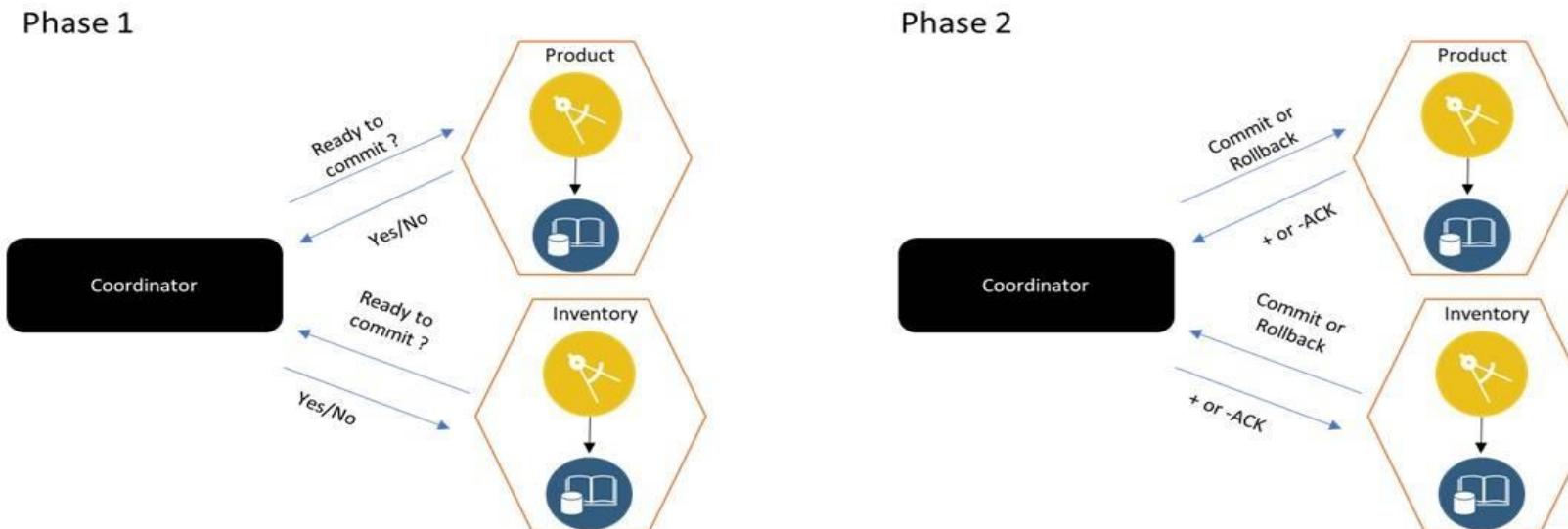


A. Two-phase commit (2PC):

In a two-phase commit, you have a controlling node that houses most of the logic, and a few participating nodes on which the actions are performed. It works in two phases:

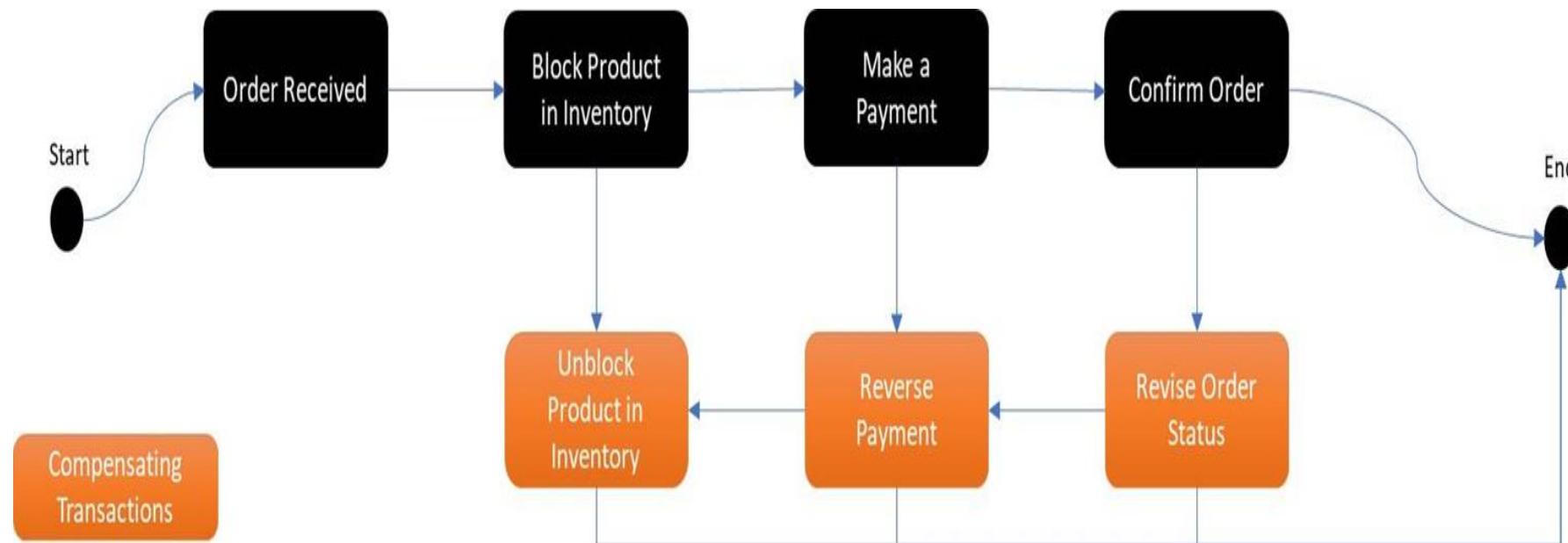
Prepare phase (Phase 1): The controlling node asks all of the participating nodes if they are ready to commit. The participating nodes respond with yes or no.

Commit phase (Phase 2): If all of the nodes replied in the affirmative, then the controlling node asks them to commit. Even if one node replies in the negative, the controlling node asks them to roll back.



B. Compensating transactions (Saga transactions):

A saga is a sequence of local transactions. Each service in a saga performs its own transaction and publishes an event. The other services listen to that event and perform the next local transaction. If one transaction fails for some reason, then the saga also executes compensating transactions to undo the impact of the preceding transactions.



The increase in resource usage may cause a microservices-based application to execute slower.

You can overcome this challenge by :

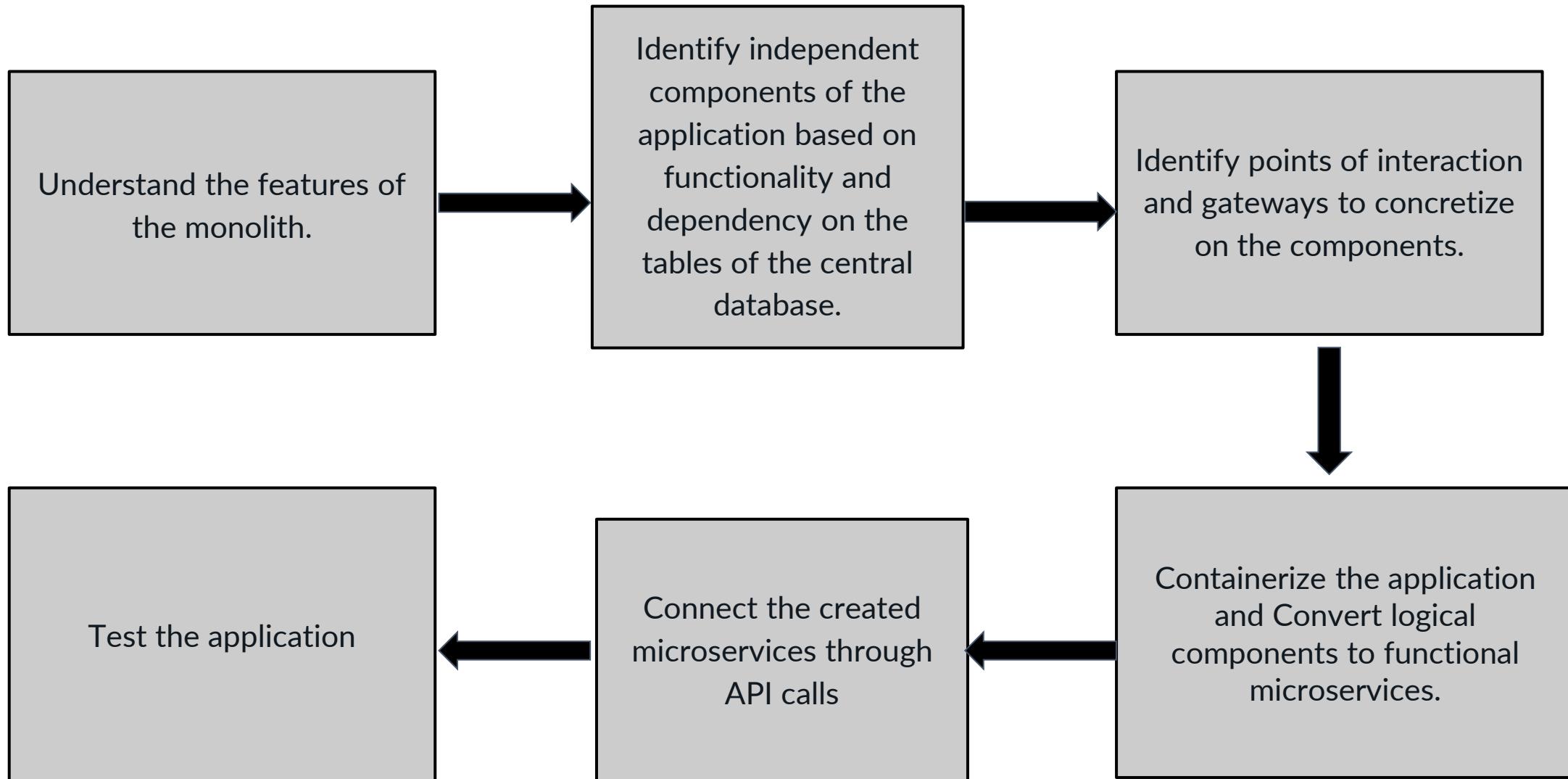
1. Introducing additional servers.
2. You can log performance data and detect the problems. You should take advantage of logging to store performance data in a repository so that you can analyze the data at a later point of time.
3. You can implement throttling, handle service timeouts, implement dedicated thread pools, implement circuit breakers and take advantage of asynchronous programming to boost the performance of microservice-based applications.

- Testing microservices can become challenging, particularly the integration tests. To write an effective integration test case, the QA engineer should have good knowledge of each of the services that are a part of the solution.
- Another reason why testing a microservices-based application is difficult is because microservices-based applications are generally asynchronous. The best approach to solving these testing challenges is adopting various testing methodologies and tools and leveraging continuous integration capabilities through automation and standard agile methodologies.

In a distributed system, the components should be able to communicate with each other and in microservices-architecture, it is no exception. In a monolithic application, the components invoke one another through method or function calls. In contrast, a microservices-based application is distributed in nature with each service running isolated from another service. Hence, it is imperative that services in a microservice-based application communicate using inter-process communication (also known as IPC) mechanisms. However, inter-service communication in a microservices-based application poses a lot of challenges.

Since microservices are distributed in nature, remote invocation of these services is a challenge and you should understand the necessary patterns to overcome the challenges involved. Since services in a microservice-based application communicate with one another using IPC mechanisms you should consider issues like, how the services will interact, how to handle failures, etc.

Generic steps for monolithic to microservices architecture



Some of the most common challenges associated with the conversion of monolithic applications to microservices / re-architecting are as follows:

- Identification of services based on the overall application's functionality
- Ensuring appropriate connection and communication between services
- Conversion of large and bulky desktop applications are challenging
- Identifying the boundary between re-architecting and rebuilding the entire application
- Requires more effort
- Operational complexity is also increased due to the increased demands on managing these services and monitoring them.
- Coordinating a large number of rapidly changing services necessitates automated continuous integration and continuous delivery.

Attributes which can be used for comparison of the migration approaches

These approaches can be compared using some of the attributes like

1. Performance
2. Scalability
3. Throughput
4. Accessibility
5. Loadtime
6. Cloud Nateness

Results from migration

However, it is important to note that not all applications are suited to all types of migration strategies. Based on the need of the business, and the amount of cloud nativeness required, the type of strategy to be used may differ. Consider the following table that shows the suitability of migration strategies based on application and business requirements

Suitability	Rehosting	Replatforming	Rearchitecting
Application suitability	<ul style="list-style-type: none">● Web compatible UI● MVC architecture● Flask, Symphony, .NET can be rehosted without complications● Applications not built with web frameworks but have a web-compatible UI (HTML, PHP) not ideal	<ul style="list-style-type: none">● Web-compatible UI and MVC architecture● Must allow easy connection to the database server● Traditional apps using Xampp-like servers are suitable	<ul style="list-style-type: none">● Monolithic architecture, but are web-compatible are suited● Large codebases● Resource intensive● Desktop applications built using frameworks are not suited
Business suitability	<ul style="list-style-type: none">● Higher network speed● Low technical debt and cost of maintenance.● Small apps that require a temporary boost in performance	<ul style="list-style-type: none">● Database hosting most common● Improve DB scalability● Boosts fault tolerance● Data is more resilient.	<ul style="list-style-type: none">● Looking for large improvements in availability, testability, continuous delivery, reusability and lower infrastructure costs



THANK YOU

Dr. H.L. Phalachandra

phalachandra@pes.edu