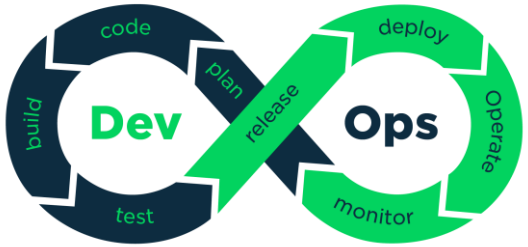
DevOps – Let the Journey Begin



Raman Khanna

Introduction

Name -

Total Experience -

Background –

Any Exposure of AWS/Git/Docker/Kubernetes/Jenkins/Terraform/Ansible

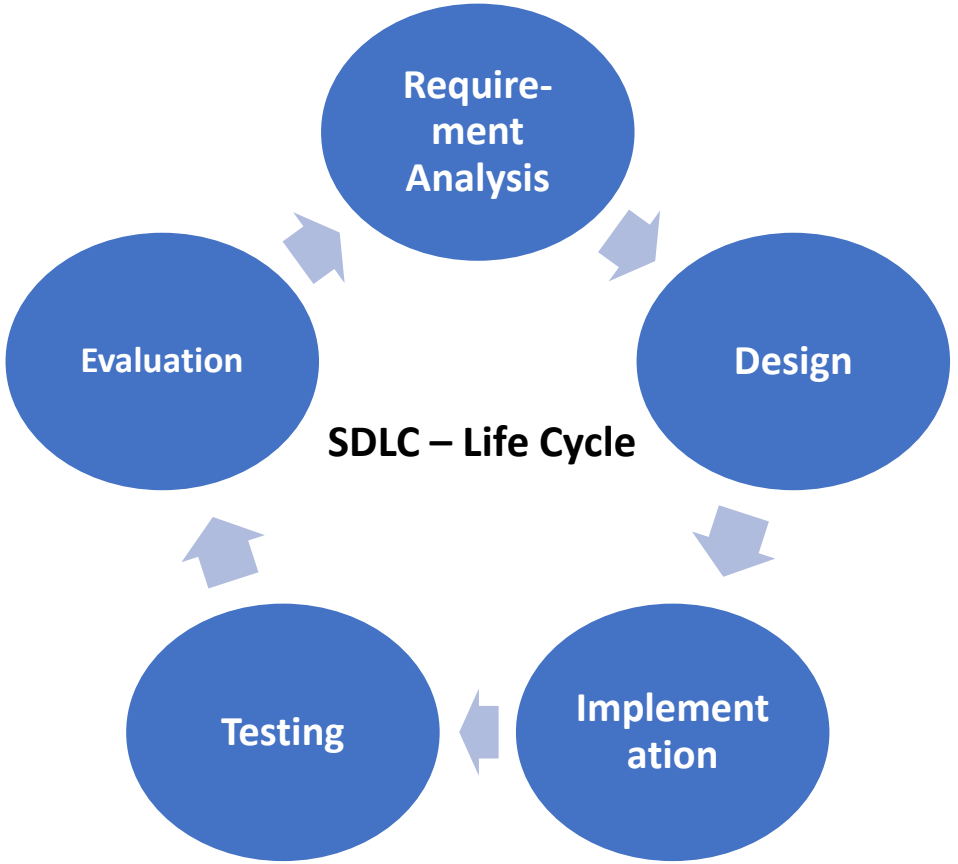
DevOps

What is DevOps?

SDLC Model

SDLC Model

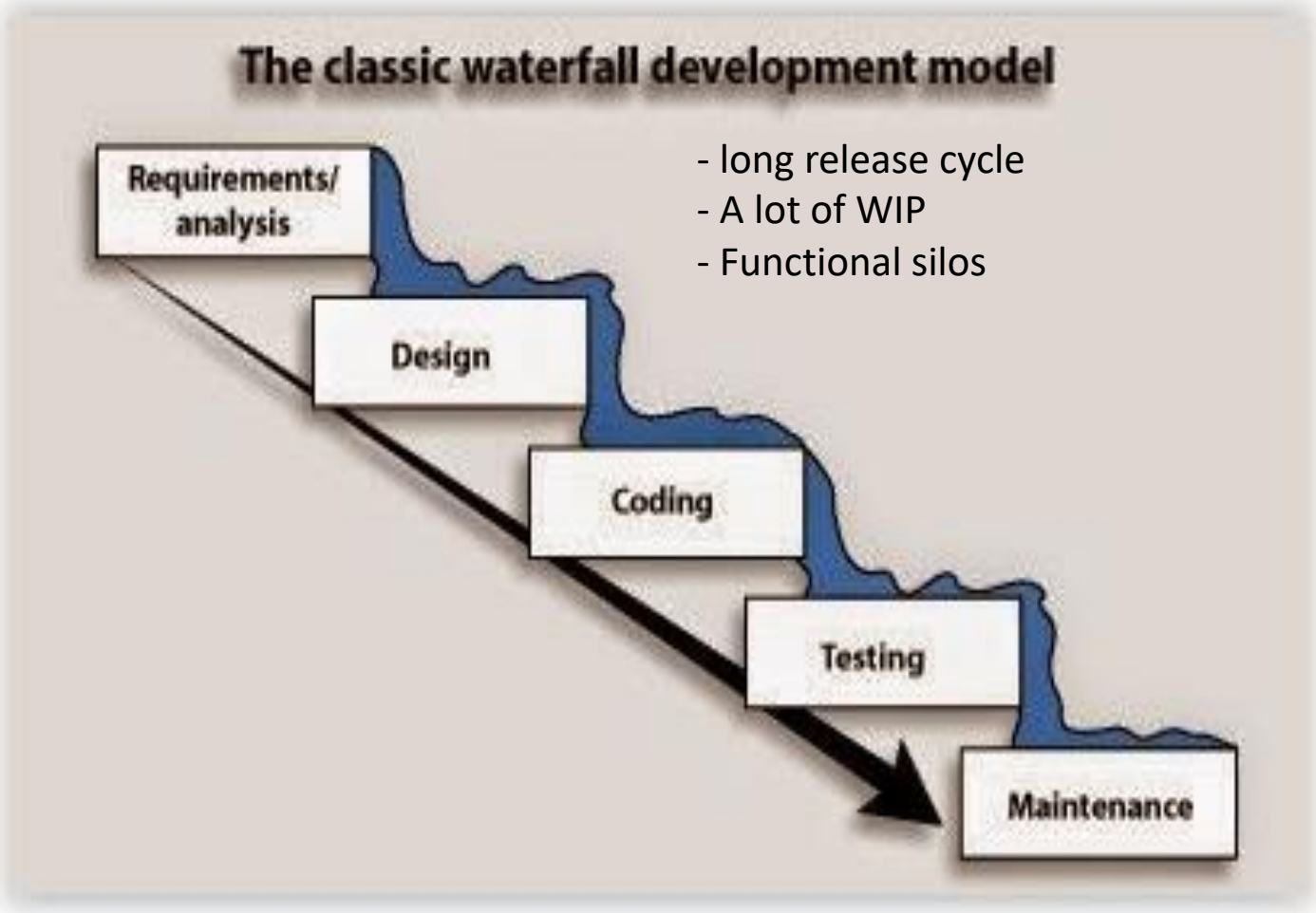
- A systems development life cycle is composed of **several clearly defined and distinct work phases** which are used by systems engineers and systems developers to plan for, design, build, test, and deliver information systems



Waterfall Development Model

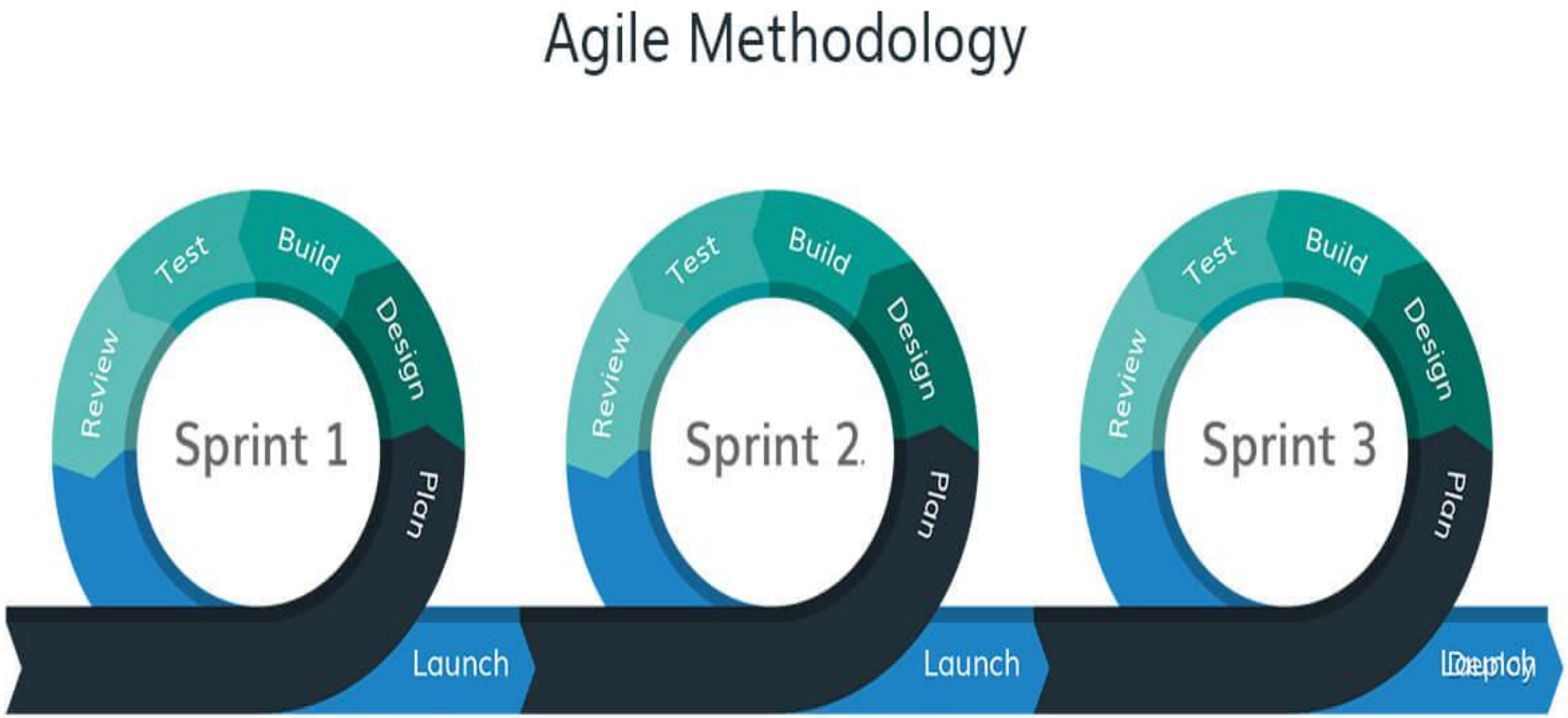
Waterfall Model

1. Determine the Requirements
2. Complete the design
3. Do the coding and testing (unit tests)
4. Perform other tests (functional tests, non-functional tests, Performance testing, bug fixes etc.)
5. At last deploy and maintain



Agile Development Model

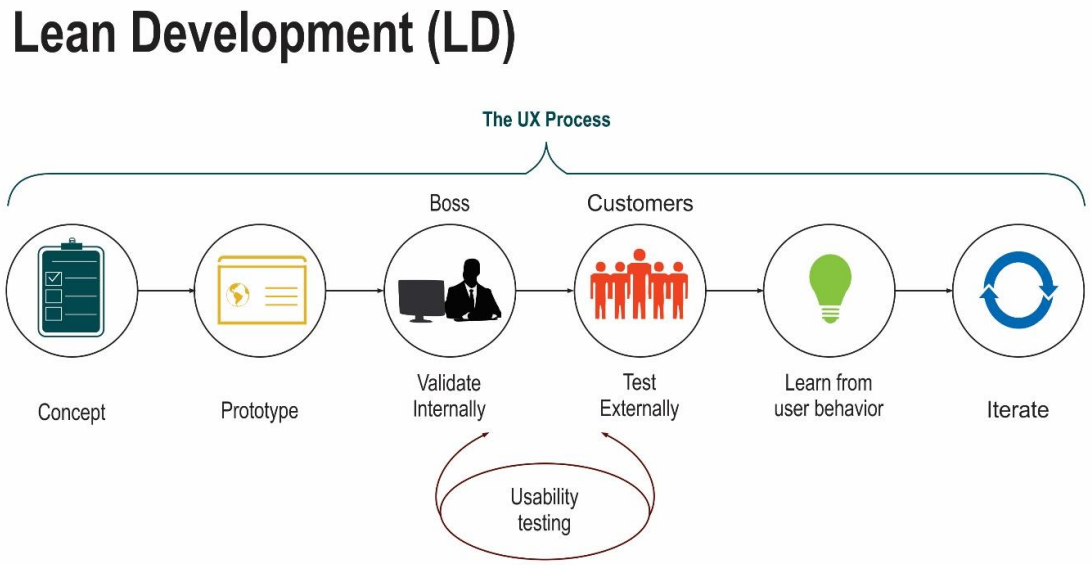
Agile

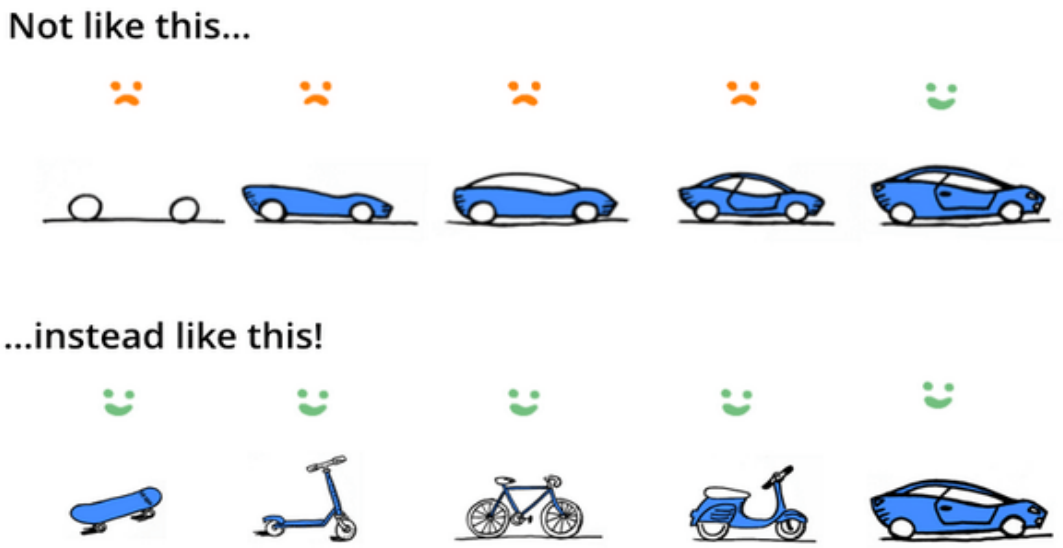


- Shorter release cycle
- Small batch sizes (MVP)
- Cross-functional teams
- Incredibly agile

Lean Development Model

Lean Development





- Suddenly ops was the bottleneck (more release less people), again WIP is more!

Challenges

Some of the challenges with the traditional teams of Development and Operations are:

A Typical Case Study

- **Development Team:**

- Monday Morning, the writing of code done, unit tests completed, code delivered to the Integration teams to get the code included in CI builds.
- To get the services tested, a ticket is opened for QA teams

- **Build/Release/Testing/Integration Team:**

- Tuesday Morning, ticket accepted, a tester put an email to the developer asking deployment instructions. There is not automated deployments, developer updated to the tester, lets come online and we will deploy the services to the QA environment together.
- Call started, developer identified the “test environment” is not compatible.
- Tuesday afternoon, a ticket raised in Ops Team with new specifications.

- **Ops Team:**

- Wednesday morning, ticket accepted, specifications checked , a new port open request was identified.
- Ticket raised for Security team, ticket accepted, change approved, port opened, email received by the Ops team the work is done.

A Typical Case Study

A Typical Case Study

- **Ops Team:**

- Identified the provisioning requirements again and started work on building the environment.

- **Build/Release/Testing/Integration Team:**

- Thursday Morning, updates received - the environment is ready. Developer and Tester again on call to deploy new services. Services deployed; tester is running test scripts. Next phase is to run regression test cases. Again a new ticket is raised for new test data with production teams and day ends.

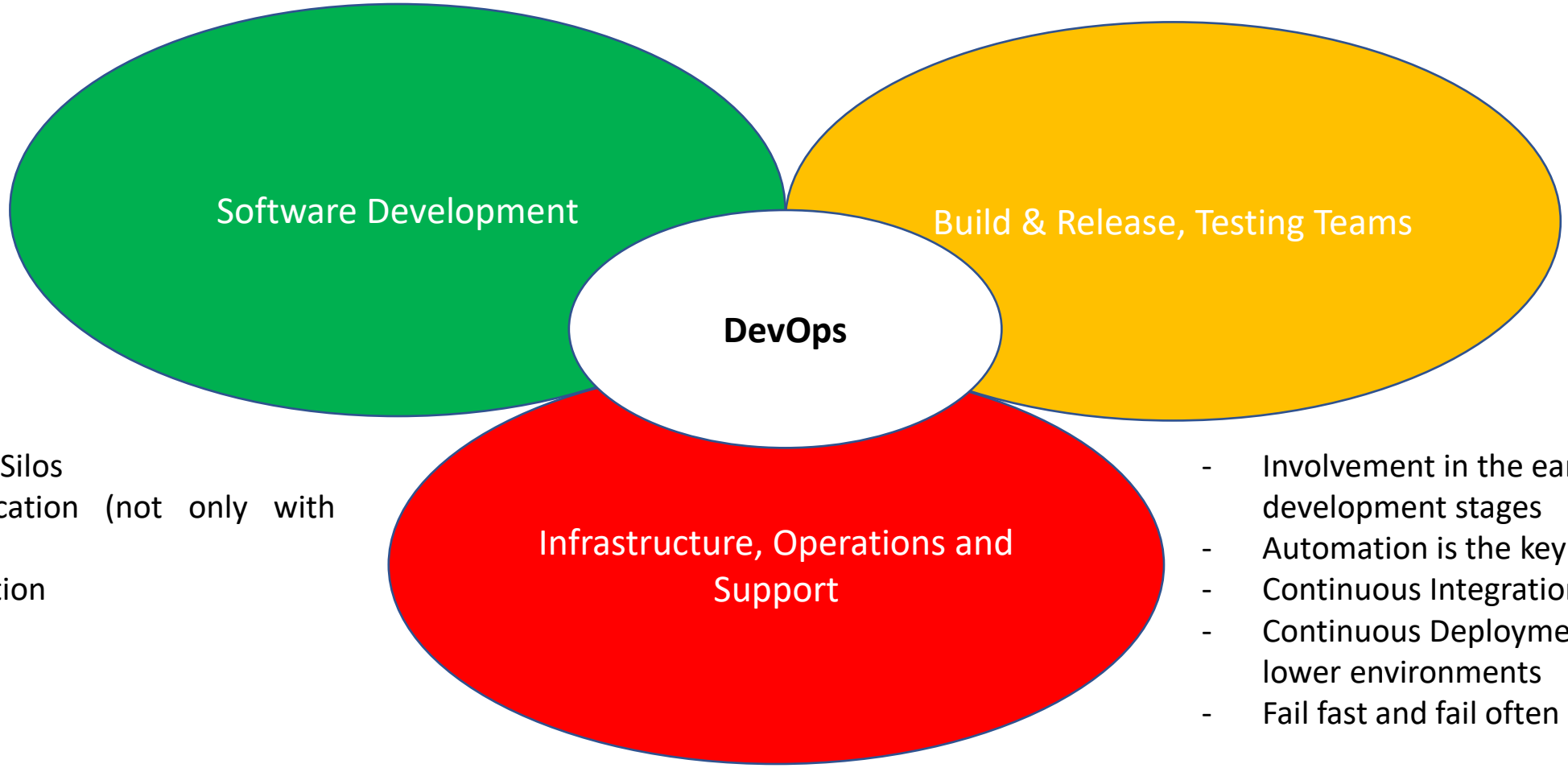
- **Ops Team:**

- Its Friday and the work is not on full swing, ticket accepted but not worked as production team has to complete rest of the works. Somehow the test data is gathered by Friday Evening.

- **Build/Release/Testing/Integration Team:**

- Monday morning, tester gets the data, regression tests run, a defect found, and ticket returned to the development team.

DevOps



- Break the Silos
- Communication (not only with emails)
- Collaboration
- Trust

- Involvement in the early development stages
- Automation is the key
- Continuous Integration
- Continuous Deployments in the lower environments
- Fail fast and fail often

DevOps Model

Collaboration

Communication

Continuity

Automation

Project Management

Customer

Build

Test

Deploy

Planning

Monitoring

Continuous Feedback

Continuous Improvement

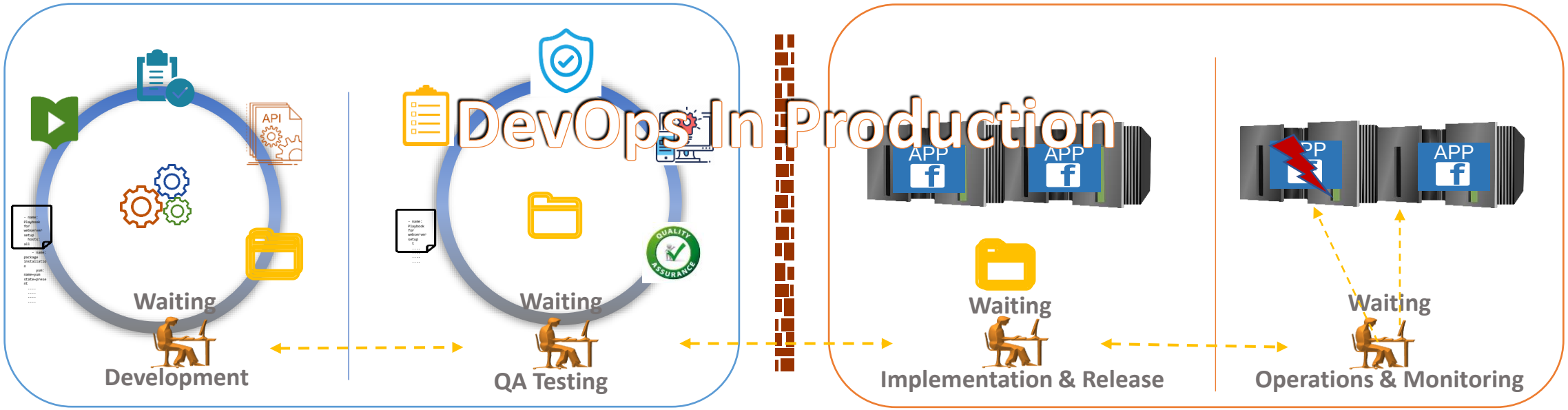
Continuous Planning

Continuous Integration

Continuous Delivery

Continuous Deployment

Continuous Monitoring



Development

Operations

DevOps Essence

Efficiency - Faster time to market

Predictability - Lower failure rate of new releases

Reproducibility – Version everything

Maintainability - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system

DevOps Core Principals

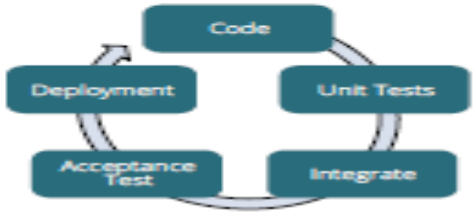
1. Customer-Centric Action

4. Cross-Functional Autonomous Teams

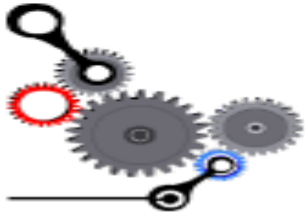
2. Create with the End in Mind

5. Continuous Improvement

3. End-to-End Responsibility



6. Automate Everything You can



How to Build DevOps Organization Culture

Retention is as important as recruitment

Establish Cross-functional team structure

Small teams are better

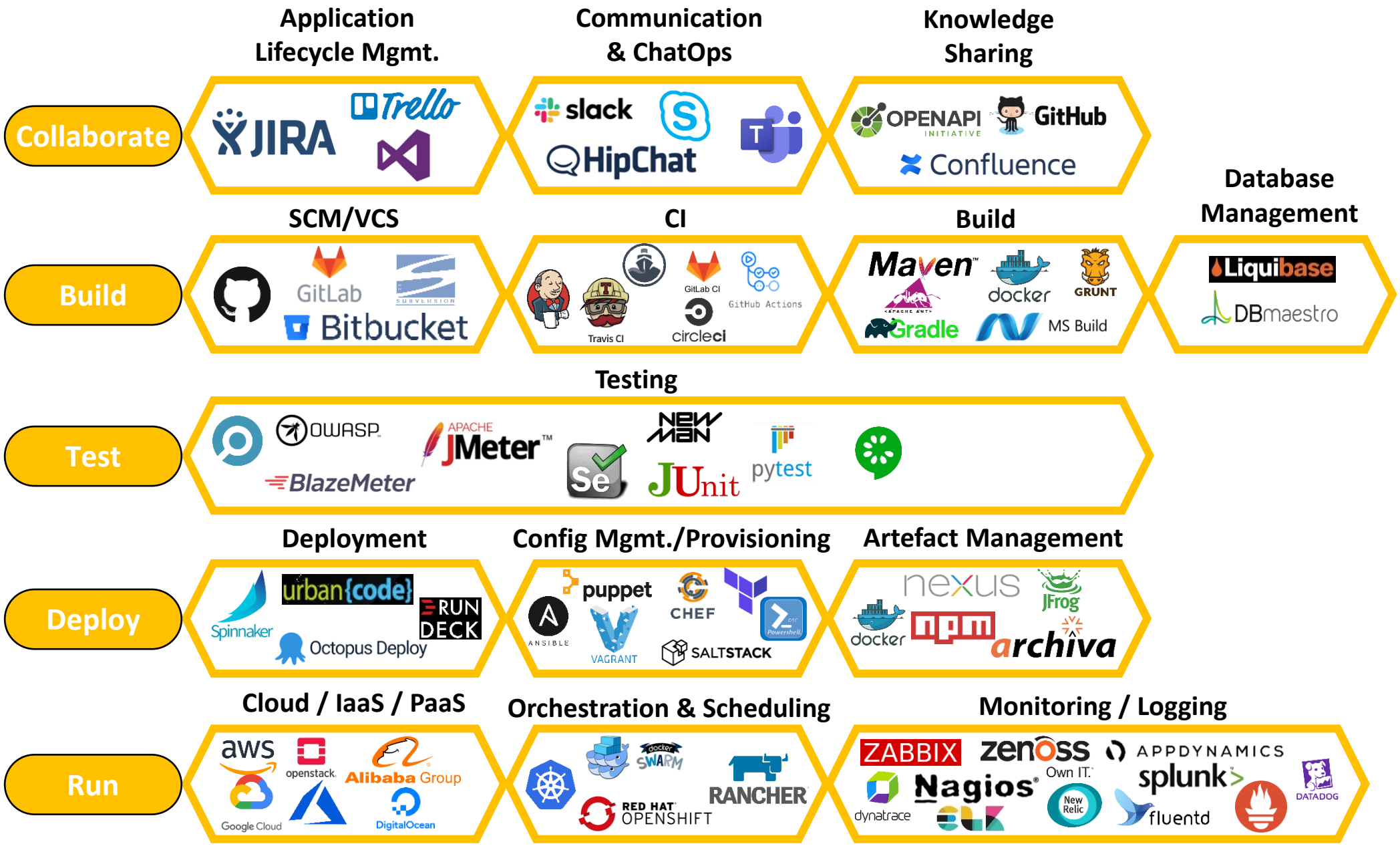
Cool tools can attract and retain

Give autonomy

Automate with existing staffs and give them a chance to learn

Take out few resources from each team, build a new virtual team for automation.

DevOps Toolsets



JIRA –
A Project Management tool

JIRA

A Product from Atlassian family

PLAN, TRACK, & SUPPORT

Jira Software
Project and issue tracking

Jira Align
Enterprise agile planning

Jira Core
Essential business management

Jira Service Desk
Collaborative IT service management

Opsgenie
Modern incident response

Statuspage
Incident communication

COLLABORATE

Confluence
Document collaboration

Trello
Collaborate visually on any project

CODE, BUILD, & SHIP

Bitbucket
Git code management

Sourcetree
Git and Mercurial desktop client

Bamboo
Integration and release management

SECURITY & IDENTITY

Atlassian Access
Security and control for cloud

Crowd
User management for self-managed environments

JIRA

A Project Management Tool made by Atlassian (Australian company).

You can manage Project, Tests cases, defects, Scrum Meeting etc.

A #1 tool for Project management due to unmatched benefits of the tool.

Everything which you need to manage projects and team is available.

Fully customizable

Hundreds of Apps to enhance the functionality.

JIRA

CLOUD	SERVER	DATA CENTER
Fast Start-up	Self-installed and managed	Same as Server, but at scale
No maintenance	One-time installation fee + ongoing support fees	Enterprise level security, availability, performance
Monthly or annual subscription	Local data	
	More customizability	

JIRA

	Free	Standard	Premium	Enterprise
	$0	**$7**	**$14**	Talk to our sales team about pricing for Cloud Enterprise
	Always free for 10 users	per user (average)	per user (average)	Contact us
	Get started	$7 a month	$14 a month	
		Start trial	Start trial	
Features				
User limit (per site)	10 users	10,000 users	10,000 users	10,000 users
Site limit	One	One	One	Unlimited
Scrum and Kanban boards	✓	✓	✓	✓
Backlog	✓	✓	✓	✓
Agile reporting	✓	✓	✓	✓
Customizable workflows	✓	✓	✓	✓
Apps and integrations ↗	✓	✓	✓	✓
Audit logs	-	✓	✓	✓
Anonymous access	-	✓	✓	✓
Admin insights	-	-	✓	✓
IP allowlisting	-	-	✓	✓
Sandbox COMING SOON	-	-	✓	✓
Release tracks COMING SOON	-	-	✓	✓
Data residency	-	-	-	✓
Storage	2 GB file storage	250 GB file storage	Unlimited storage	Unlimited storage
Support ↗	Community Support	Local Business Hours	24/7 Premium Support	24/7 Enterprise Support
Uptime SLA ↗	-	-	99.90%	99.95%
Org-Level Billing	-	-	-	✓

Agile Principles with JIRA

Iterative Development

Adaptive to changing Requirements

Frequent Delivery

Cross-functional team structure

Scrum and Kanban Methodologies

JIRA

	Scrum	Kanban
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint	Continuous delivery
Roles	Product owner, scrum master, development team	No required roles
Key metrics	Velocity	Lead time, cycle time, WIP
Change philosophy	Teams should not make changes during the sprint.	Change can happen at any time

Scrum

Breaks projects into Epics and Stories

Epics – Large Stories or Work Items, usually broken into small storeies

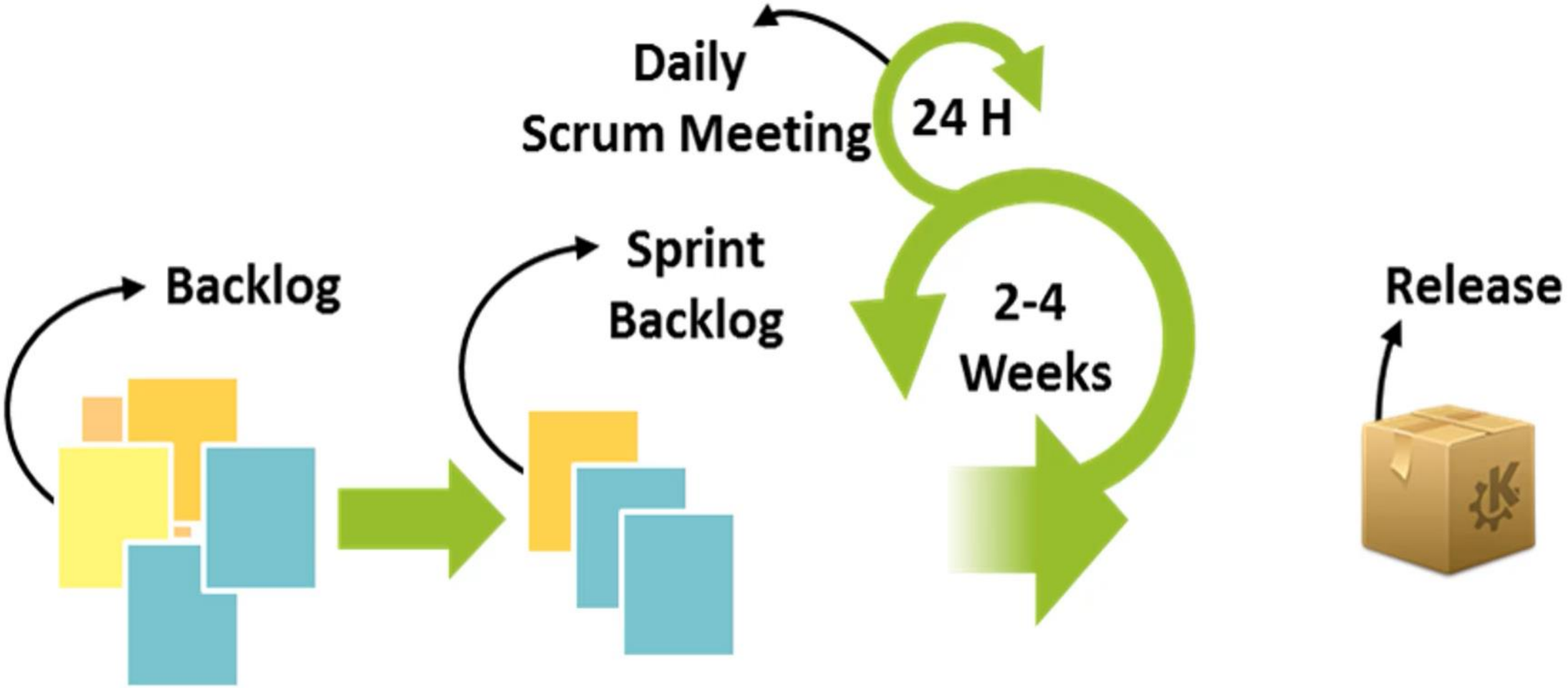
User Stories: Smallest Unit of work

User stories can be prioritized.

Development is performed in short cycles.

Terms like Product Owner, Development Team, Scrum Master

Scrum flow


Daily
Scrum Meeting
24 H
Backlog
Sprint
Backlog
2-4
Weeks
Release

Kanban

Simpler form of Agile Development

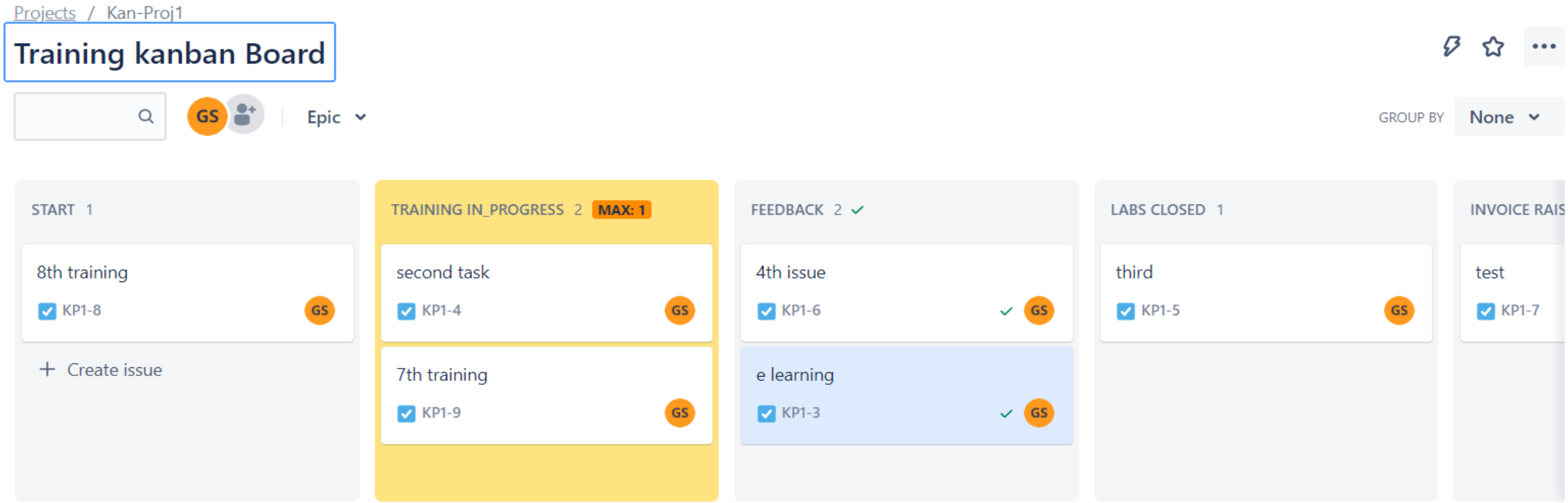
Define Flow of work

Kanban Keyboard - Visualize the workflow

Limit on Work In Progress (WIP) Items

Monitor, Adapt and Improve

Kanban


Projects / Kan-Proj1
Training kanban Board
GS
Epic
GROUP BY
None
START 1
8th training
KP1-8
GS
Create issue
TRAINING IN_PROGRESS 2
MAX: 1
second task
KP1-4
GS
7th training
KP1-9
GS
FEEDBACK 2
4th issue
KP1-6
GS
e learning
KP1-3
GS
LABS CLOSED 1
third
KP1-5
GS
INVOICE RAIS
test
KP1-7

Introduction to Cloud Computing

What is Cloud?

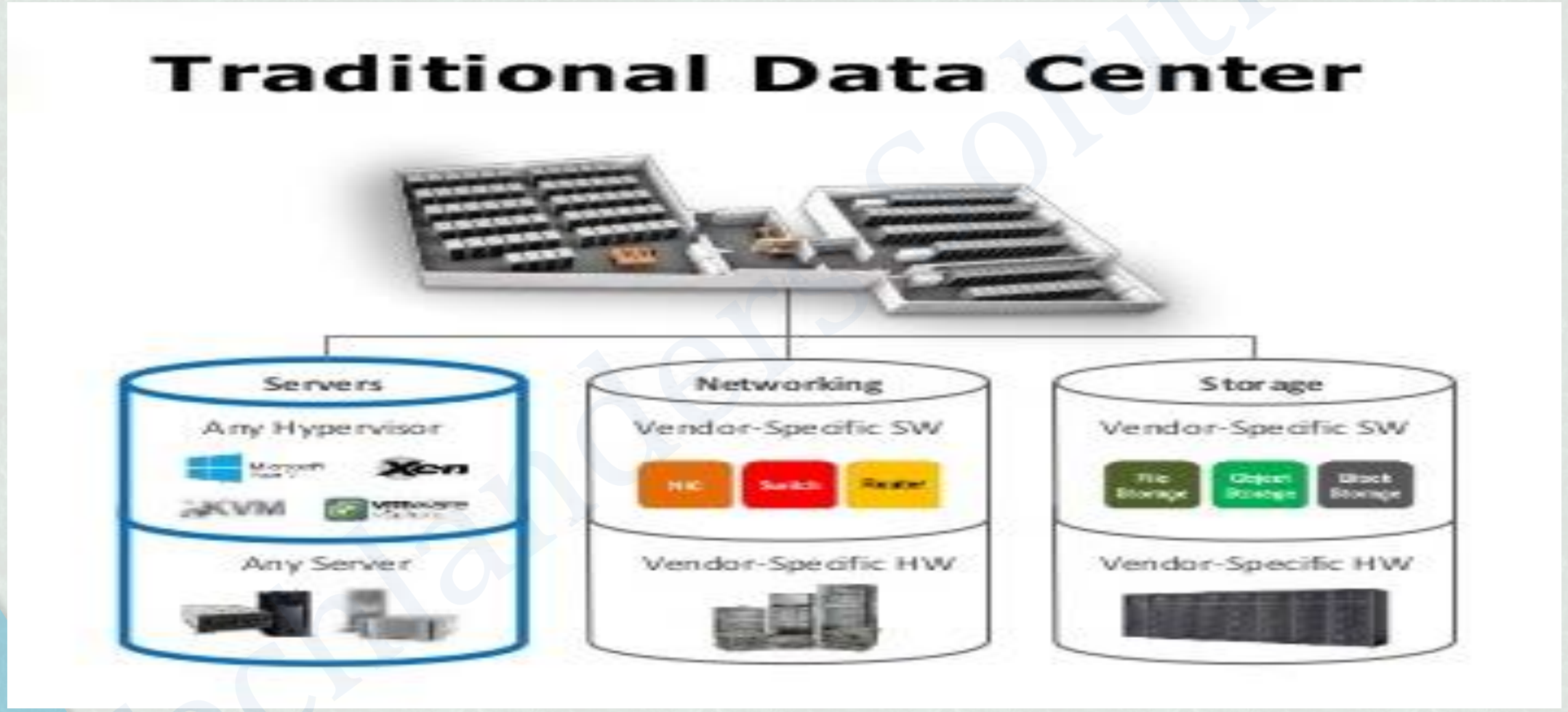
Introduction to Cloud Computing

In simple words, Cloud computing is – Placing your data on someone else's datacenter, letting them manage underline hardware Infrastructure (optionally underline Database or applications too); while having your full control on the data, and accessing that data through Internet or dedicated network.

Cloud computing is a model for enabling **universal**, **on-demand** access to a **shared pool** of configurable computing resources (e.g., computer networks, servers, storage, applications and services), which can be **rapidly provisioned** and **released** with **minimal management effort** on **Pay-per-use basis.**

Free available cloud Examples:	Gmail, IRCTC, WhatsApp/Facebook
Paid available cloud Examples:	AWS, Azure(Microsoft), Oracle Cloud

Traditional DataCenters



Traditional DataCenters

- Main issues with Traditional IT Infrastructure.

- Infrastructure is not a core business
- Hard to Scale
- Dedicated Infrastructure teams
- Dedicated Datacenters
- Dependency on vendors (servers, switches, cables etc.)
- Underutilized Resources
- High Cost
- Difficult Capacity Planning
- On-Spot demands were hard to manage
- Provisioning resources was very time consuming

Why cloud?

- To overcome all of the discussed challenges, IT infrastructure domain drifted towards Service based model which is a real “cloud computing”

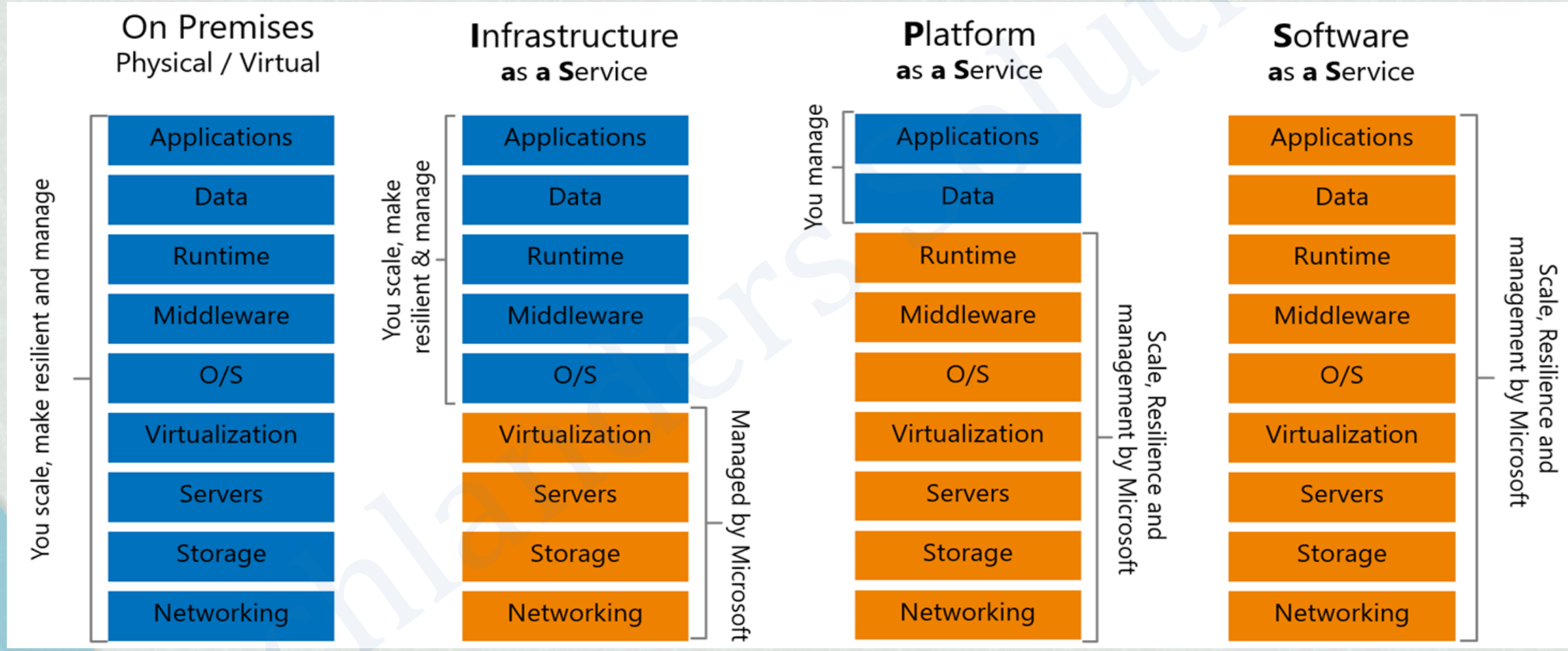
- No Dedicated Datacenter
- No Different Infrastructure Teams
- Higher/Faster Scalability
- Elasticity
- Pay per use model
- Option to adopt high availability
- Better performance
- Instant provisioning
- Optimized use of resources
- On demand scaling to any extent
- No to worry about capacity planning

Cloud Advantages

Cloud Service Models

- There are three Cloud Computing Service Models:
  - Infrastructure as a Service (IaaS)
  - Platform as a Service (PaaS)
  - Software as a Service (SaaS)

Responsibility- Who owns What?



Responsibility- Who owns What?

Cloud Service Models - IaaS

IaaS is the most basic Cloud Service Model

It offers Underline Infrastructure for Compute, Storage and Networking

Infrastructure can be selected by customers as per their choice and Pay-per-use model.

- Examples: Bare metal servers, virtual Instances, Load balancers

IaaS - Benefits

- Drastic reduction in capital investment
- Easily Scalable
- Pay only for the used resources
- High Flexibility
- Reduced infrastructure support teams

Cloud Service Models - PaaS

Another service model, where cloud provider manages the OS & middleware part, along with IaaS

Provide capability to deploy applications on cloud infrastructure without managing underline Infra

Consumers are responsible for managing deployed applications and their environment specific configurations

- Examples: webservers and databases

PaaS - Benefits

- Includes all IaaS benefits
- No upfront licensing cost
- More reduction in Infrastructure support team
- Rapid time to market

Cloud Service Models - SaaS

SaaS deliver complete application to the consumers over the internet.

Consumers are not responsible for managing any application or underlying infrastructure.

SaaS application are delivered as “one-to-many” model.

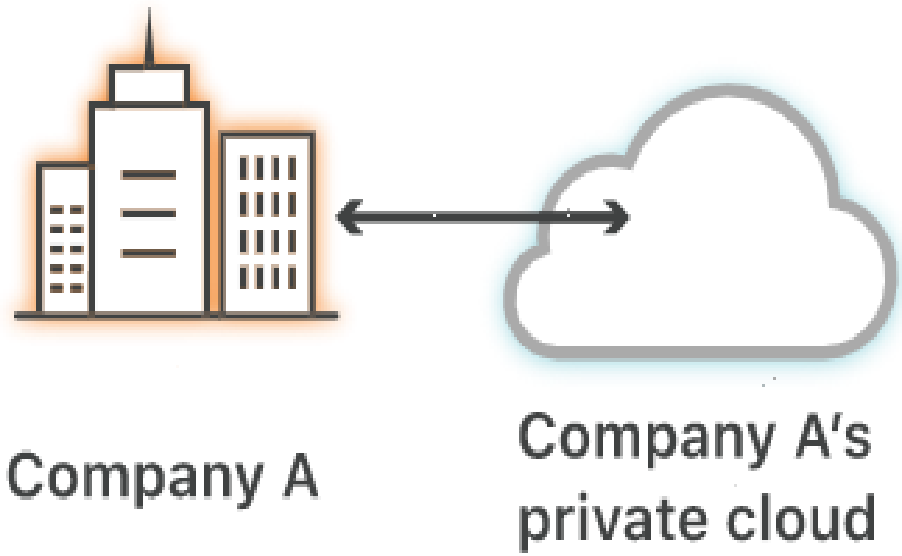
- Examples: office365, Gmail, WhatsApp, JIRA, GIT, Service Now

SaaS - Benefits

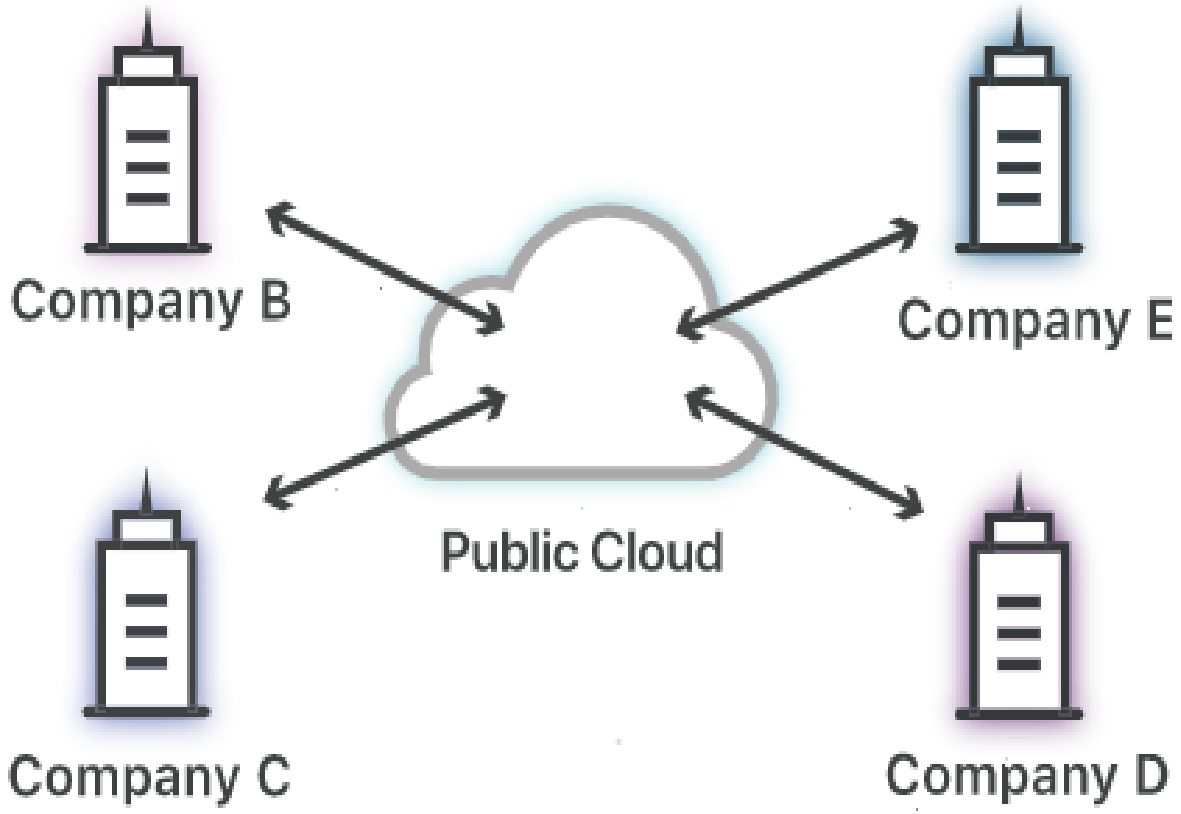
- Includes all discussed benefits which we get in PaaS
- Ability to access from anywhere
- Ability to access from multiple devices
- No installations and maintenance requirements
- No Application management/Licensing Required

Cloud Essentials Characteristics

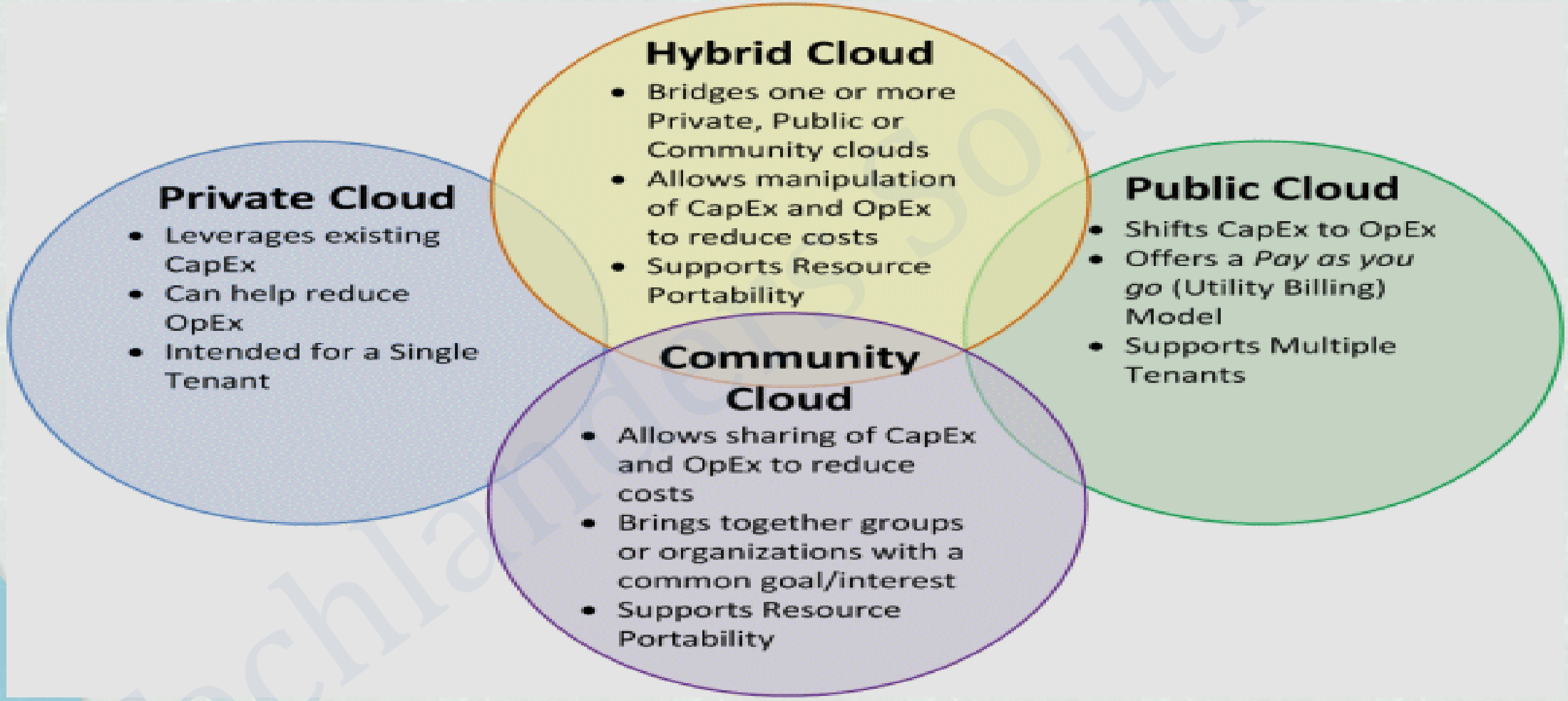
Private cloud



Public cloud shared by multiple companies



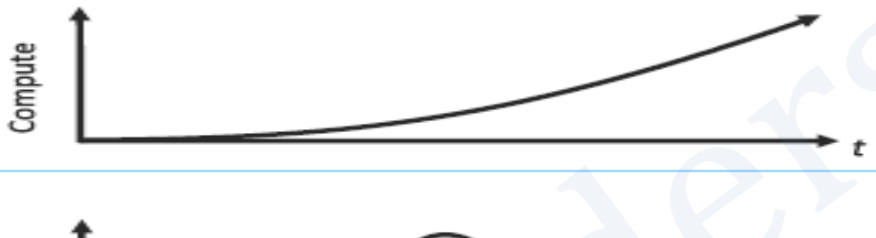
Cloud Deployments Types



Cloud's Major Use Cases

On and Off

On and off workloads (e.g. batch job)
Over provisioned capacity is wasted
Time to market can be cumbersome

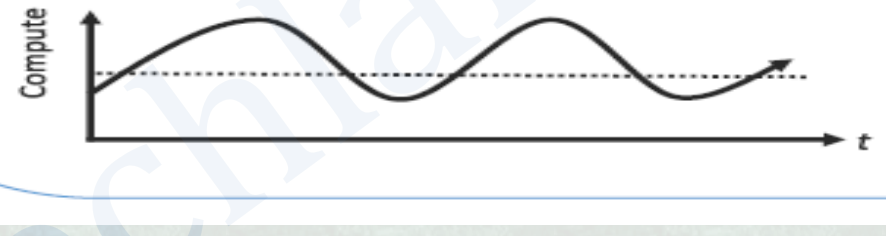


Growing Fast

Successful services needs to grow/scale
Keeping up with growth is a big IT challenge
Cannot provision hardware fast enough

Unpredictable Bursting

Unexpected/unplanned peak in demand
Sudden spike impacts performance
Cannot over provision for extreme cases

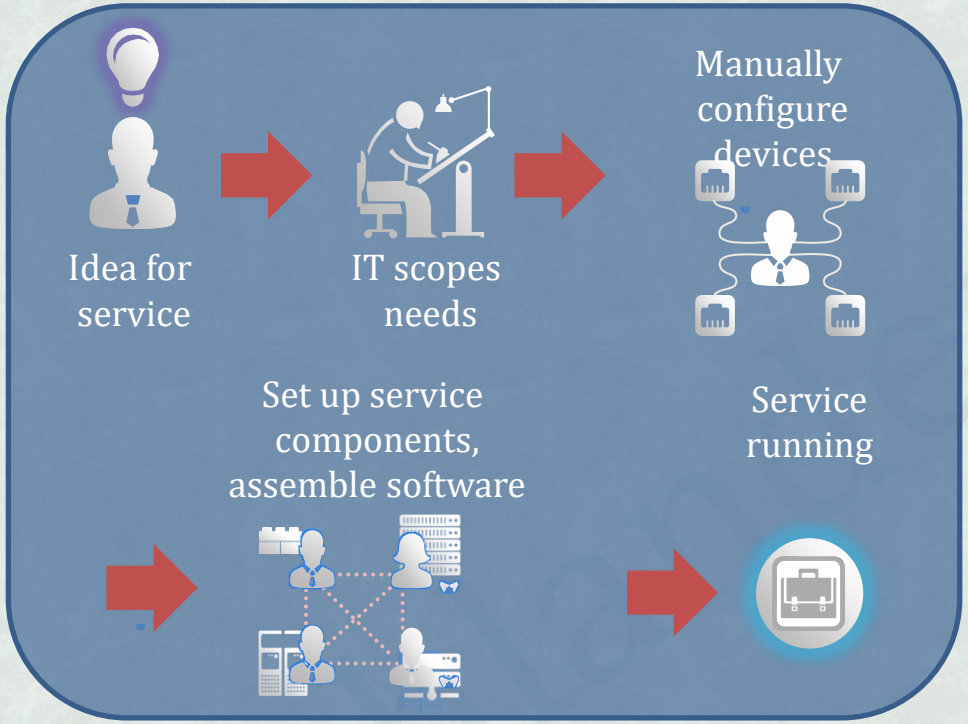


Predictable Bursting

Services with micro seasonality trends
Peaks due to periodic increased demand
IT complexity and wasted capacity

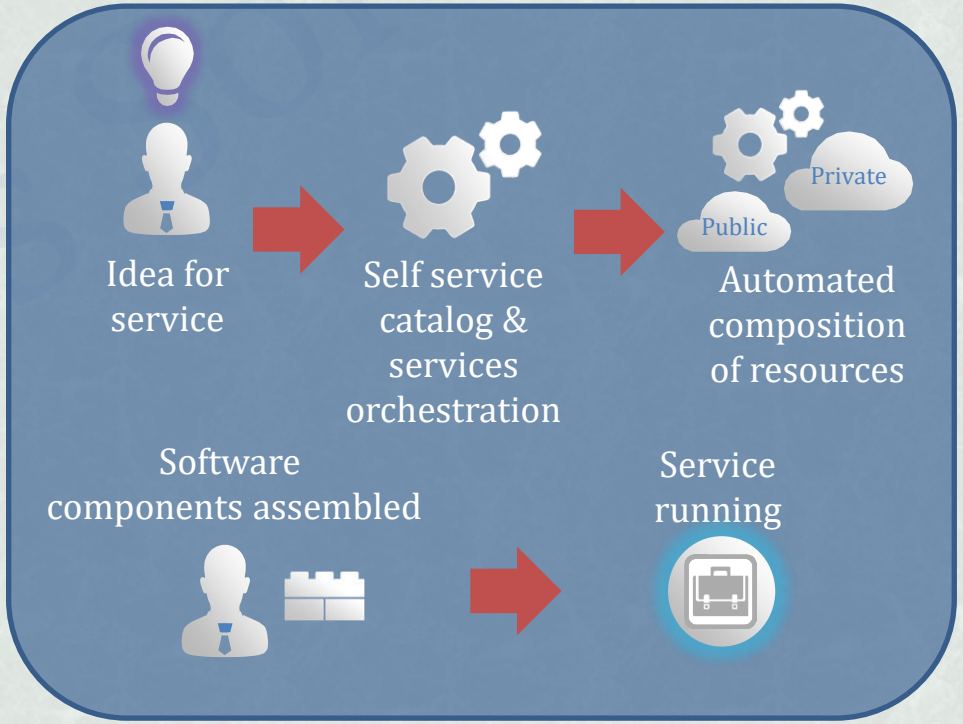
Business Impact of Cloud

Traditional Datacenter



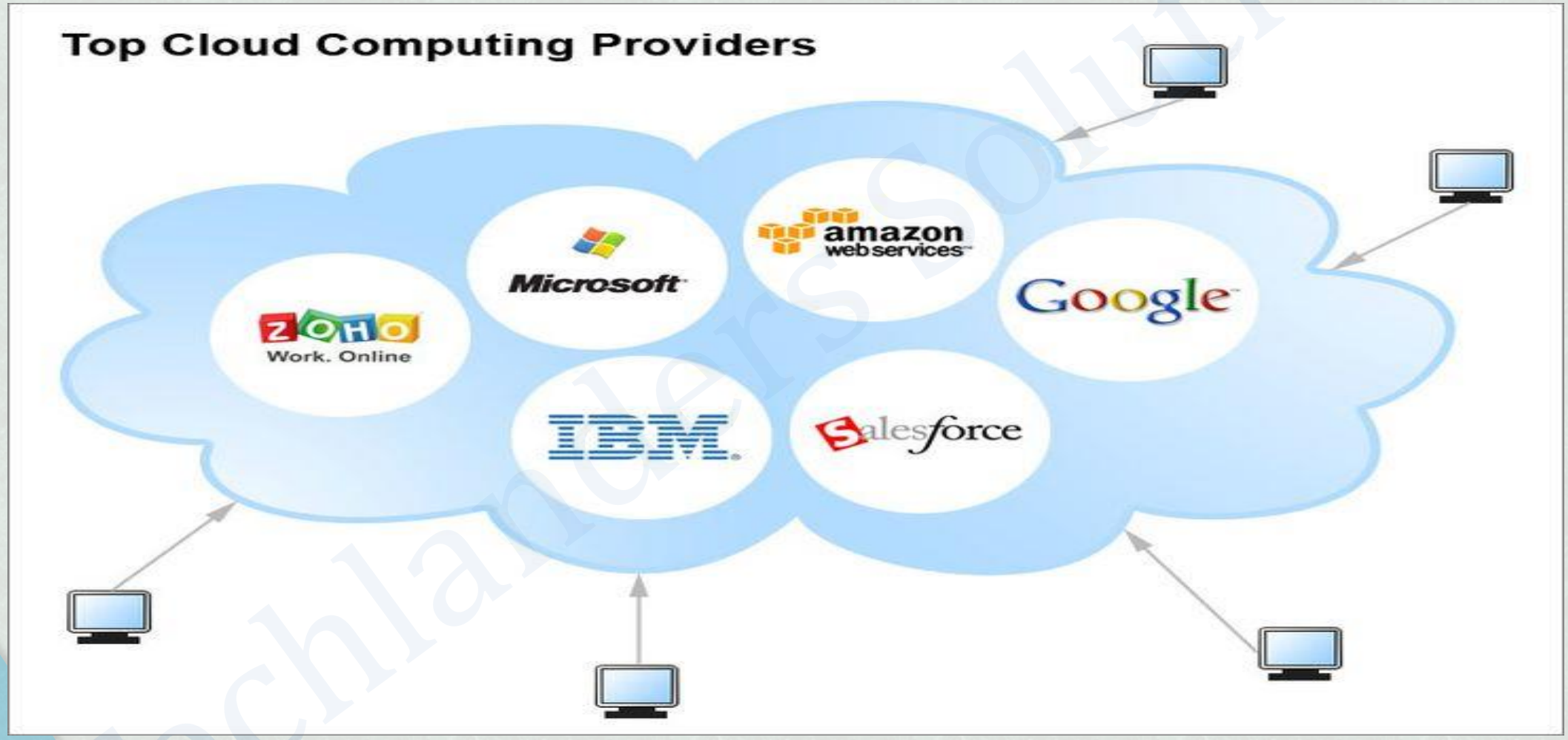
Time to Provision New Service: Months

Cloud Infrastructure

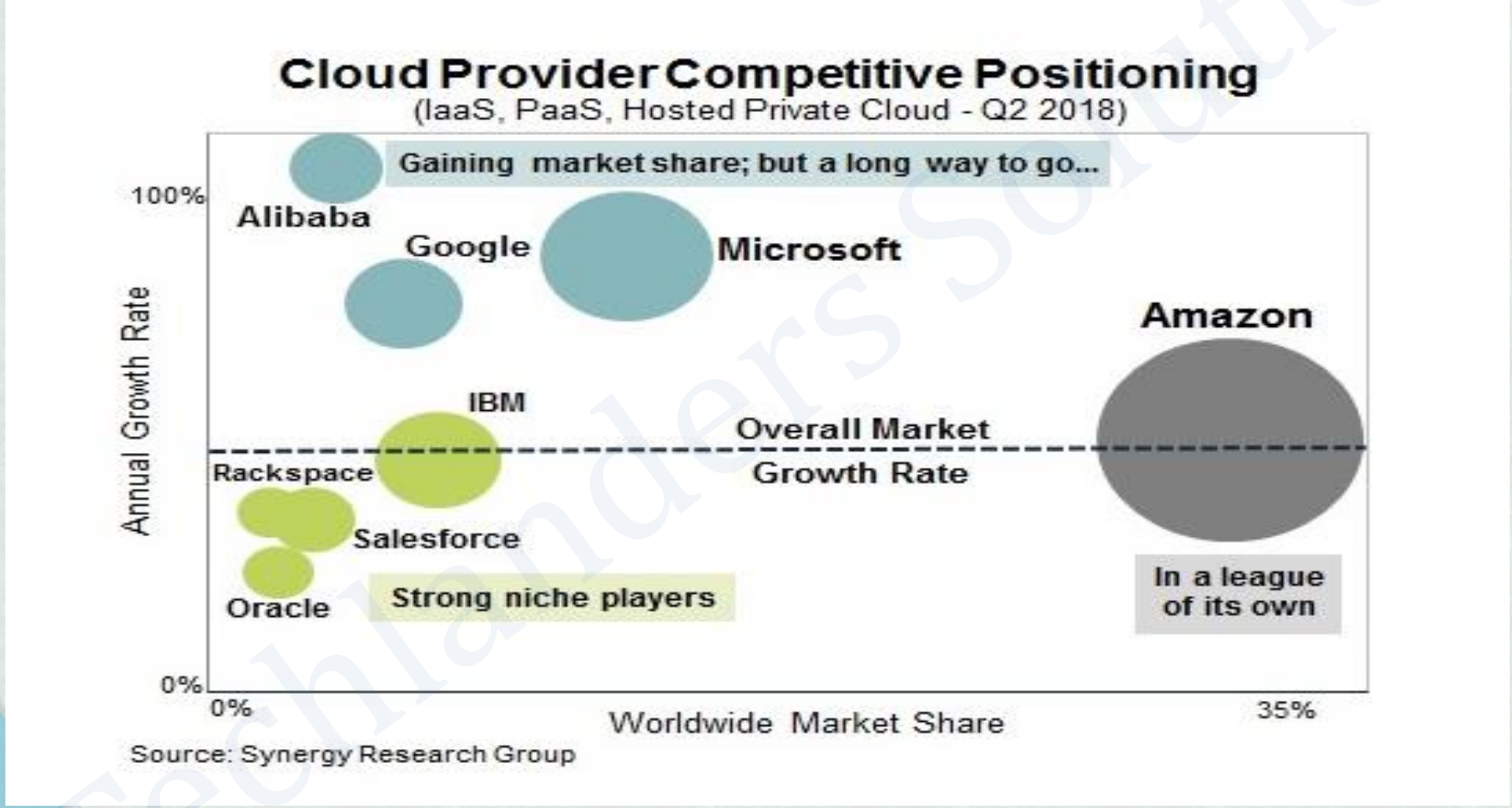


Time to Provision New Service: Minutes

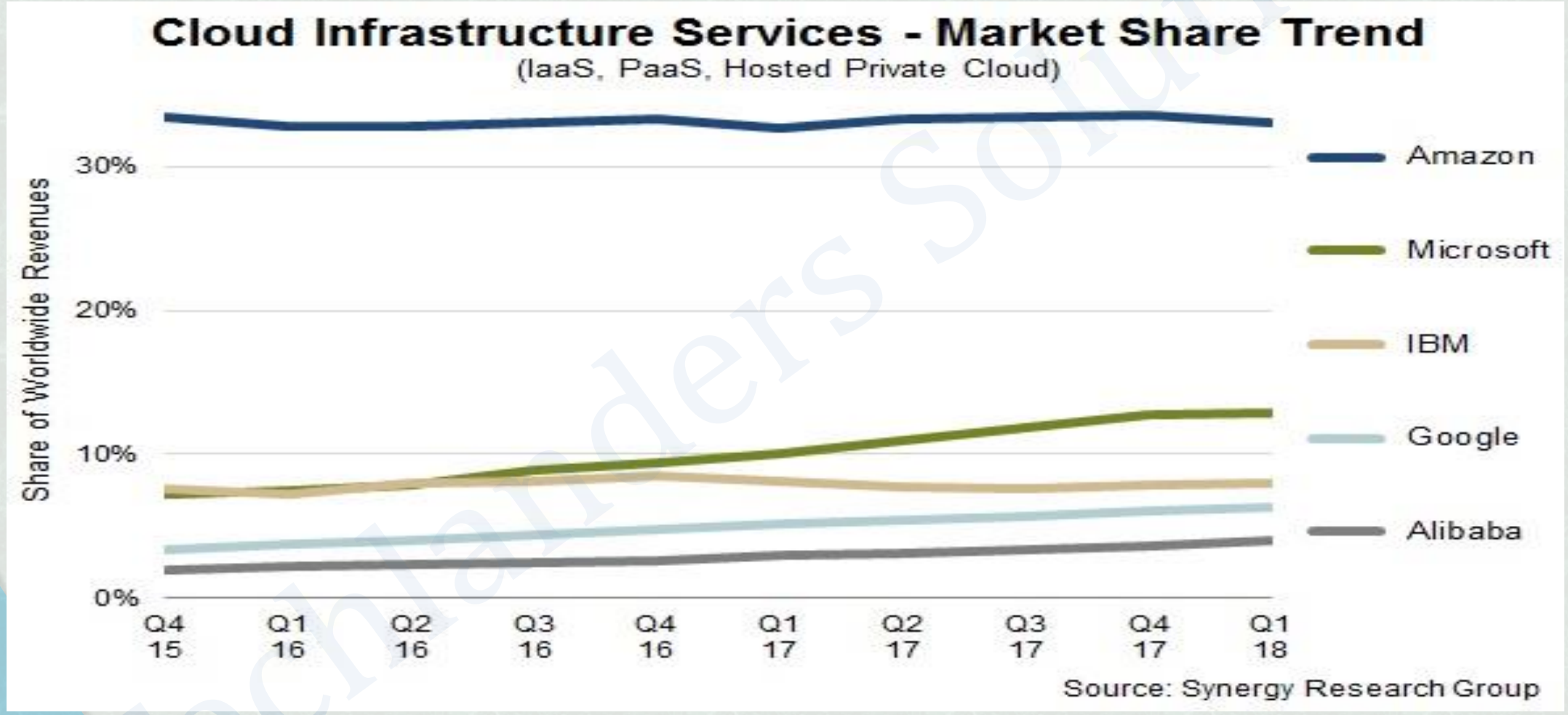
Major Cloud Vendors



Who stands where?



Who stands where?



Knowledge Checks

- Which Service Level (IaaS, PaaS, SaaS) provides you most control?
- What is Hybrid Cloud?
- Can two public clouds be connected?
- Connecting two public clouds, will be know as public cloud or Hybrid?
- Cloud provided Database, is a PaaS or SaaS?

AWS
(Amazon Cloud)

Amazon Web Services

- AWS (Amazon Web Services) is a group of web services (also known as cloud services) being provided by Amazon since 2006.
- AWS provides huge list of services starting from basic IT infrastructure like CPU, Storage as a service, to advance services like Database as a service, Serverless applications, IOT, Machine Learning services etc..
- Hundreds of instances can be build and use in few minutes as and when required, which saves ample amount of hardware cost for any organizations and make them efficient to focus on their core business areas.
- Currently AWS is present and providing cloud services in more than 190 countries.
- Well-known for IaaS, but now growing fast in PaaS and SaaS.

Why AWS?

- **Low Cost:** AWS offers, pay as you go pricing. AWS models are usually cheapest among other service providers in the market.
- **Instant Elasticity:** You need 1 server or 1000's of servers, AWS has a massive infrastructure at backend to serve almost any kind of infrastructure demands, with pay for what you use policy.
- **Scalability:** Facing some resource issues, no problem within seconds you can scale up the resources and improve your application performance. This cannot be compared with traditional IT datacenters.
- **Multiple OS's:** Choice and use any supported Operating systems.
- **Multiple Storage Options:** Choice of high I/O storage, low cost storage. All is available in AWS, use and pay what you want to use with almost any scalability.
- **Secure:** AWS is PCI DSS Level1, ISO 27001, FISMA Moderate, HIPAA, SAS 70 Type II passed. In-fact systems based on AWS are usually more secure than in-house IT infrastructure systems.

AWS Global Infrastructure

AWS Regions:

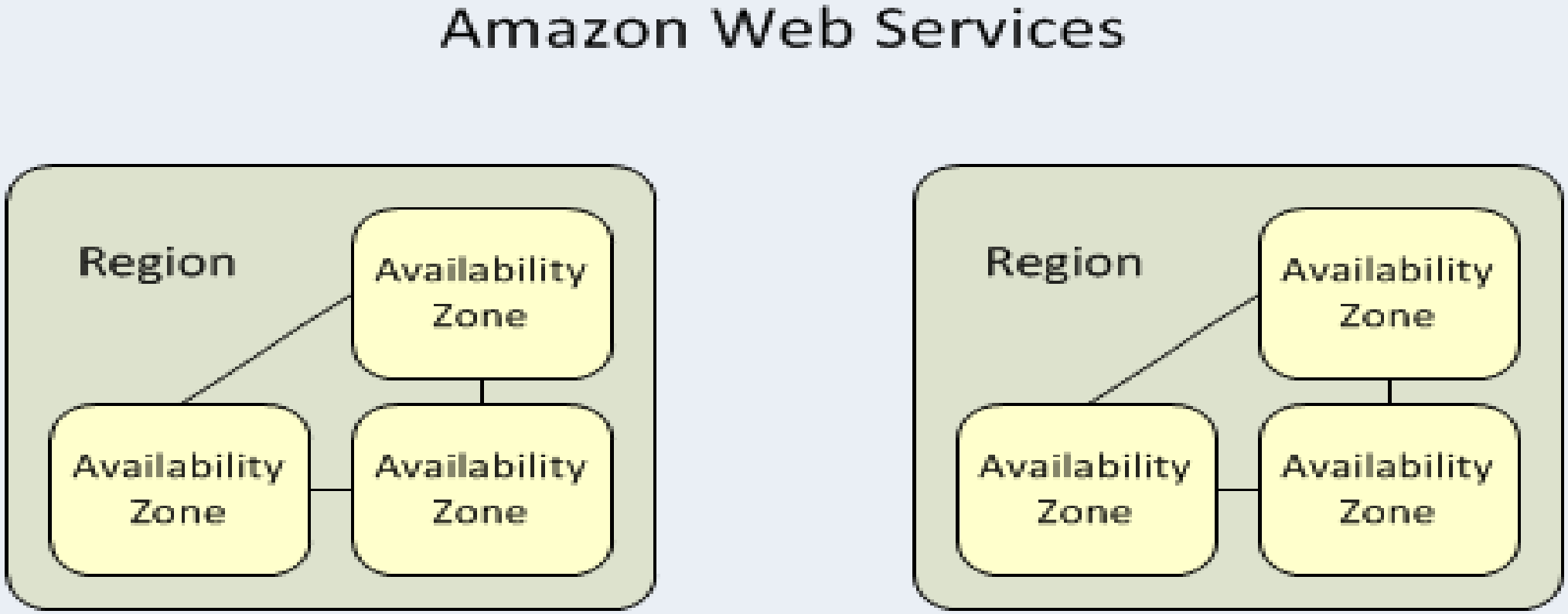
- Geographic Locations
- Consists of at least two Availability Zones(AZs)
- All of the regions are completely independent of each other with separate Power Sources, Cooling and Internet connectivity.

AWS Availability Zones

- AZ is a distinct location within a region
- Each zone is insulated (with low-latency links) from other to support single point of failures
- Each Region has minimum two AZ's
- Most of the services/resources are replicated across AZs for HA/DR purpose.

Note: Resources aren't replicated across regions unless you do so specifically.

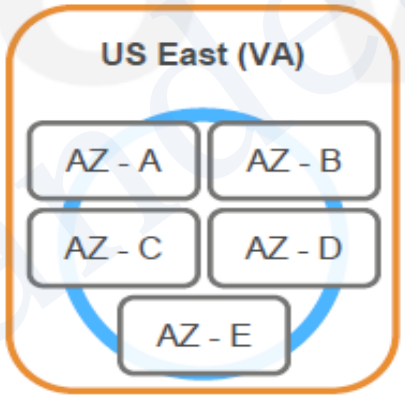
AWS Global Infrastructure

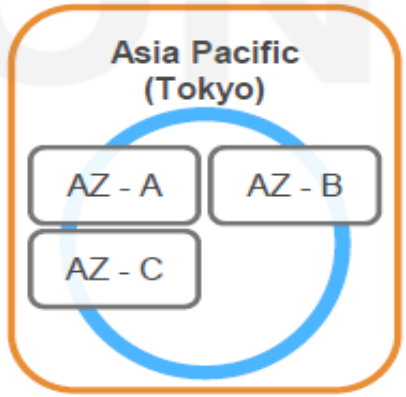


AWS Global Infrastructure

At least 2 AZs per region.

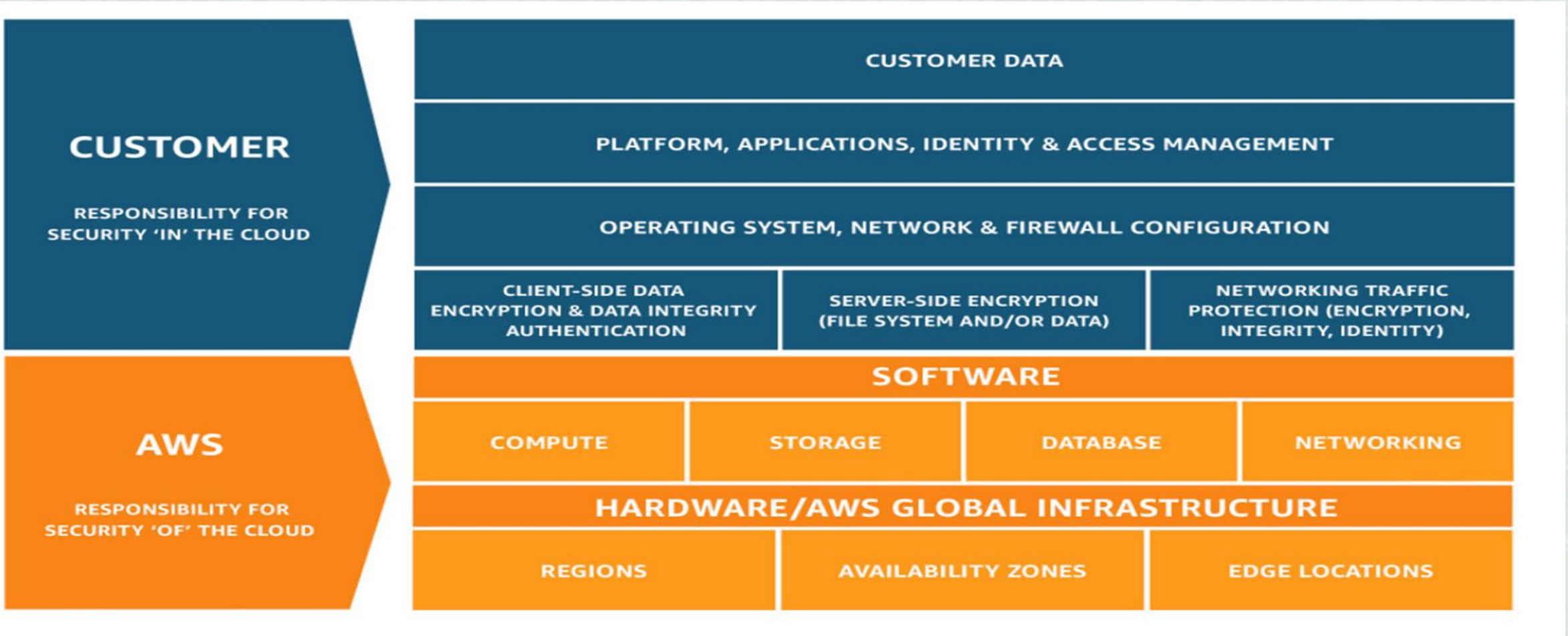
- Examples:
  - US East (N. Virginia)
    - us-east-1a
    - us-east-1b
    - us-east-1c
    - us-east-1d
    - us-east-1e
  - Asia Pacific (Tokyo)
    - ap-northeast-1a
    - ap-northeast-1b
    - ap-northeast-1c



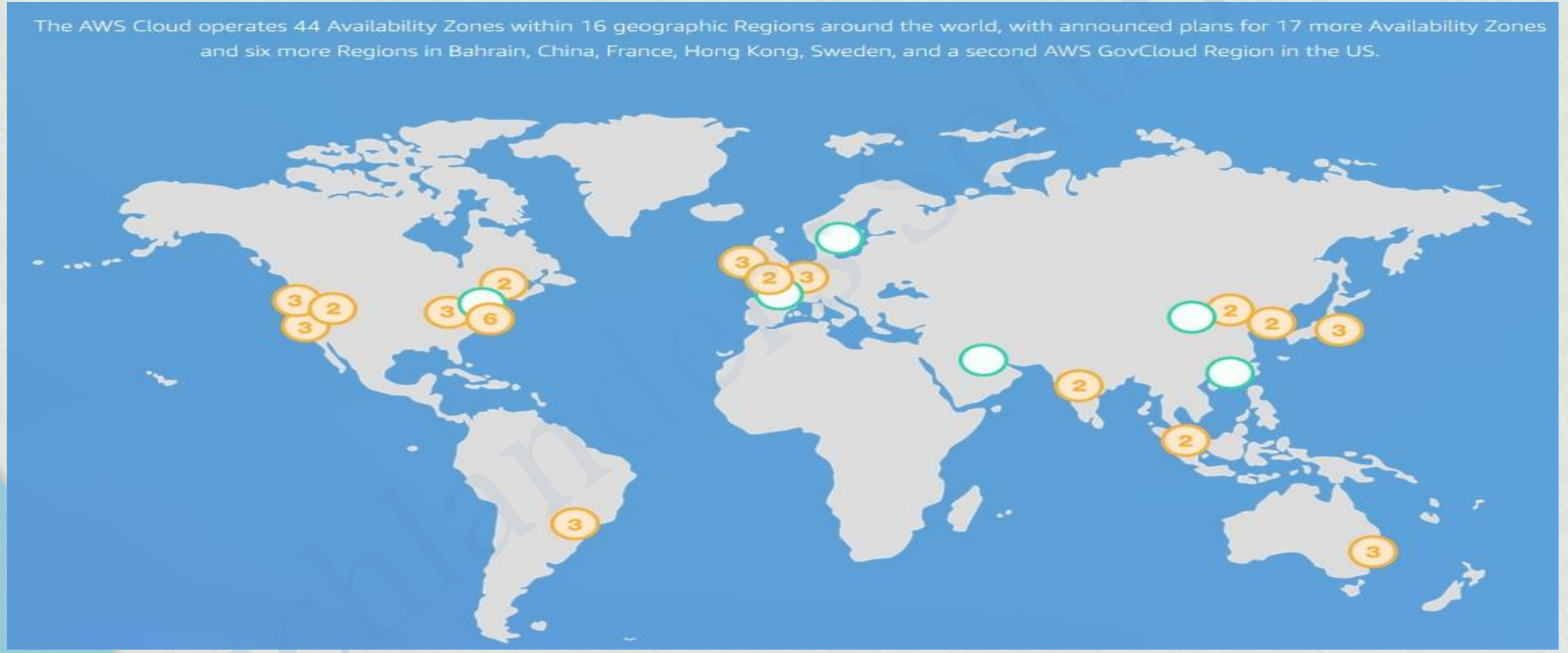


Note: Conceptual drawing only. The number of Availability Zones (AZ) may vary.

Shared Responsibility

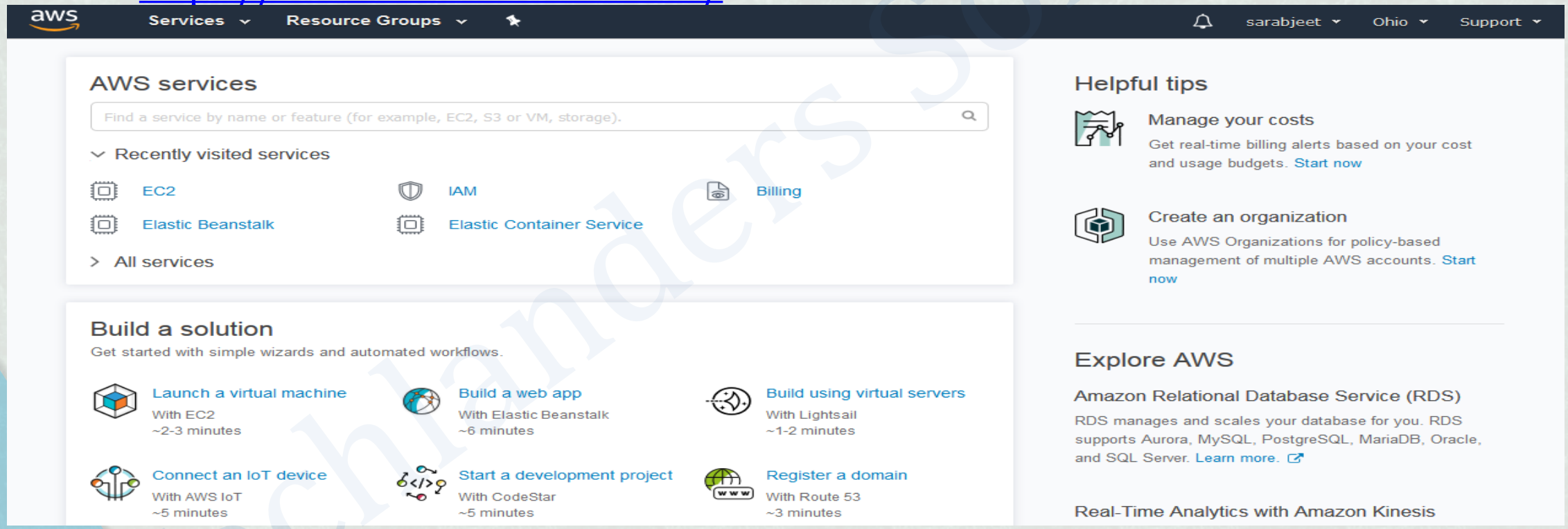


AWS Global Infrastructure



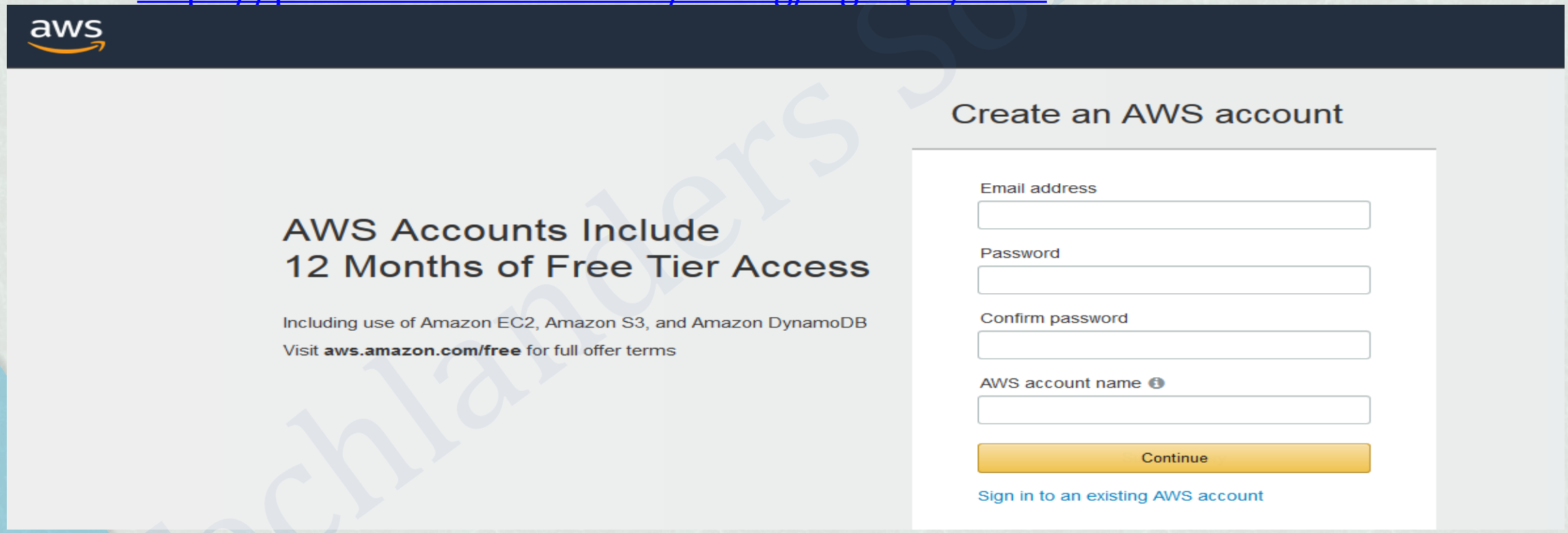
AWS Management Console

- Simple and intuitive web-based user interface.
  - https://console.aws.amazon.com/

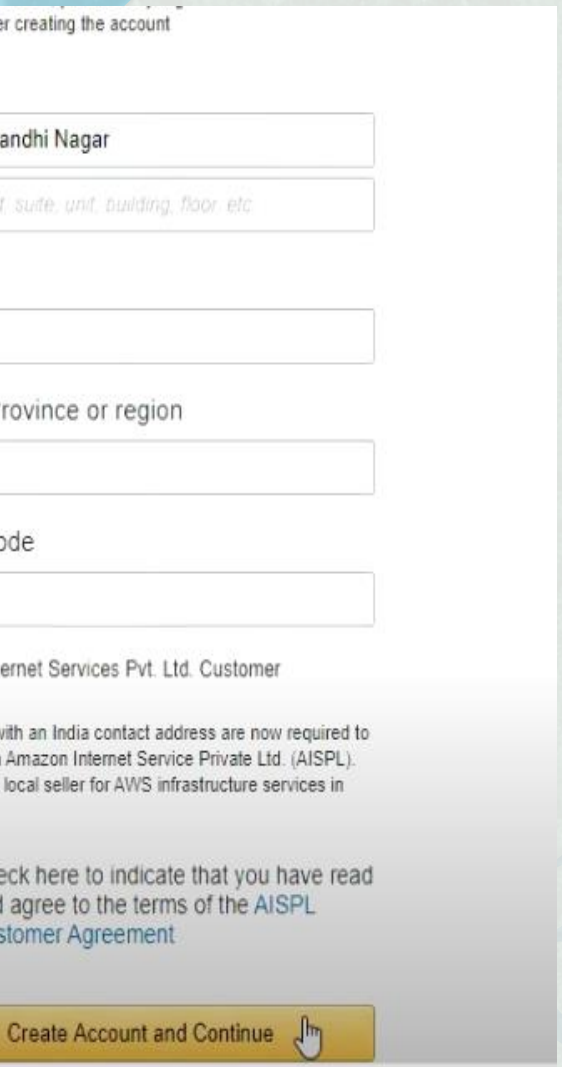


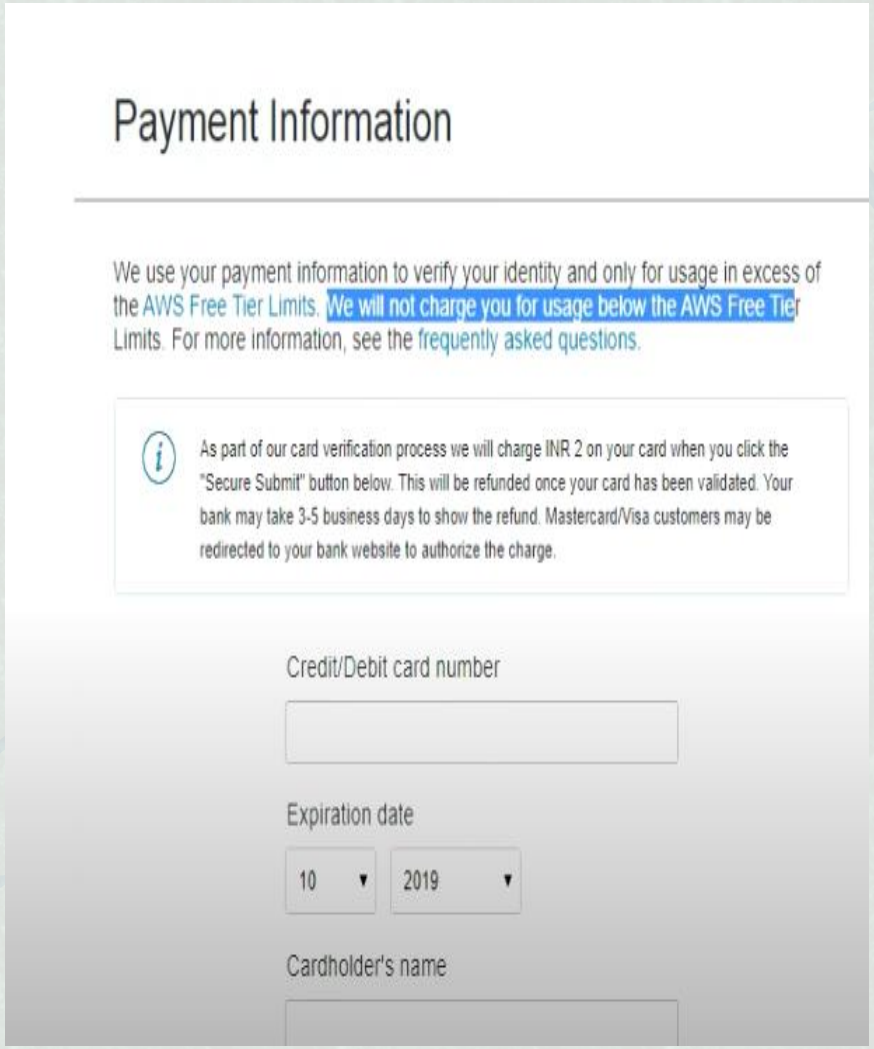
LAB 1 : AWS Signup

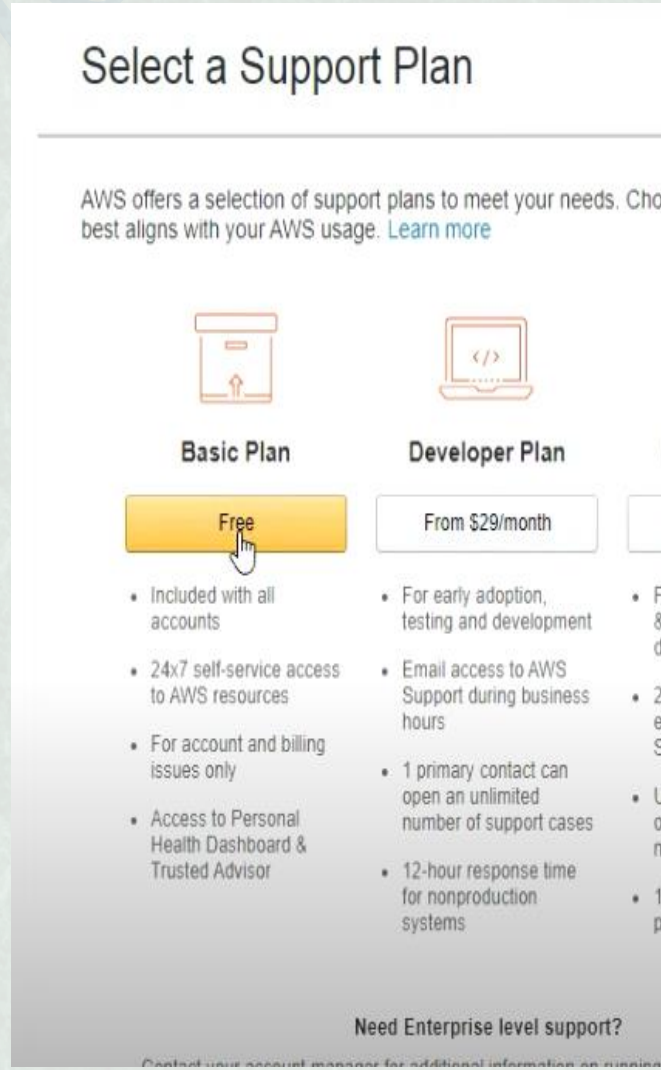
- Create a new account at
  - https://portal.aws.amazon.com/billing/signup#/start



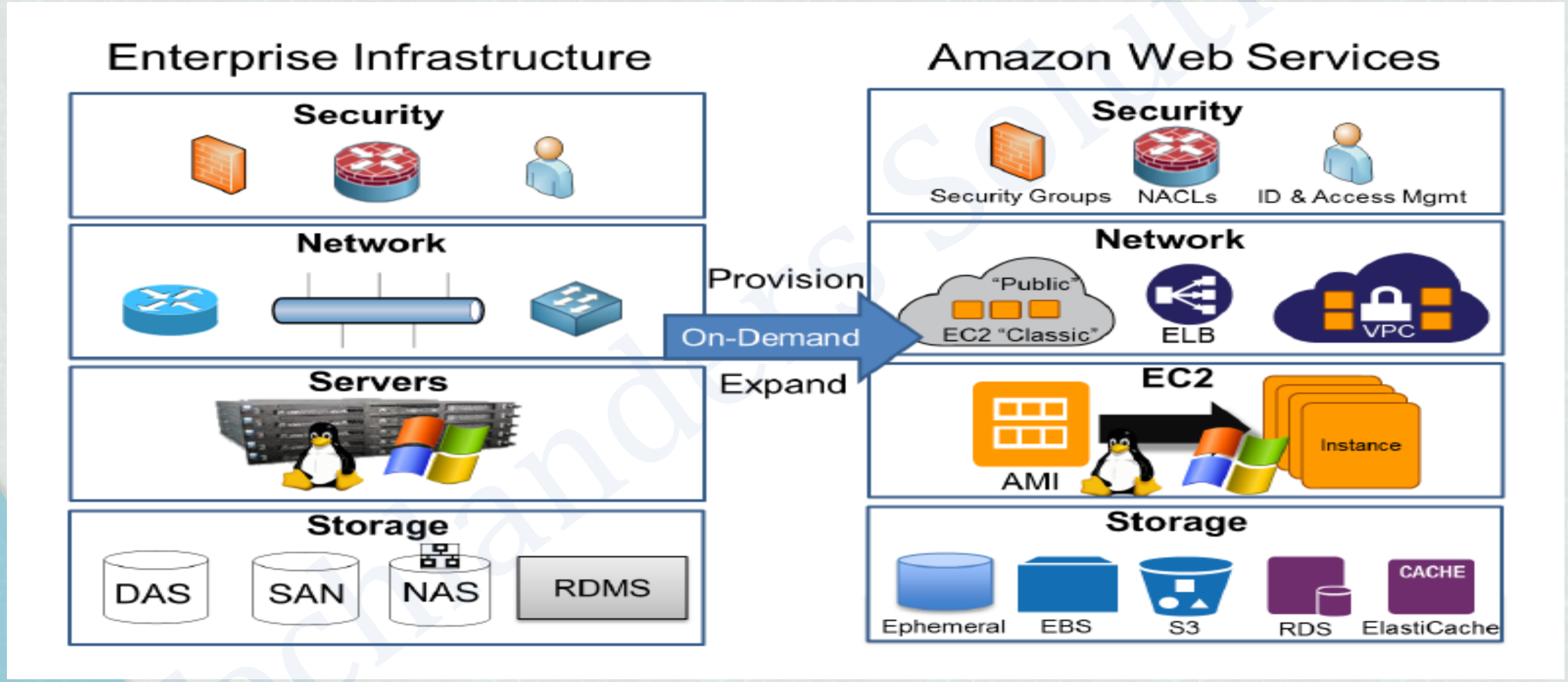
AWS SIGNUP







AWS Core Infrastructure Services



AWS Security

- **Physical Security:**
- 24/7 trained security staff
- AWS data centers in nondescript and undisclosed facilities
- Two-factor authentication for authorized staff
- Authorization for data center access
- Multiple approval based change process

AWS Security

- **Hardware, Software, and Network :**
- Authentication and authorization in place
- RBAC based access control mechanism
- Firewall and other boundary devices
- Security at Server level, Application level and Network level
- AWS monitoring tools
- Services to log AWS resources access

AWS Security

Amazon Resource Names (ARNs)

Amazon Resource Names (ARNs) uniquely identify AWS resources.

We require an ARN when you need to specify a resource unambiguously across all of AWS, such as in IAM policies, API calls etc.

ARN have a specific format:
arn:partition:service:region:account-id:resourcetype/resource

- IAM user name
  arn:aws:iam::123456789012:user/David
- IAM instance id:
  arn:aws:ec2:*region*:*account-id*:dedicated-host/*host_id*
  Eg. arn:aws:ec2:us-east-1:123456789012:dedicated-host/h-12345678

AWS Access Credentials

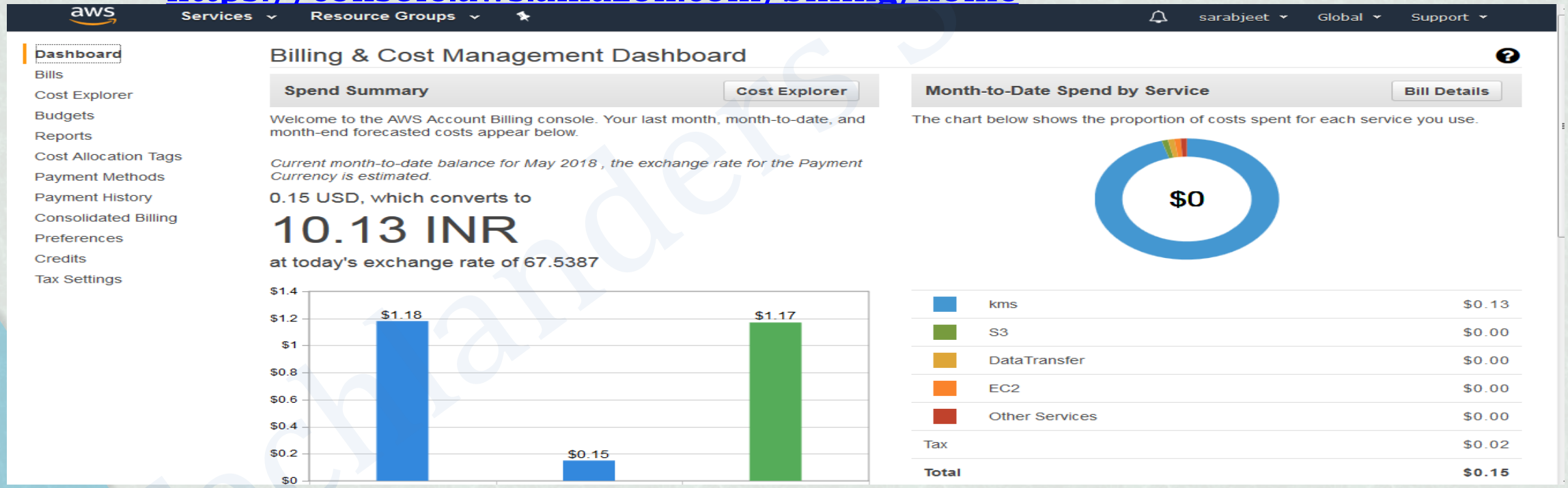
AWS resources can be access using several authentication methods:

- IAM User-id / Password
- Account ID/ AK (Access Key)/ SK (Security Key)
- Certificates
- Key pairs

AWS Billing

AWS Billings history, Previous payments, Current month cost, Budget fixing, Setting usage alarms etc, can be managed from AWS billing page :

https://console.aws.amazon.com/billing/home



AWS Pricing calculators

Simple Monthly Calculator:
You can Estimate your expected monthly bill using Simple Monthly Calculator.
http://calculator.s3.amazonaws.com/index.html

TCO Calculator:
You can Quickly compare the total cost of ownership (TCO) of your **on-premises infrastructure with a comparable AWS deployment** using TCO Calculator and estimate savings you can realize by moving to AWS. https://awstcocalculator.com/#

Cost Explorer:
With Cost Explorer, you can track your actual account usage and bill, at any time using the billing portal. You can view data for up to the last 13 months, forecast how much you are likely to spend for the next three months, and get recommendations for what Reserved Instances to purchase.
https://console.aws.amazon.com/billing/home#/costexplorer

Resource Management Tools

- **AWS Management Console**
- AWS Console Mobile App (View resources)
- **AWS Command line interface**
- AWS Toolkit for PowerShell
- AWS-Shell

AWS
Compute Services

Virtual Machines

- What are the bare minimum infrastructure requirements for any application?
- Hardware (RAM, Processor, Processor type, HBA's, etc.)
- OS Disk & Data Disk
- OS Images
- Network
- Security (Firewall, Networking, Access Mechanism)
- Credentials

AWS Elastic Compute Cloud

- Amazon **EC2** stands for Elastic Compute Cloud, and is the Primary AWS web service.
- Provides Resizable compute capacity
- **Reduces the time required** to obtain and boot new server instances to minutes
- There are two key concepts to Launch instances in AWS:
  - **Instance Type**
  - **AMI**

EC2 Facts:

- Scale capacity as your computing requirements change
- Pay only for capacity that you actually use
- Choose Linux or Windows OS as per need.
- **Deploy across AWS Regions and Availability Zones for reliability/HA**
- Only X86 based OS supported. So platform specific OS are not supported.

Instance Types (EC2)

- The instance type defines the different virtual hardware Models (sizes) supporting an Amazon EC2 instance.

- There are dozens of instance types available, varying in the following dimensions:

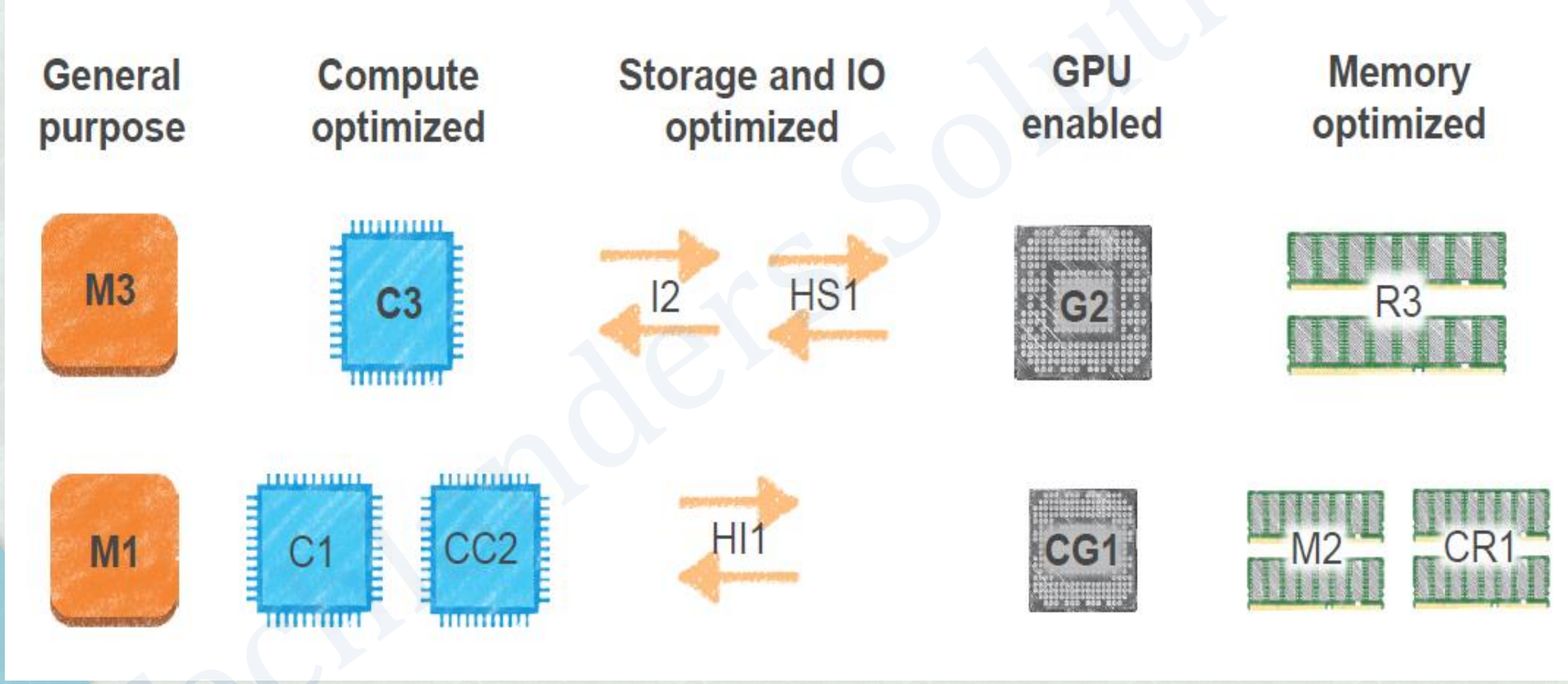
- Virtual CPUs (vCPUs)
- Memory
- Storage (size and type)
- Network performance
- Graphical Processing

- Instance types are grouped into families based on the ratio of these values to each other.

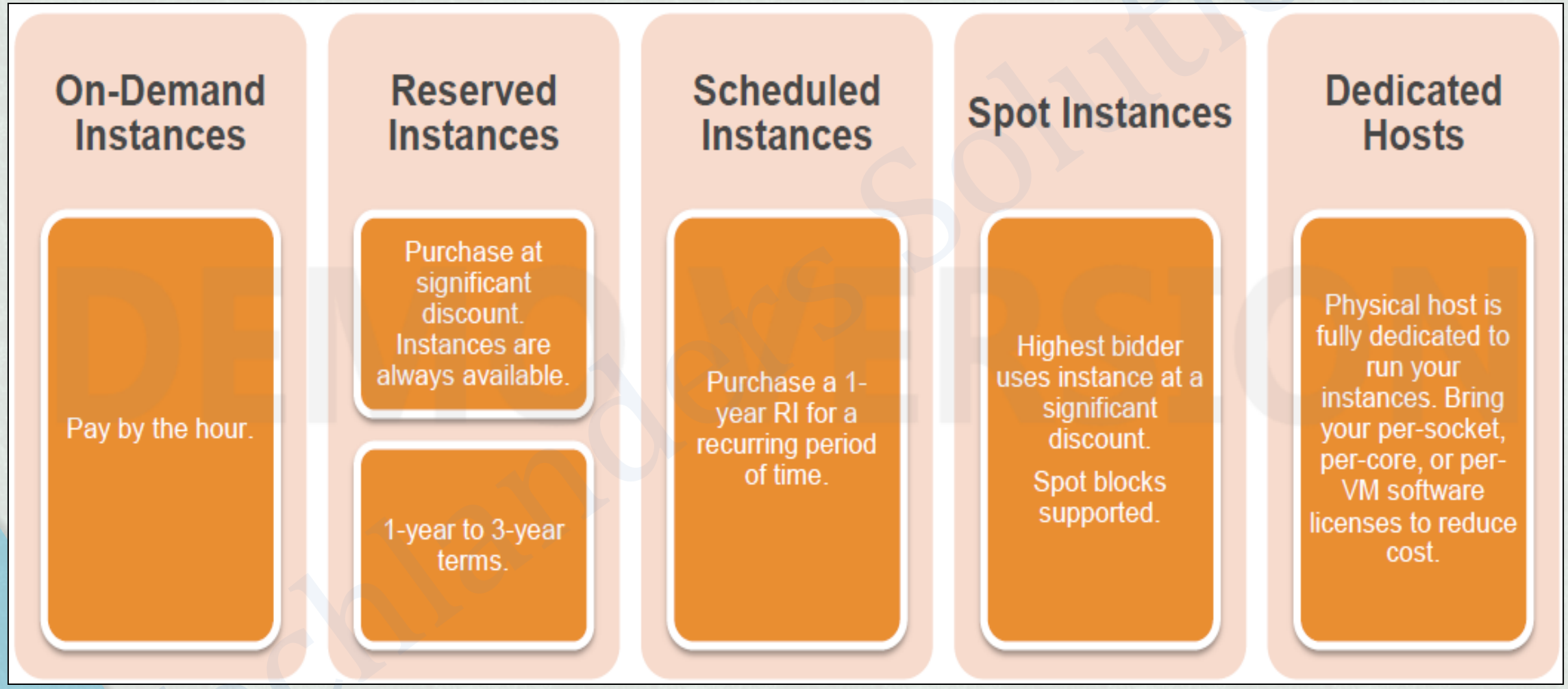
Instance Types (EC2)

- EC2 instance types are optimized for different use cases and come in multiple sizes. This allows you to optimally scale resources to your workload requirements.
- AWS uses Intel® Xeon® processors for EC2 instances, providing customers with high performance and value.
- Consider the following when choosing your instances: Core count, memory size, storage size and type, network performance, and CPU technologies.
- Hurry Up and Go Idle - A larger compute instance can save you time and money, therefore paying more per hour for a shorter amount of time can be less expensive.

Instance Types with AWS



EC2 Pricing Models



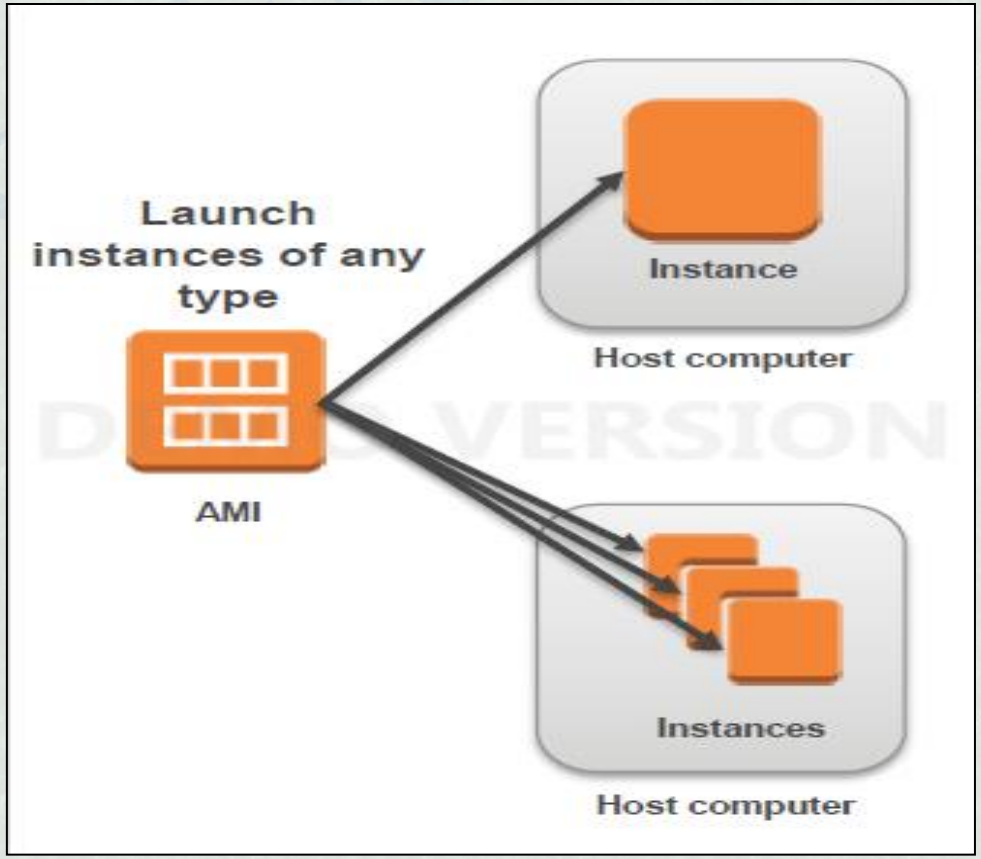
AMI's

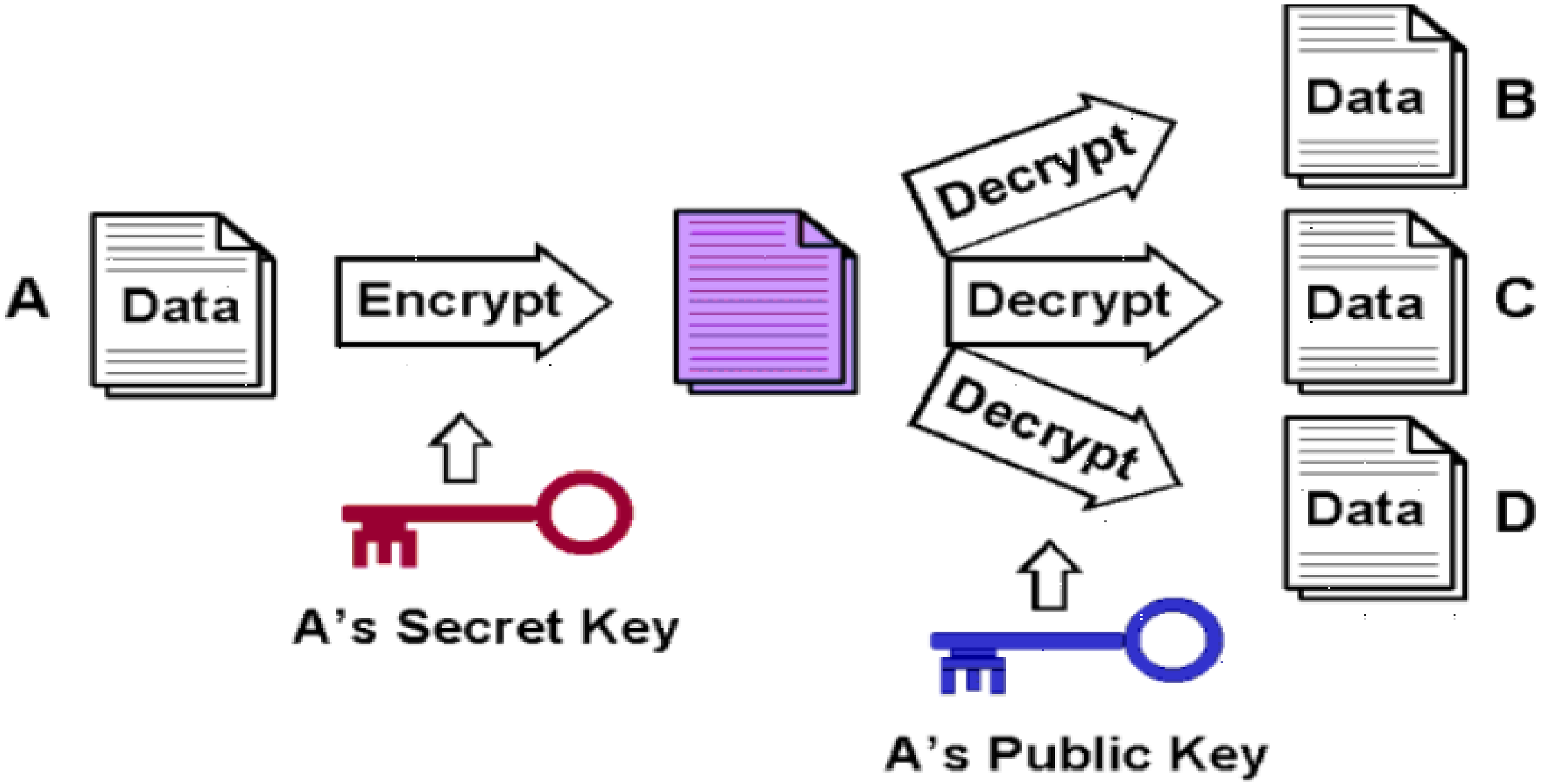
- Stands for **Amazon Machine Image**
- **AMI is a template for the root volume** for the instance (example: an OS image, a webserver, an application server etc.)
- It also includes the launch permissions that control which AWS accounts can use the AMI to launch instances.
- All AMIs are based on x86 OSs, either Linux or Windows.

Instances and AMI's

Select an AMI based on:

- Region
- Operating system
- Architecture (32-bit or 64-bit)
- Launch permissions
- Storage for the root device
- Virtualization Type




A
Data
Encrypt
A's Secret Key
Decrypt
Decrypt
Decrypt
A's Public Key
Data
B
Data
C
Data
D

Credentials

- Used to access an Instance.
- Amazon EC2 uses public-key cryptography to encrypt and decrypt login information.
- Public-key cryptography uses a public key to encrypt a piece of data and an associated private key to decrypt the data.
- These two keys together are called a *key pair*.
- AWS stores the public key, and the private key is kept by the customer. The private key is essential to acquiring secure access to an instance for the first time.
- When launching a Windows instance, Amazon EC2 generates a random password for the local administrator account and encrypts the password using the public key.

Launching EC2 Instance

- Procedure to launch EC2 instance in minutes:
- Determine the AWS Region in which you want to launch the Amazon EC2 instance.
- Launch an Amazon EC2 instance from a pre-configured Amazon Machine Image (AMI).
- Choose an instance type based on CPU, memory, storage, and network requirements.
- Configure network, security groups, storage volume, tags, and key pair; or directly launch instance post selecting AMI and Instance type.

LAB 3 : Create an EC2 Linux Instance

Console Access: https://console.aws.amazon.com/

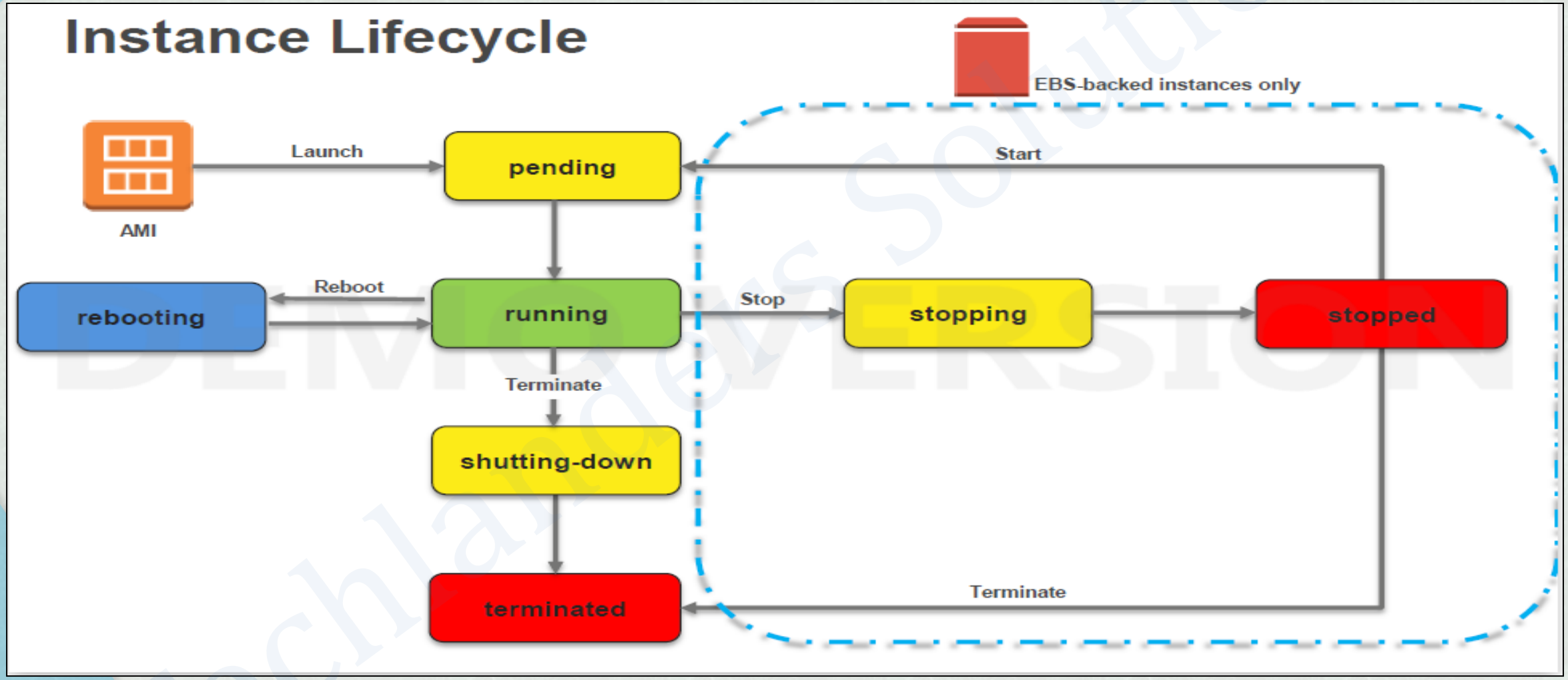
- Observe different available AMIs and Instance types
- Exercise:

  Create a Linux Operating System using mandatory configurations:
  - Instance Type : **T2.Micro**
  - AMI : **Amazon Linux 2 AMI**
  - Credentials: **Create/download your own key pair and use same**.

- Observe the instance details and navigate through the diff tabs
- Reboot, Stop, Start your server
- Access your instance using above downloaded key pair using different platforms i.e , instance connect , using putty , cli.

Note: (For the first instance take the default parameters wherever inputs required)

Instance Lifecycle



Network part

Private IP: Alike traditional VMs/hosts, each EC2 instance must at least belong to one network (VPC in AWS) and must have at least one Private IP from that network/subnet.

Public IP: A launched instance may also have a public IP address assigned. This IP address is assigned from the addresses reserved by AWS and cannot be specified. This IP address is unique on the Internet, persists only while the instance is running, and cannot be transferred to another instance.

Private/Public Domain Name System (DNS): When you launch an instance, AWS creates one private and one Public DNS name that can be used to access the instance. This DNS name is generated automatically and cannot be specified by the customer. This DNS name persists only while the instance is running and cannot be transferred to another instance.

Elastic IP (EIP): An elastic IP address is an address unique on the Internet that you reserve independently and associate with an Amazon EC2 instance.

EC2 Security – Security Group

Virtual Firewall Protection

AWS allows you to control traffic **in** and **out** of your instances through virtual firewalls called *security groups*.

Security groups allow you to control traffic based on *port*, *protocol*, and *source(inbound)/destination(outbound)*.

Security groups are associated with instances when they are launched. **Every instance must have at least one security group.** Though they can have more.

A security group is default deny.

Amazon Server Storage

Amazon EBS (Elastic Block Store)

- Data stored on an Amazon EBS volume can persist independently of the life of the instance.
- Persistent **block-level storage volumes** for use with Amazon EC2 instances.
- 99% used volume type in AWS.

Amazon EC2 Instance Store

- Like Swap volumes attached to your server for faster processing.
- Data stored on a local instance store persists only as long as the instance is alive.
- Storage is ephemeral.
- Older volume type, used exceptionally in some instances i.e DB,High processing systems.

AWS Tags

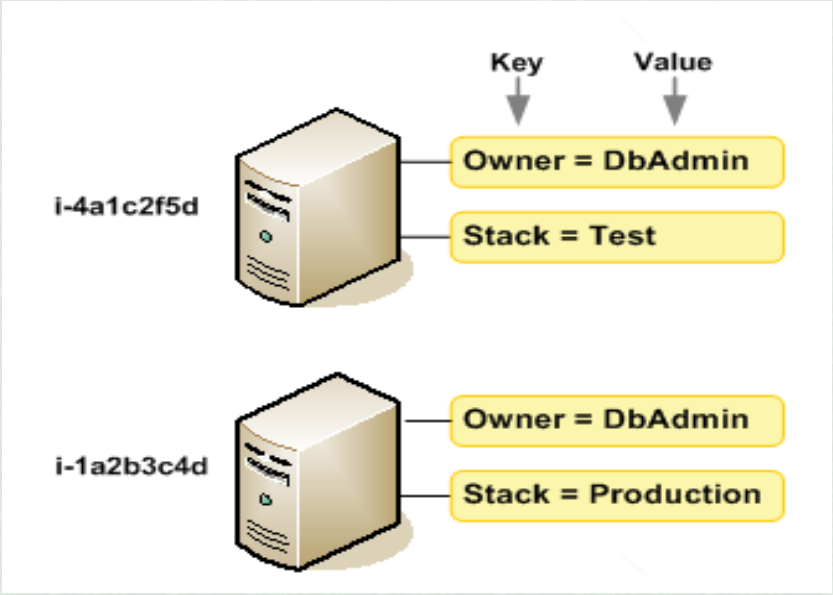
Tags helps you manage your instances, images, and other Amazon EC2 resources.

Its your own metadata to each resource in the form of *tags*.

Each tag consists of a *key* and an optional *value*, both of which you define.

It enable you to categorize your AWS resources in different ways, for example, by purpose, owner, or environment.

You can organize your billing information based on resources that have the same tag key values.



AWS Tags

- Maximum number of tags per resource—50
- Tag keys and values are case-sensitive.
- You can't terminate, stop, or delete a resource based solely on its tags; you must specify the resource identifier : ARN’s
- Not all resources supports tags. Pls check the supported list from AWS website.

EC2 Metadata

AWS Instance Metadata –

Instance metadata is data/information about your instance.

An HTTP call to **http://169.254.169.254/latest/meta-data/** will return the top node of the instance metadata tree and will return the information about your instance e.g. Public IP, DNS name, OS type etc..

For Linux systems:
curl http://169.254.169.254/latest/meta-data/

Dynamic Instance Identity metadata:
http://169.254.169.254/latest/dynamic/instance-identity/

LAB 4 : Create Windows Instance

Exercise:

- Create a Windows Operating System using custom configurations:
  - Instance Type : **T2.Micro**
  - AMI : **Microsoft Windows Server 2019 Base**
  - Credentials: **Use existing key from Lab 3 or if not create a new one.**
  - Assign tags, **Key = Name, Value= Server-Name**
  - Assign tags, **Key = Owner, Value= Your name**
  - Add inbound port of **RDP**
- Access your instance using Administrator password, which will be retrieved using your downloaded key pair using mstsc or dns

Version Control Systems with GIT

What is Version Control System

As name states Version Control System is the **“Management of changes to anything”.**

Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

Think about traditional versioning of file with names – Login001.java, Login002.java, Login_final.java.

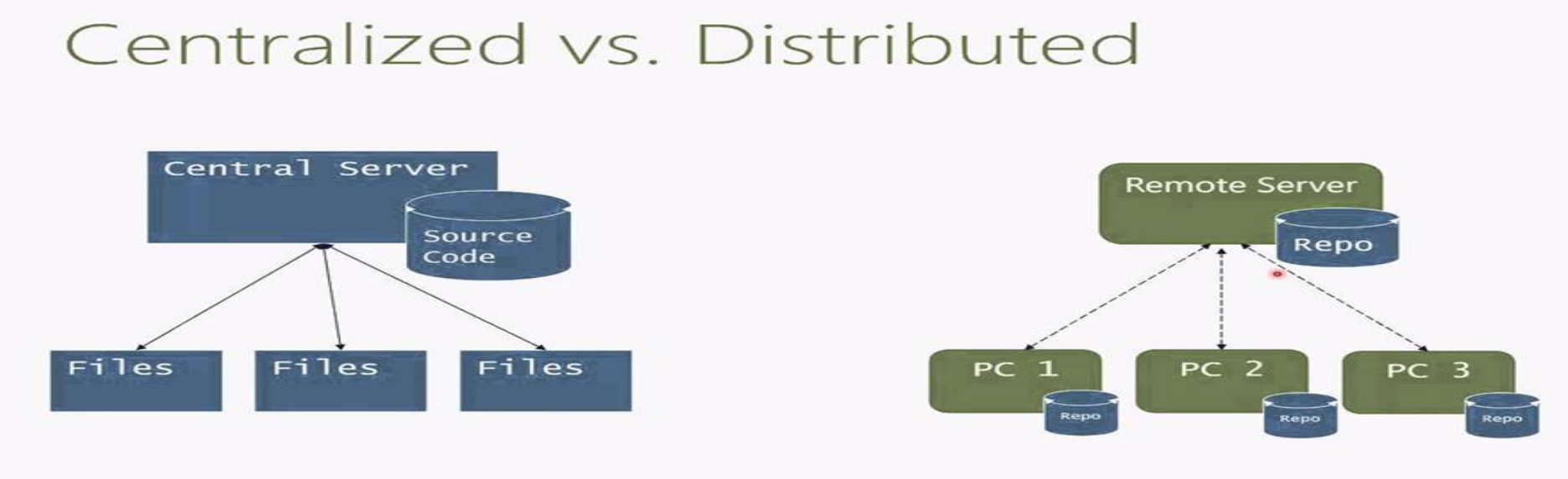
Its not just for code, it also helps in

- Backups & Restoration
- Synchronization
- Reverts
- Track Changes
- Most importantly in Parallel Development

Types of VCS

Majorly VCS is divided into two parts:

- Centralized Version Control System – CVS, Subversion, Visual Source Safe
- Distributed Version Control System – Mercurial, Bitkeeper, Git



Git History

Linus uses BitKeeper to manage Linux code

Ran into BitKeeper licensing issue

Liked functionality

Looked at CVS as how not to do things

April 5, 2005 - Linus sends out email showing first version

June 15, 2005 - Git used for Linux version control

Why Git?

- **Branching:** gives developers a great flexibility to work on a replica of master branch.

- **Distributed Architecture:** The main advantage of DVCS is **“no requirement of network connections to central repository”** while development of a product.

- **Open-Source:** Free to use.

- **Integration with CI:** Gives faster product life cycle with even faster minor changes.

Git Installation

- yum install autoconf libcurl-devel expat-devel gcc gettext-devel kernel-headers openssl-devel perl-devel zlib-devel –y

- Visit git release page - https://github.com/git/git/releases and pick desired version.

- curl -O -L https://github.com/git/git/archive/v2.14.0.tar.gz

- tar -zxvf v2.14.2.tar.gz

- cd git-v2.14.0
- make clean
- make configure
- ./configure --prefix=/usr/local
- make

- make install
- ln -s /usr/local/bin/git /usr/bin/git

Git Commands

- git –version

- [root@techlanders ~]# git config --global user.email “Gagandeep.singh@techlanders.com”
- [root@techlanders ~]# git config --global user.name “Gagandeep Singh”
- [root@techlanders ~]# git config --global -l
- user.name=Gagandeep Singh
- user.email=Gagandeep.singh@techlanders.com
- [root@techlanders ~]#

- //Initializing a repo
- [root@master git]# mkdir /Repo1
- [root@master git]# cd /Repo1/
- [root@master Repo1]# git init
- Initialized empty Git repository in /Repo1/.git/
- [root@master Repo1]#

Github/Bitbucket

- **What is Bitbucket /GitHub/Gitlab?.**

- Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.

- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

- **Host in the cloud**: free for small teams (till 5 users) and paid for larger teams.

- **Host on Your server**: One-Time pay for most solutions.

- Visit "https://bitbucket.org/" and click "Get Started" to sign up for free account.
- Visit "https://github.com/" for Github details

Version Control Systems with GIT

Raman Khanna

What is Version Control System

- As name states Version Control System is the **“Management of changes to anything”.**

- Version Control is way of storing files in central location accessible to all team members and enabling them to keep track of changes being done in the source code by whom, when & why. It also help teams to recover from some inevitable circumstances.

A History Lesson

- Before Version Control System
  - File renaming
    - Login001.java
    - Login002.java
    - LoginFinal.java
  - Directories
    - June_Release_Code
    - August_Release_Code
  - Zip Files
    - Package_May.zip
    - Package_June.zip
  - Nothing at all

Version Control helps in

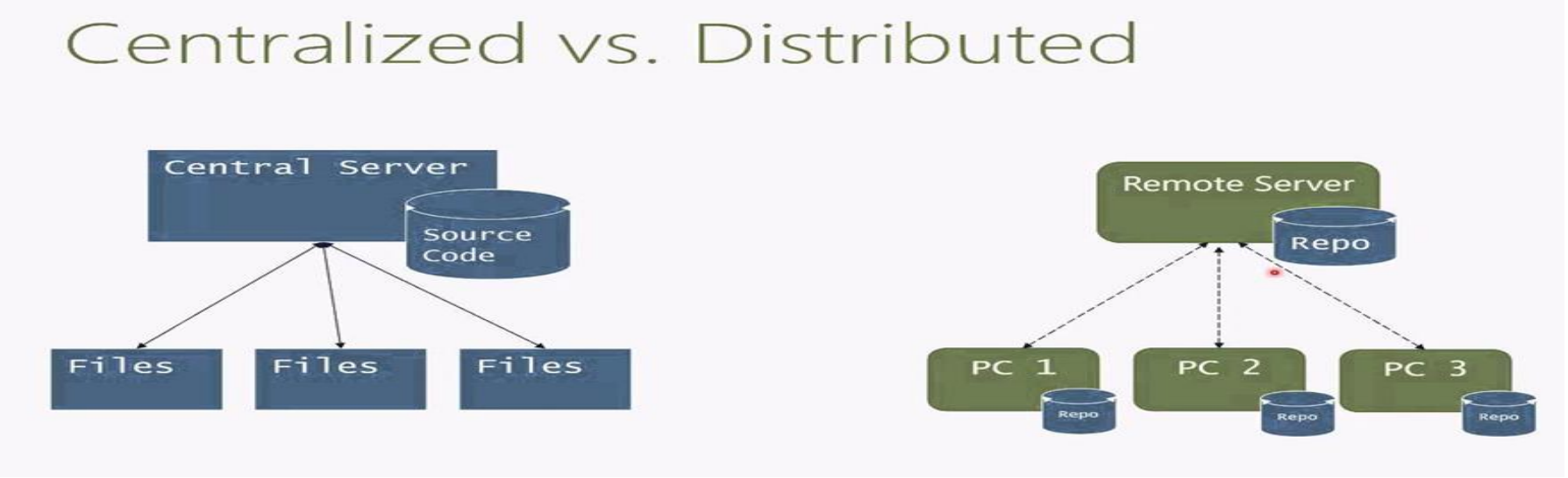
- Its not just for code, it also helps in
    - Backups & Restoration
    - Synchronization
    - Reverts
    - Track Changes
    - Most importantly in Parallel Development

Types of Version Control Systems

- Majorly VCS is divided into two parts:

  - Centralized Version Control System

  - Distributed Version Control System



Centralized Version Control System

- Traditional version control system
    - Server with database
    - Clients have a working version
- Examples
    - CVS
    - Subversion
    - Visual Source Safe
- Challenges
    - Multi-developer conflicts
    - Client/server communication

Distributed Version Control Syetem

- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Other distributed systems include
    - Mercurial
    - BitKeeper
    - Darcs
    - Bazaar

GIT History

- Linus uses BitKeeper to manage Linux code

- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things

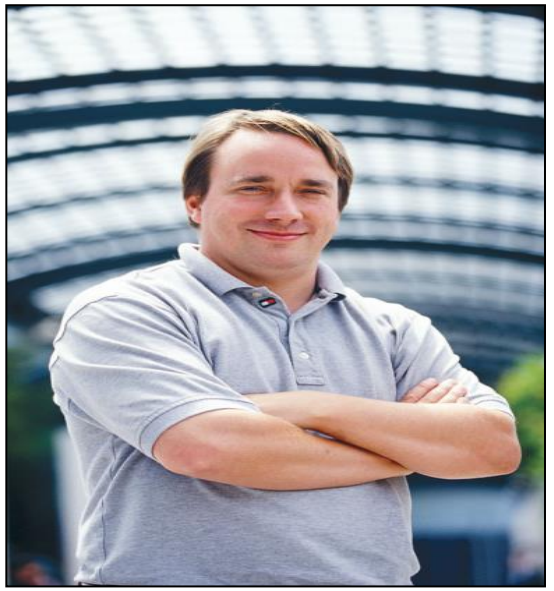
- April 5, 2005 - Linus sends out email showing first version

- June 15, 2005 - Git used for Linux version control

GIT is not a SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable file system, used to track directory trees.

Linus Torvalds, 7 Apr 2005



Why GIT

- **Why Git:**

- **Branching:** gives developers a great flexibility to work on a replica of master branch.

- **Distributed Architecture:** The main advantage of DVCS is **“no requirement of network connections to central repository”** while development of a product.

- **Open-Source:** Free to use.

- **Integration with CI:** Gives faster product life cycle with even more faster minor changes.

GIT Installation

- GIT comes as default offering for all major Linux flavors
- Though you can download the installers from below link for Windows, Mac OS X, Linux, Solaris
  - https://git-scm.com/downloads
  - you can install same with yum too:
    - yum install git
- Installing GIT Bash on Windows

GIT Installation on Linux

- yum install autoconf libcurl-devel expat-devel gcc gettext-devel kernel-headers openssl-devel perl-devel zlib-devel –y

- Visit git release page - https://github.com/git/git/releases and pick desired version.

- curl -O -L https://github.com/git/git/archive/v2.14.0.tar.gz

- tar -zxvf v2.14.2.tar.gz

- cd git-v2.14.0
- make clean
- make configure
- ./configure --prefix=/usr/local
- make

- make install
- rm –rf /usr/bin/git
- ln -s /usr/local/bin/git /usr/bin/git

GIT VERSION

```
[root@user20-master plays]# git --version
git version 1.8.3.1
[root@user20-master plays]#
```

GIT HELP

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Setting Identity in GIT

```
[root@Techlanders ~]# git config --global  user.email “raman.khanna@Techlanders.com”
[root@Techlanders ~]# git config --global  user.name “raman.khanna”
[root@Techlanders ~]# git config --global -l
user.name=Gagandeep Singh
user.email=Gagandeep.singh@Techlanders.com
[root@Techlanders ~]#
```

GIT Repository

- A **repository** is usually used to organize a single project.
- Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs.
- Repository is like a unique shared file system for a project.

Initializing a Repository

```
[root@master git]# mkdir /Repo1
[root@master git]# cd /Repo1/
[root@master Repo1]# git init
Initialized empty Git repository in /Repo1/.git/
[root@master Repo1]# ls -lrt /Repo1/.git/
total 12
drwxr-xr-x. 4 root root  31 Dec 19 19:32 refs
drwxr-xr-x. 2 root root  21 Dec 19 19:32 info
drwxr-xr-x. 2 root root 242 Dec 19 19:32 hooks
-rw-r--r--. 1 root root  73 Dec 19 19:32 description
drwxr-xr-x. 2 root root   6 Dec 19 19:32 branches
drwxr-xr-x. 4 root root  30 Dec 19 19:32 objects
-rw-r--r--. 1 root root  23 Dec 19 19:32 HEAD
-rw-r--r--. 1 root root  92 Dec 19 19:32 config
[root@master Repo1]#
```

Adding file to a Repository

```
[root@master Repo1]# git status
# On branch master
# Initial commit - nothing to commit (create/copy files and use "git add" to track)
[root@master Repo1]# echo "File1 content" >> file1
[root@master Repo1]# ll
-rw-r--r--. 1 root root 14 Dec 19 19:35 file1
[root@master Repo1]# git add file1
[root@master Repo1]# git status
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:   file1
#
[root@master Repo1]#
```

Checking Repository Status

```
[root@user20-master Repo1]# touch file2
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@user20-master Repo1]# git add --all
[root@user20-master Repo1]# git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   file1
#       new file:   file2
#
[root@user20-master Repo1]# git commit -m "Commit one - Added File1 & File2"
[master (root-commit) 9c301cb] Commit one - Added File1 & File2
 2 files changed, 1 insertion(+)
 create mode 100644 file1
 create mode 100644 file2
[root@user20-master Repo1]#
```

Committing changes to a Repository

```
[root@master Repo1]#echo "File2 content added" > file2
[root@master Repo1]#git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   file2
#
no changes added to commit (use "git add" and/or "git commit -a")
[root@master Repo1]#git commit -am "second commit - Changes done in File2"
[master 77de849] second commit - Changes done in File2
 1 file changed, 1 insertion(+)
[root@master Repo1]#
```

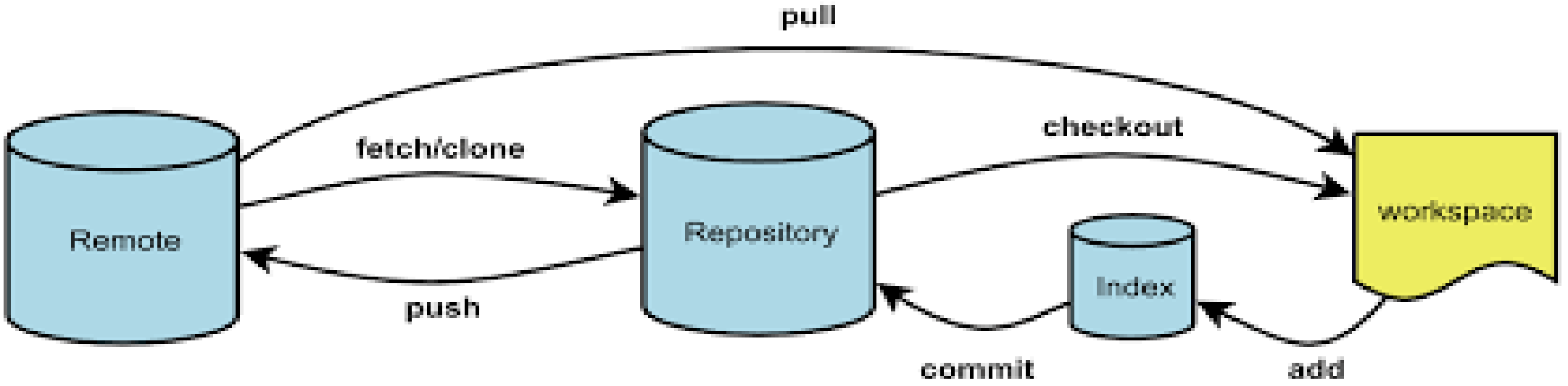
Git Log

```
[root@master Repo1]#git log
commit 77de8496c39c8d442d8e1212f9f3879a33253a1c
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date:   Wed Dec 19 19:48:56 2018 +0000

    second commit - Changes done in File2

commit 9c301cb93733f666e959a87c7a3f61142d1d9f48
Author: admin.gagan@gmail.com <admin.gagan@gmail.com>
Date:   Wed Dec 19 19:43:25 2018 +0000

    Commit one - Added File1 & File2
[root@master Repo1]#
```

Git Diff- Comparing two commits

```
[root@master Repo1]#git diff 9c301cb93733f666e959a87c7a3f61142d1d9f48
77de8496c39c8d442d8e1212f9f3879a33253a1c
diff --git a/file2 b/file2
index e69de29..de51b99 100644
--- a/file2
+++ b/file2
@@ -0,0 +1 @@
+File2 content added
[root@master Repo1]#
```

Git Flow



Git Push

```
[root@master Repo1]#git push origin master
Username for 'https://github.com': admin.gagan@gmail.com
Password for 'https://admin.gagan@gmail.com@github.com':
Counting objects: 14, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.22 KiB | 0 bytes/s, done.
Total 12 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/admingagan/ansibleplaybooks.git
   d9b5ae7..64bad4d  master -> master
[root@master Repo1]#
```

Initializing a Remote Repository

```
[root@master Repo1]#git remote add origin https://github.com/admingagan/repo1.git
[root@master Repo1]#
[root@master Repo1]#git pull origin master
From https://github.com/admingagan/repo1
 * branch            master     -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 README.md |  1 +
 abc       |  0
 ntp.yaml  | 13 +++++++++++++
 3 files changed, 14 insertions(+)
 create mode 100644 README.md
 create mode 100644 abc
 create mode 100644 ntp.yaml
[root@master Repo1]#
```

Git Pull

```
[root@master Repo1]#ls
abc  file1  file2  ntp.yaml  readme5 README.md

[root@master Repo1]#git pull origin dev
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/admingagan/ansibleplaybooks
 * branch            dev        -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 readme6 | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 readme6
[root@master Repo1]#ls
abc  file1  file2  ntp.yaml  readme5  readme6  README.md
[root@master Repo1]#
```

Git Clone

```
[root@user20-master git]# git clone https://github.com/admingagan/test.git
Cloning into 'test'...
remote: Enumerating objects: 23, done.
remote: Counting objects: 100% (23/23), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 23 (delta 5), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (23/23), done.
[root@user20-master git]# cd test
[root@user20-master test]# ll
total 16
-rw-r--r--. 1 root root 10 Dec 20 07:45 Readme
-rw-r--r--. 1 root root 24 Dec 20 07:45 Readme2
-rw-r--r--. 1 root root 57 Dec 20 07:45 readme3
-rw-r--r--. 1 root root  9 Dec 20 07:45 README4
[root@user20-master test]#
```

GIT BRANCH

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch training

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
  training
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch -d training
Deleted branch training (was 74e76df).

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git branch
* master
```

GIT CHECKOUT

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git checkout training
Switched to branch 'training'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training)
$ git checkout -b training_checkout
Switched to a new branch 'training_checkout'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git branch
  master
  training
* training_checkout
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git log
commit 76e137f900eb33b7eb4a4ac7c81f160af8124697 (HEAD -> training_checkout)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Fri Jun 9 20:38:59 2017 +0530

    commit in branch

commit 74e76dfcaf297ae9b63f950988907cdce8d275de (training, master)
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:16:16 2017 +0530

    second commit

commit 3eaadebb9972b29b7463822e99b485b9d9b490eb
Author: Kulbhushan Mayer <kulbhushan.mayer@thinknyx.com>
Date:   Thu Jun 8 23:14:06 2017 +0530

    my initial commit
```

GIT MERGE

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (training_checkout)
$ git checkout master
Switched to branch 'master'

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training_checkout
Updating 74e76df..76e137f
Fast-forward
 file.txt | 2 ++
 1 file changed, 2 insertions(+)
```

- Run git checkout master
- Run git merge training

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master)
$ git merge training
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.

kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ ls -ltr
total 2
-rw-r--r-- 1 kmayer 197121  30 Jun  9 20:51 file-in-training-branch.txt
-rw-r--r-- 1 kmayer 197121 238 Jun 12 20:47 file.txt
```

```
kmayer@mayer MINGW64 ~/thinknyx-repositories/repository-1 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   file.txt
```

GIT MERGE – Resolving Conflicts

```
1   Adding dummy data - Kulbhushan Mayer
2   Second Update
3
4   <<<<<<< HEAD
5   Making changes in branch for demo
6
7   Making one more change - 09/06/2017 08:44
8
9   Making change at 08:46 PM
10  =======
11  Making change to check merging - 12/06/2017
12  >>>>>>> training
13
```

- <<<<<<< depicts changes from the HEAD or BASE branch
- ======= divides your changes form the other branch
- >>>>>>> depicts the end of changes
- Remove <<<<<<<, =======, >>>>>>> from the file and make then necessary changes
- Now you have to commit the changes explictly


```
1   Adding dummy data - Kulbhushan Mayer
2   Second Update
3
4
5   Making changes in branch for demo
6
7   Making one more change - 09/06/2017 08:44
8
9   Making change at 08:46 PM
10
11  Making change to check merging - 12/06/2017
12
```

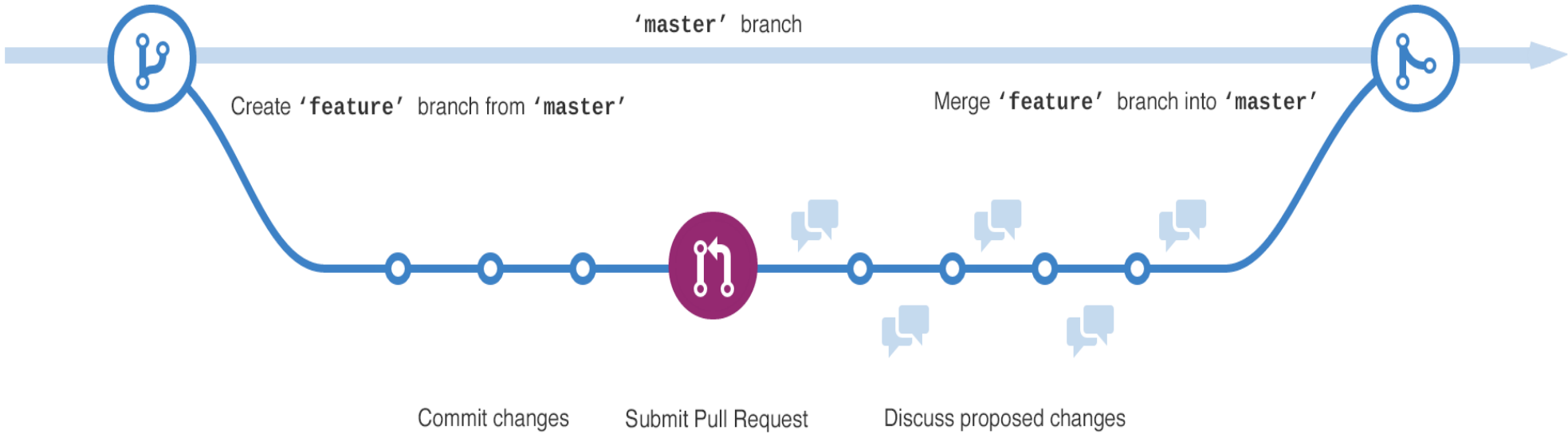
GIT Branch

- Branching is the way to work on different versions of a repository at one time.
- By default your repository has one branch named master which is considered to be the definitive branch.
- We use branches to experiment and make edits before committing them to master.
- When you create a branch off the master branch, you're making a copy, or snapshot, of master as it was at that point in time.
- If someone else made changes to the master branch while you were working on your branch, you could pull in those updates.

GIT Branch

Here you have:

- The master branch
- A new branch called feature (because we'll be doing 'feature work' on this branch)
- The journey that feature takes before it's merged into master



BitBucket/GitHub/GitLab

- **What is Bitbucket /GitHub/Gitlab?.**

- Bitbucket is a Git solution for professional teams. In simple layman language its a UI for Git, offered by Atlassian, similarly we have different available UI solutions from Github (most famous) and Gitlab.

- GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere.

- **Host in the cloud**: free for small teams (till 5 users) and paid for larger teams.

- **Host on Your server**: One-Time pay for most solutions.

- Visit "https://bitbucket.org/" and click "Get Started" to sign up for free account.
- Visit "https://github.com/" for Github details