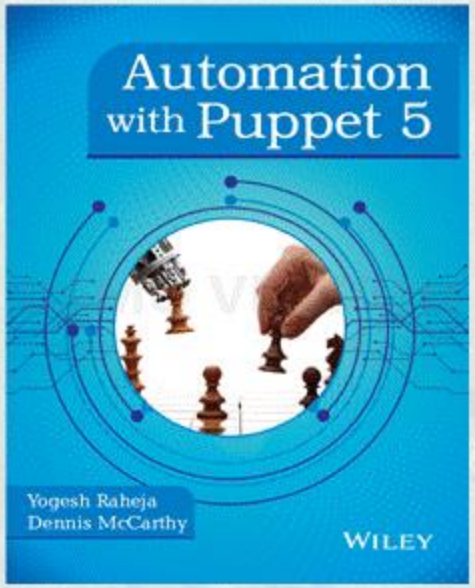
DevOps – A Culture

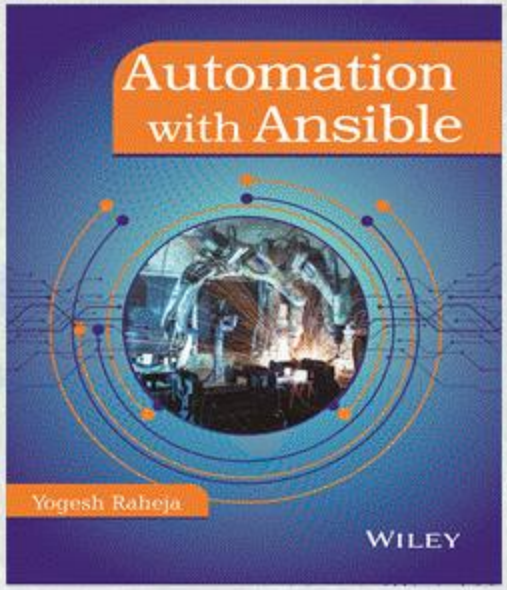
THINKNYX
TECHNOLOGIES

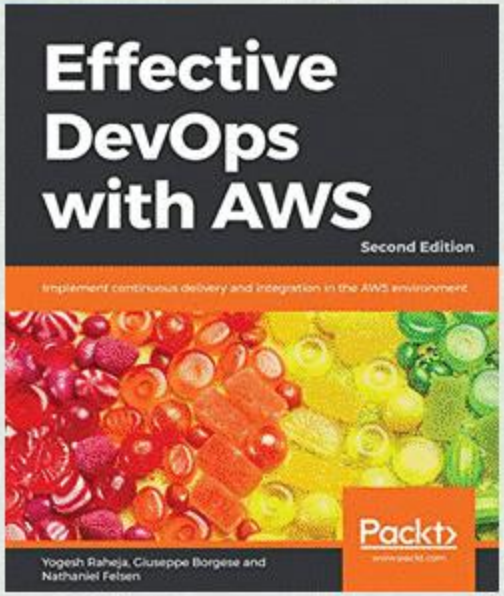
Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

WHO AM I

Thinknyx







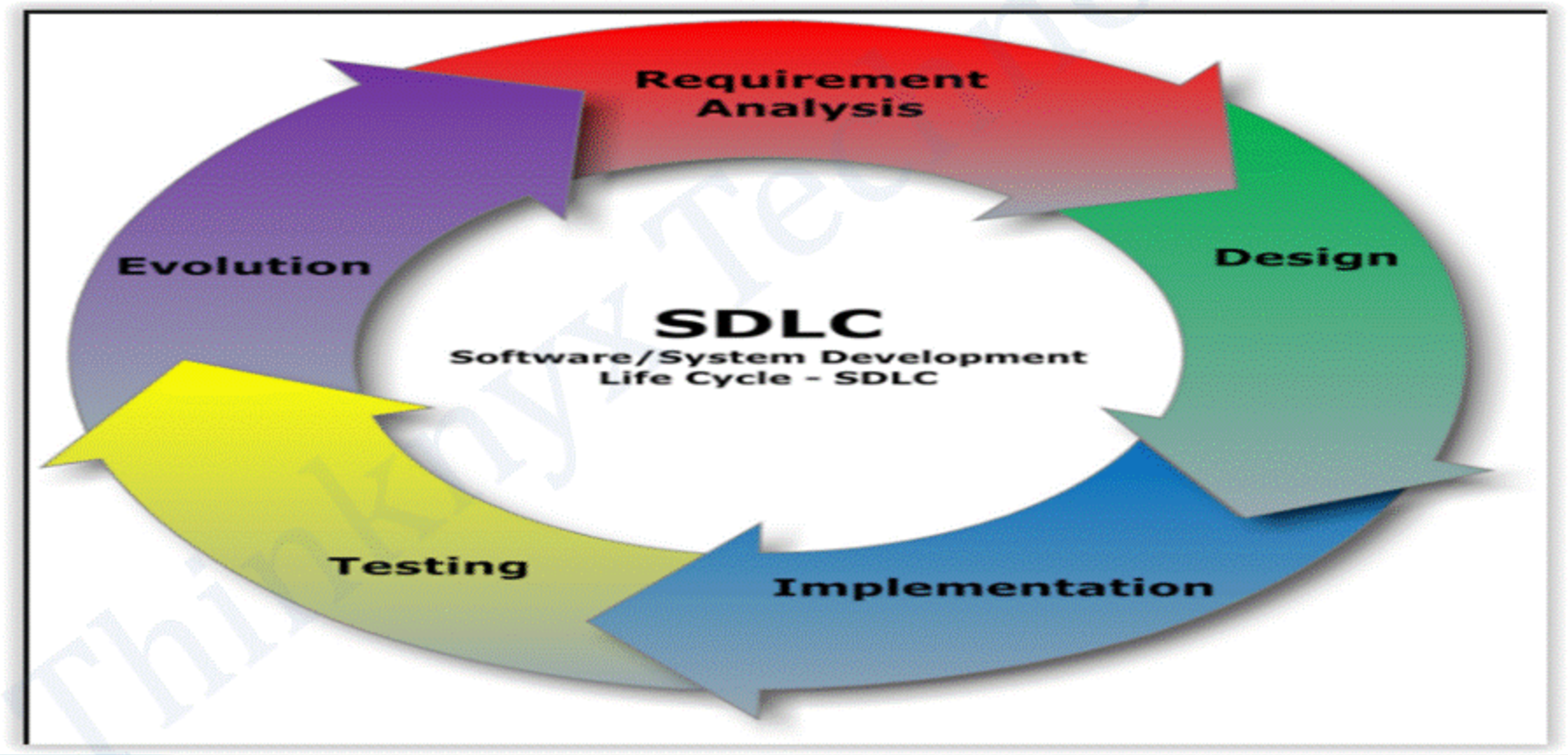
YOGESH RAHEJA

Agenda

- History of Software Development
- What is DevOps?
- Why DevOps?
- DevOps Skillset
- DevOps Qualities

Tradition SDLC Model

- A systems development life cycle is composed of **a number of 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



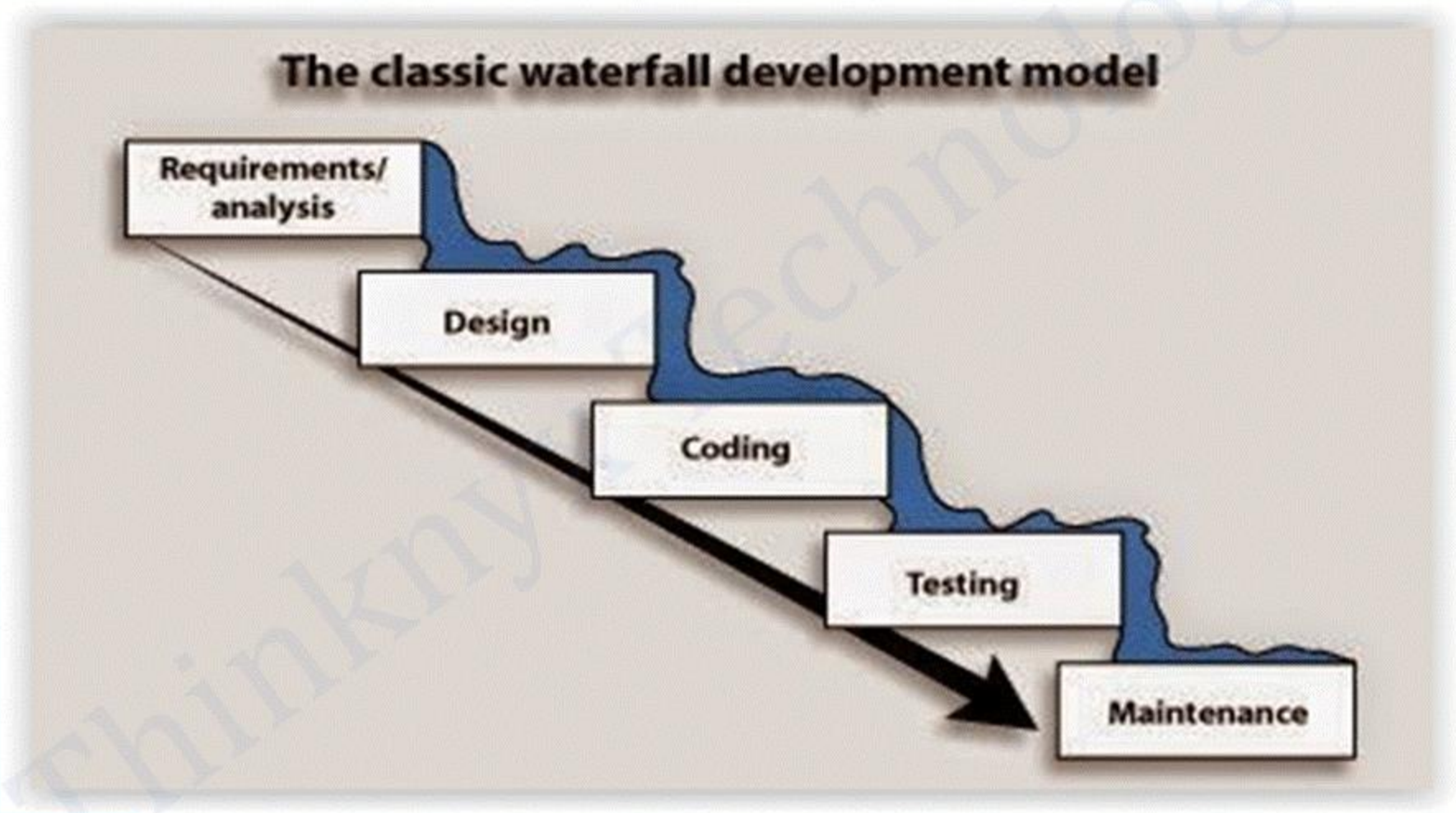
SDLC Continued ...

- There are multiple methodologies developed and evolved in Software development vertical for SDLC phases, all methodologies follow SDLC but with different methods:

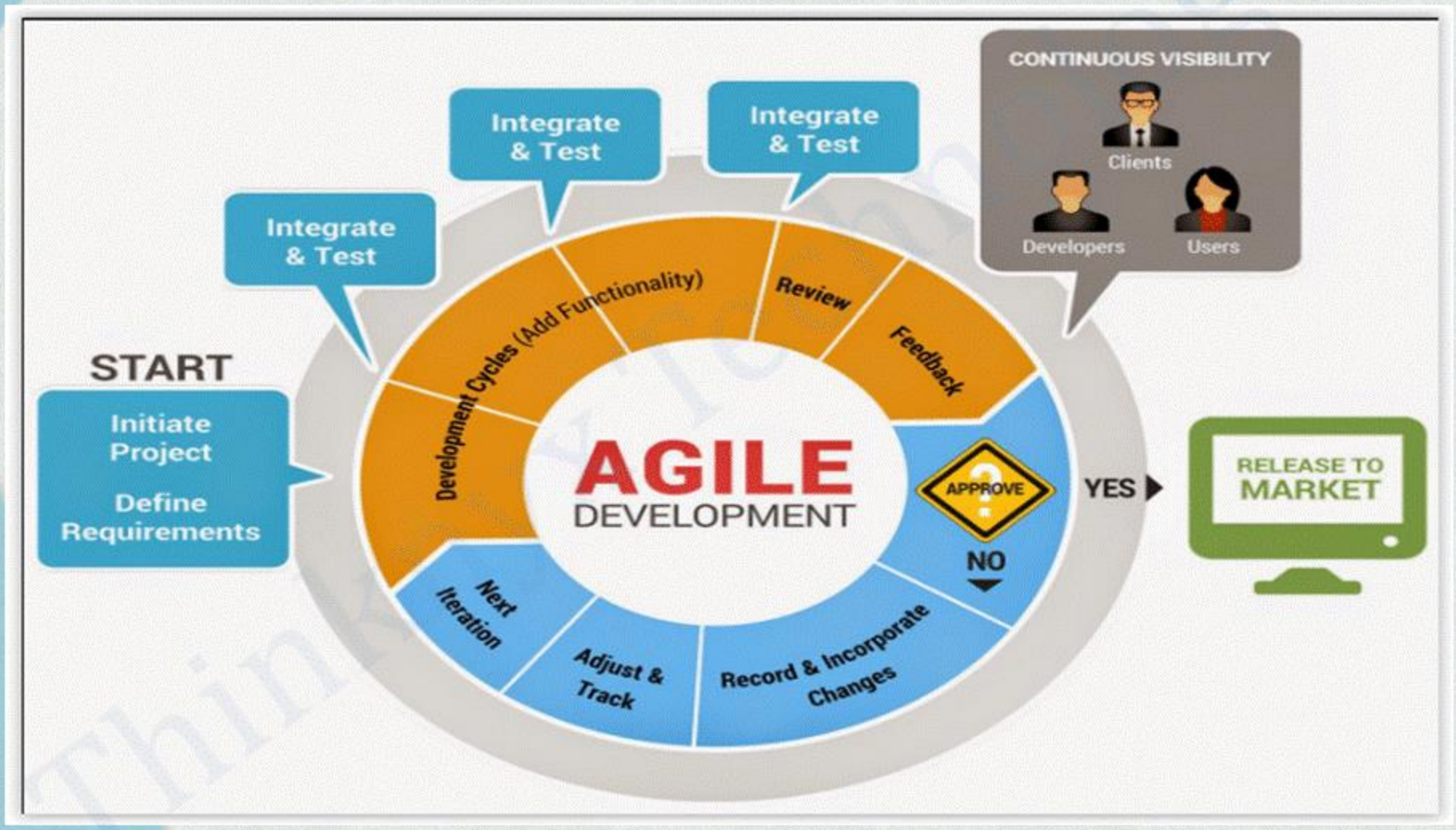
- Some of the most commonly used methodologies in SDLC are:

- Waterfall
- Agile
- Lean
- Scrum

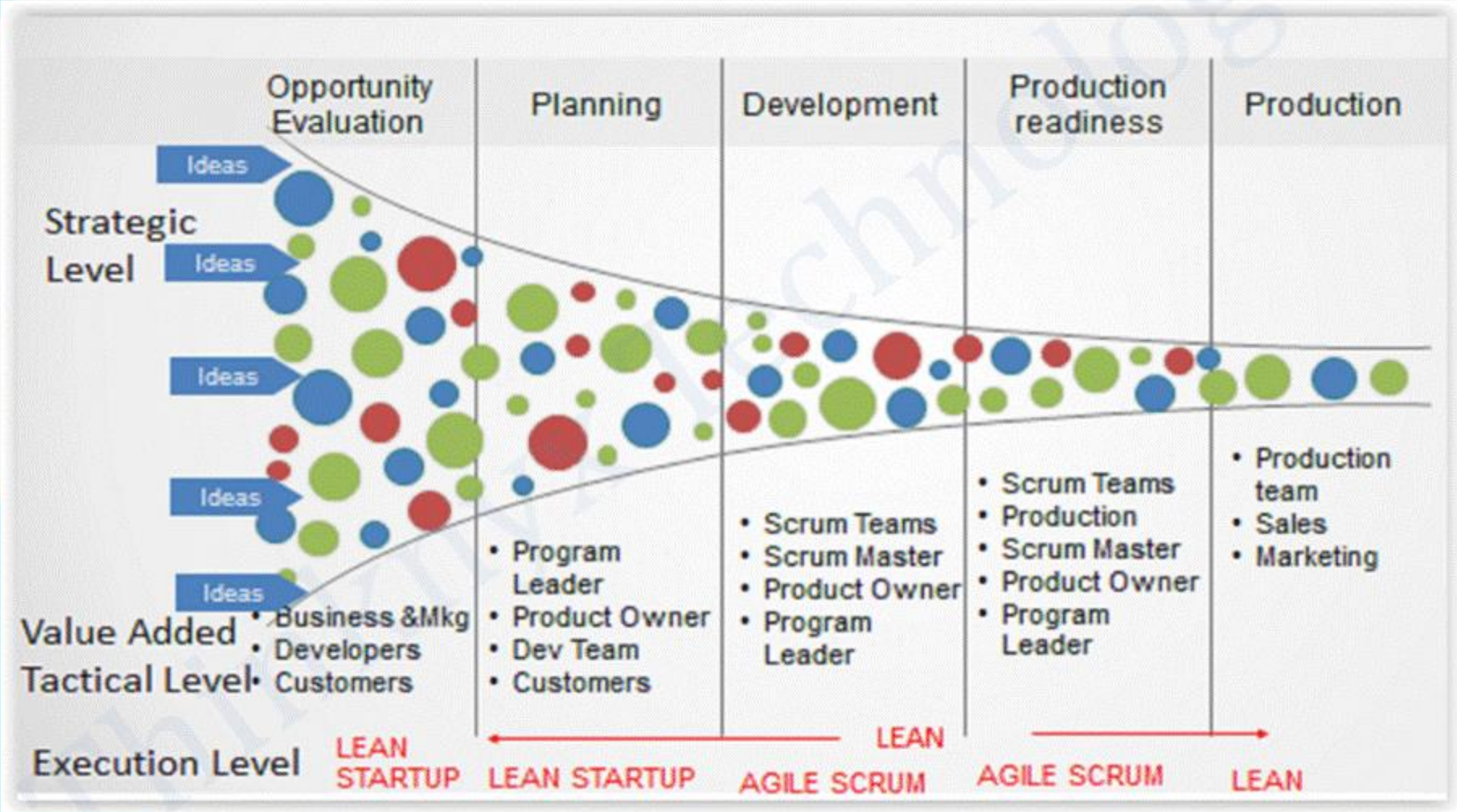
Waterfall Development Model



Agile Development Model



Lean Development Model

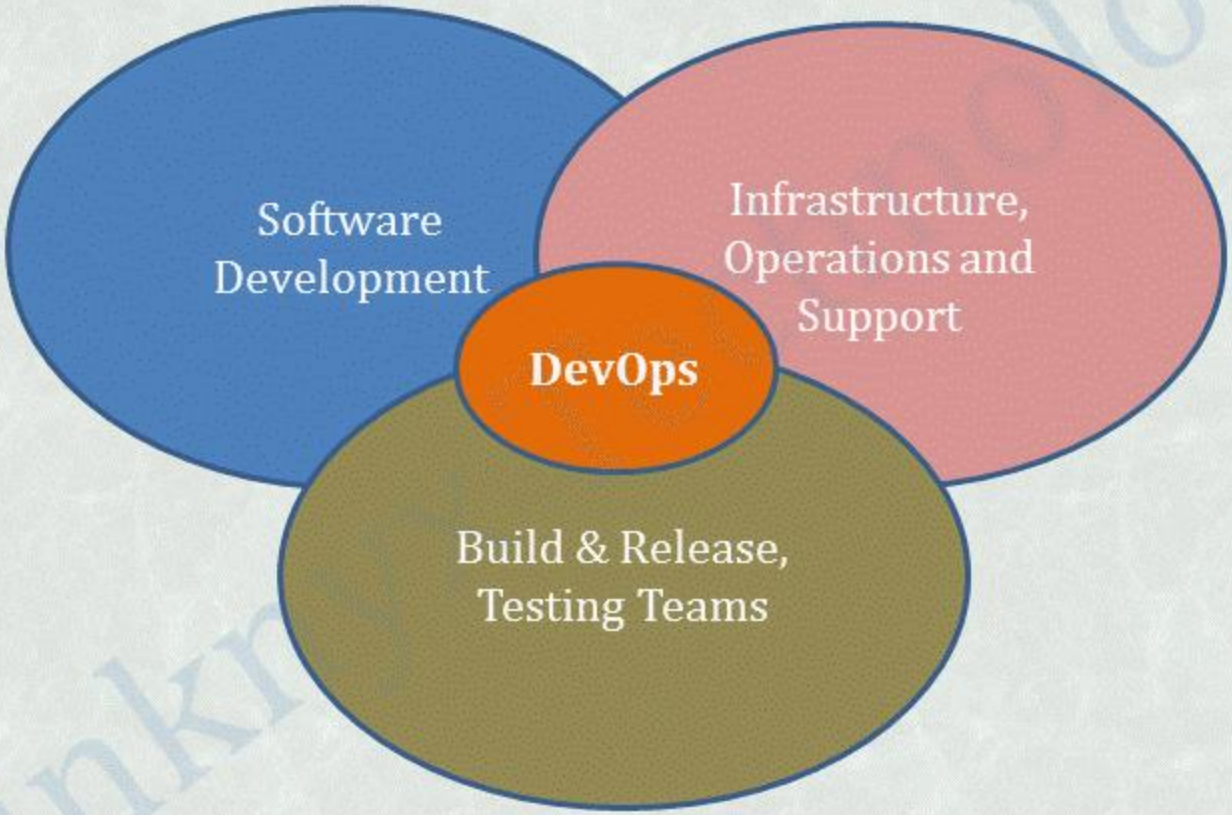


Few Questions .. Why DevOps ??

Few Questions .. Why DevOps ??

- The problems due to the wall of confusion have been encountered by many IT organizations.
- DevOps took birth due to the wall of confusion between Development and Operations.

DevOps



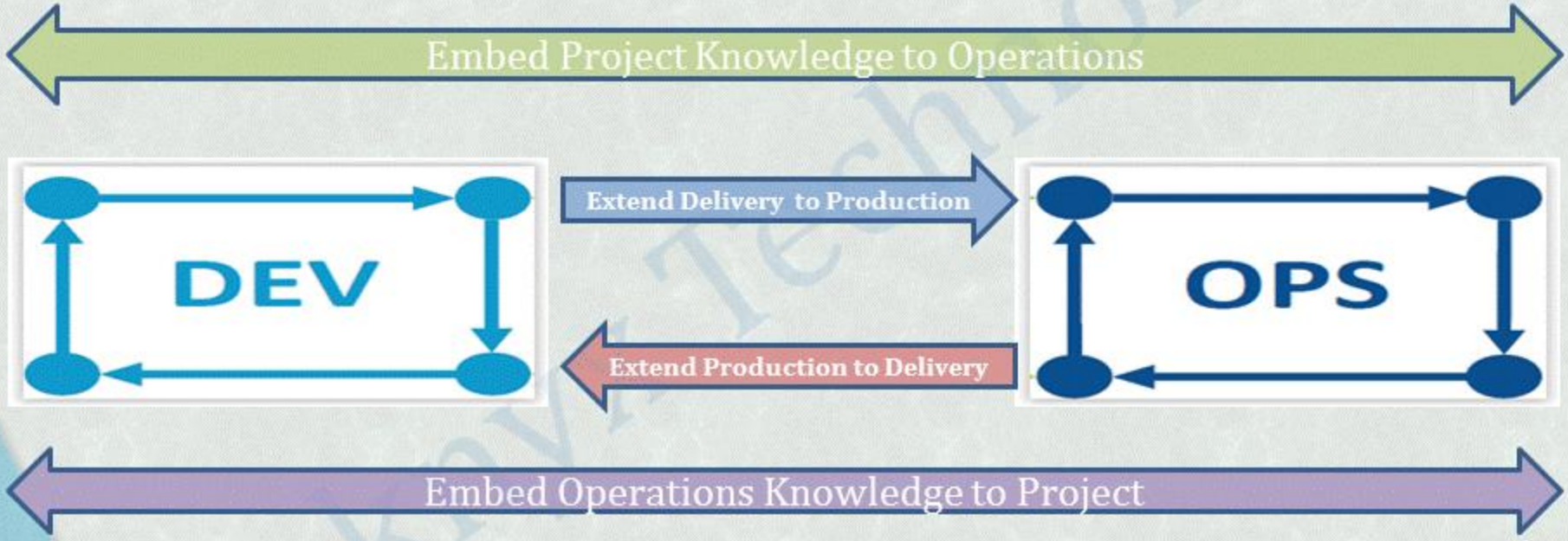
What DevOps is for?

- DevOps is a culture for Integration, Collaboration and Communication between different cross functional teams (including ops) for Continuous delivery.
- DevOps encourages Operations to participate early in the Development cycle so that products are designed to be easily deployable and maintainable.
- DevOps emphasizes on keeping WIP/Inventory low and go to production ASAP.
- DevOps is not just a framework or a workflow. It's a culture that is overtaking the business world. DevOps ensures collaboration and communication between software engineers (Dev) and IT operations (Ops). **With DevOps, changes make it to production faster.** Resources are easier to share. And large-scale systems are easier to manage and maintain.

DevOps - Pillars

- Integration
- Collaboration
- Communication
- Adaptation to Changes
- A Cultural Handshake

DevOps - Objectives

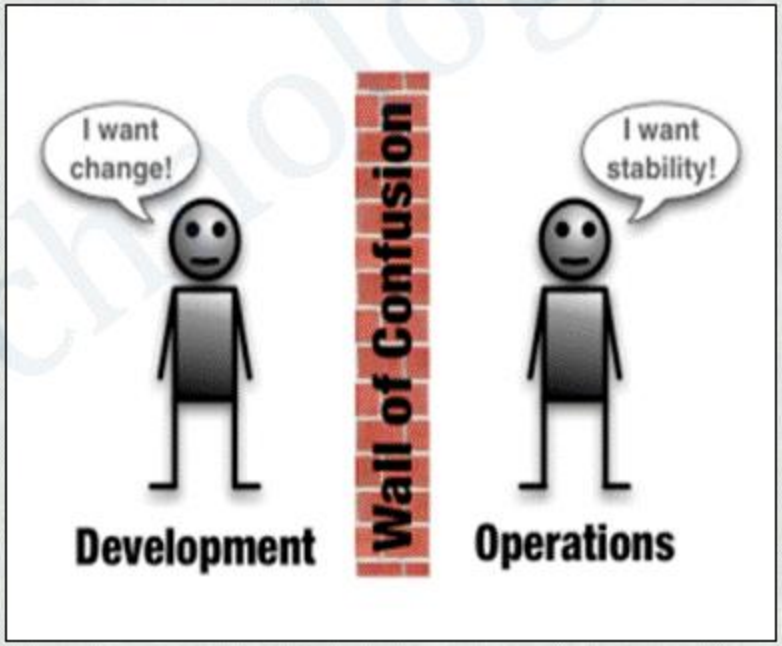


Establish DevOps by ..

- Continuous Integration
- Automated Testing
- Continuous Deployment
- Continuous Delivery
- Infrastructure as a Code
- Metrics (Reporting)
- Collaboration
- Making smart use of smart people

DevOps – Essence

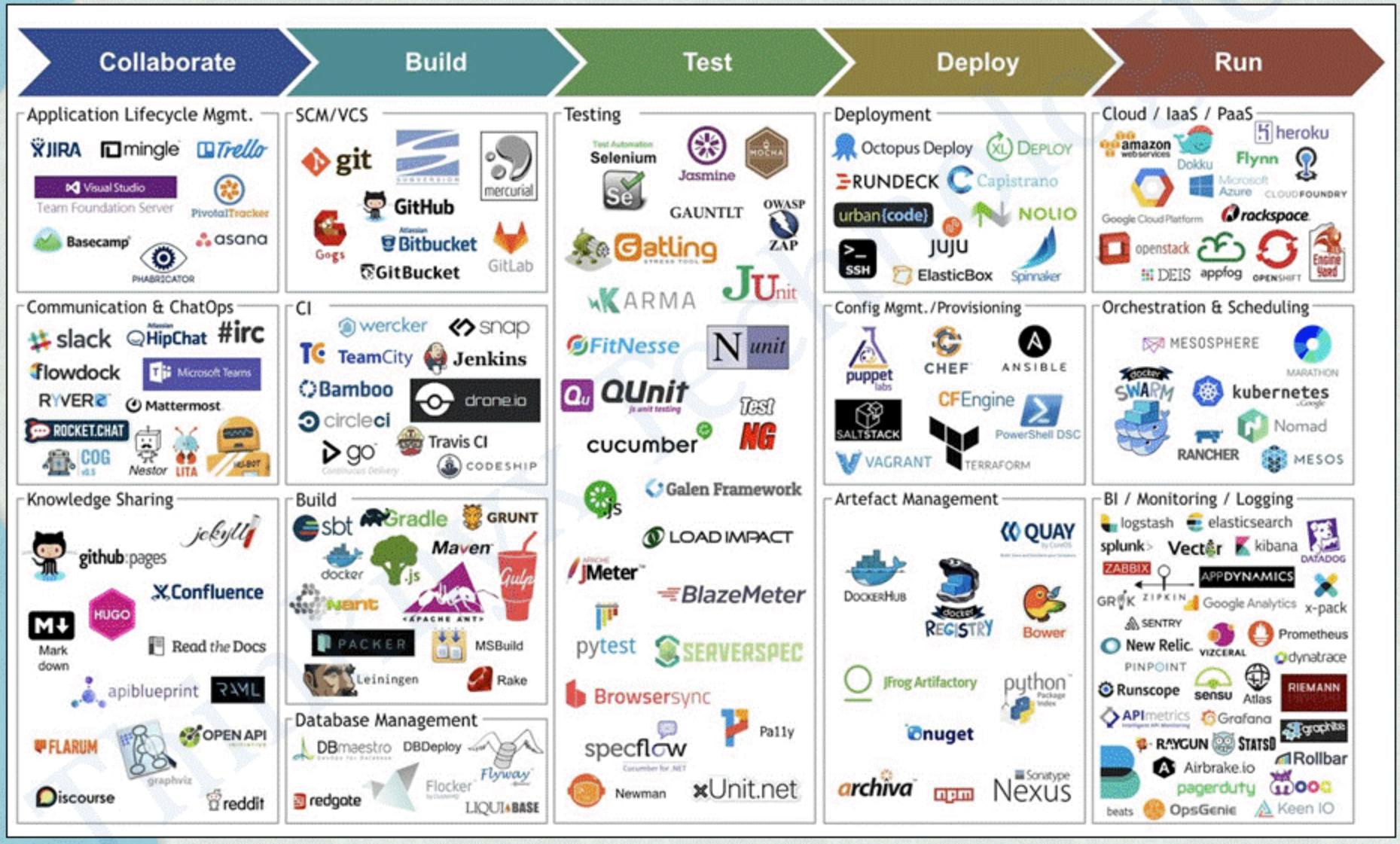
- **Efficiency** - Faster time to market
- **Predictability** - Lower failure rate of new releases
- **Reproducibility** – Version everything
- **Maintainability** - Faster time to recovery in the event of a new release crashing or otherwise disabling the current system



"Break down the wall between development and operations"

DevOps - ToolSet

Thinknyx

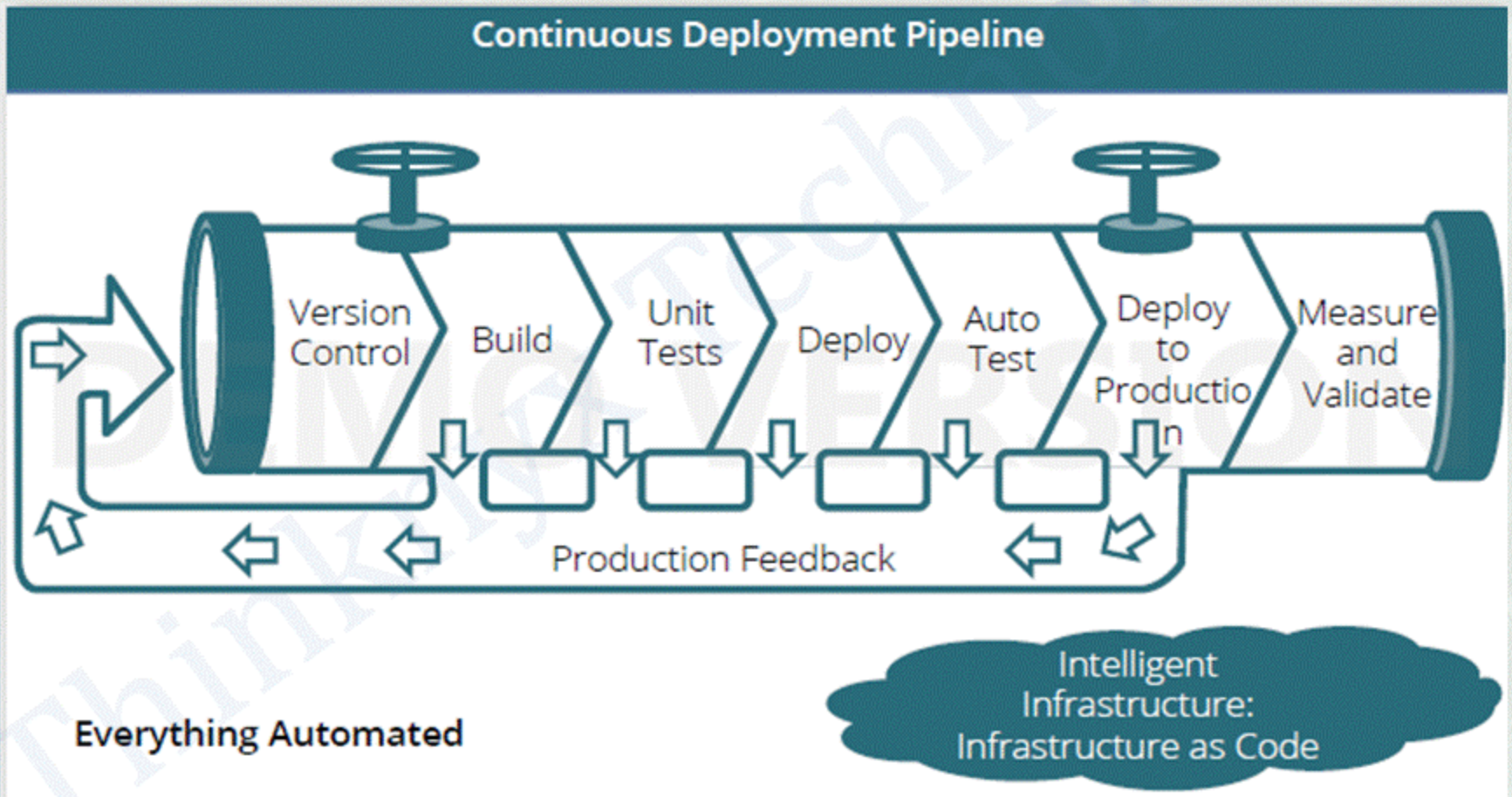


Qualities of DevOps Engineer

- Technical Strong
- Understand the need
- Servant Leader
- Flexible
- Open Minded
- Courageous
- Risk Taker

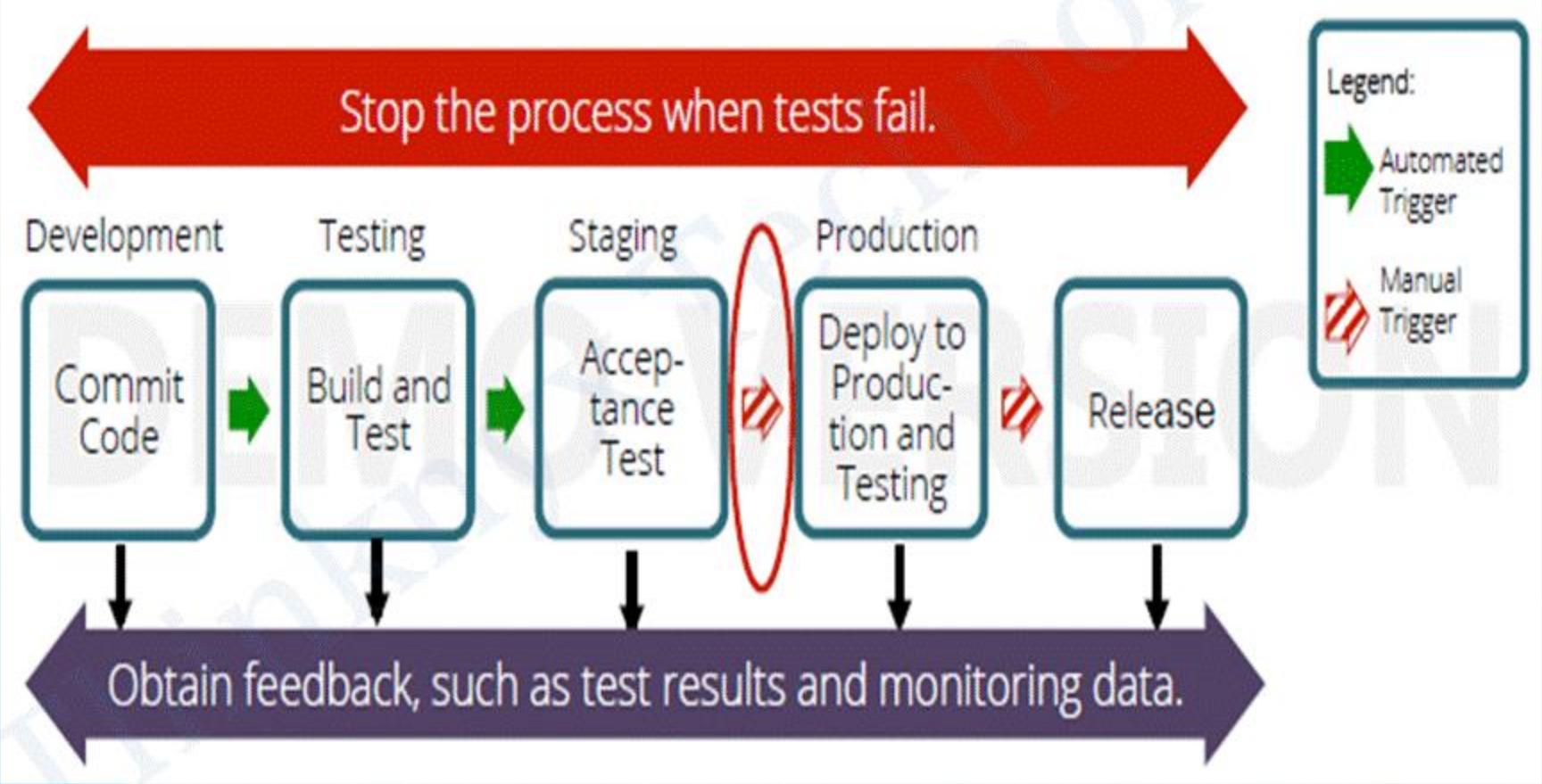
Continuous Deployment

- Automate Everything and Obtain Immediate Feedback



Continuous Delivery

- The implementation of the code to Production is manual



Cloud Computing Fundamentals

THINKNYX
TECHNOLOGIES

Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

Introduction to Cloud Computing

- The obvious question today in every mind is

What is Cloud?

- Cloud computing is the use of computing resources (hardware & software) that are delivered as a service over a network (typically the internet)

What is cloud?

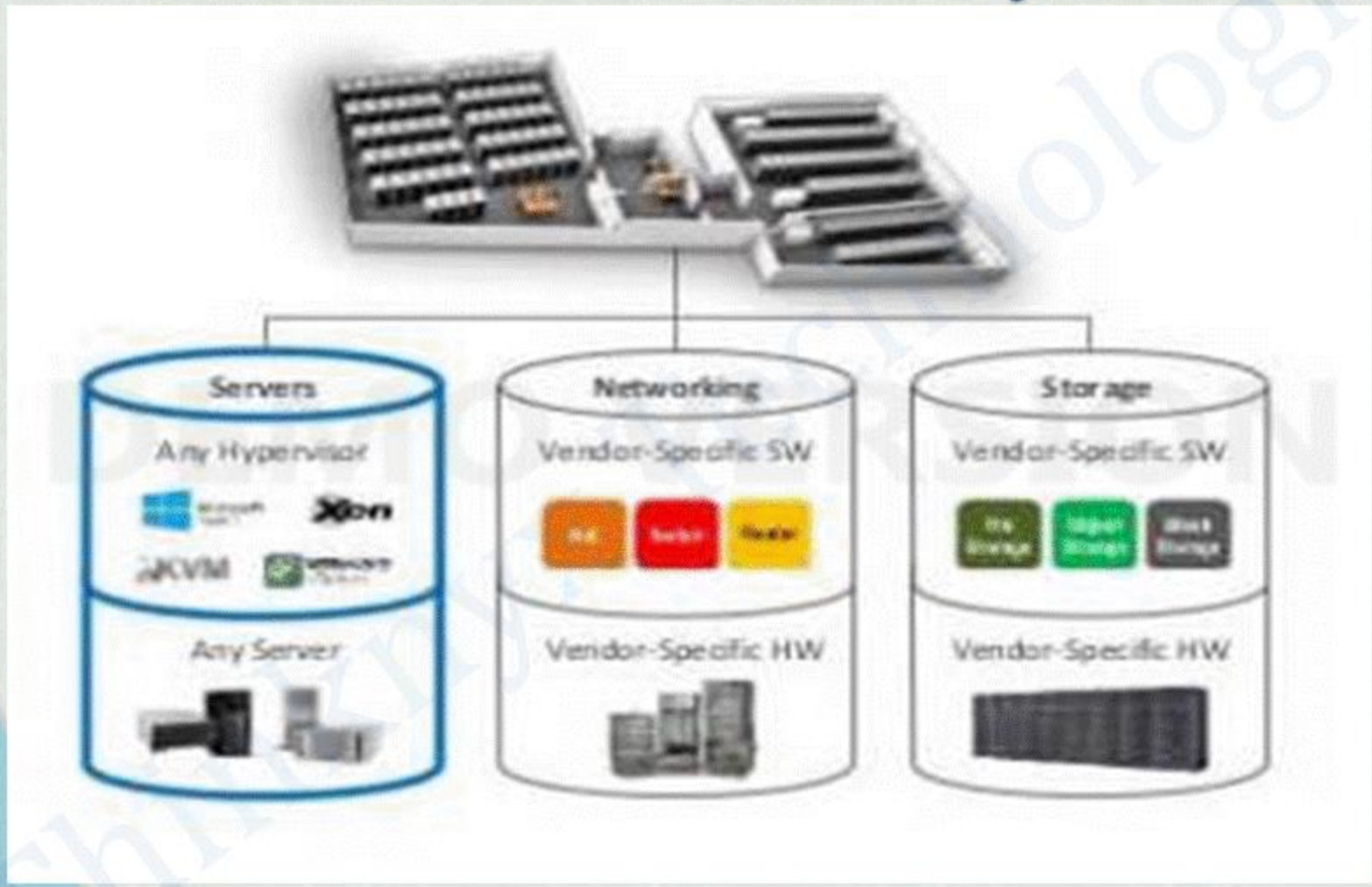
- "Cloud Computing is the delivery of Computing as a service rather than a Product."

- But how?..

- Any cloud model is composed of:

- five essential characteristics
- three service models
- four deployment models

Traditional DC and why cloud?



Traditional DC and why cloud?

- Main issues with Traditional IT Infrastructure.

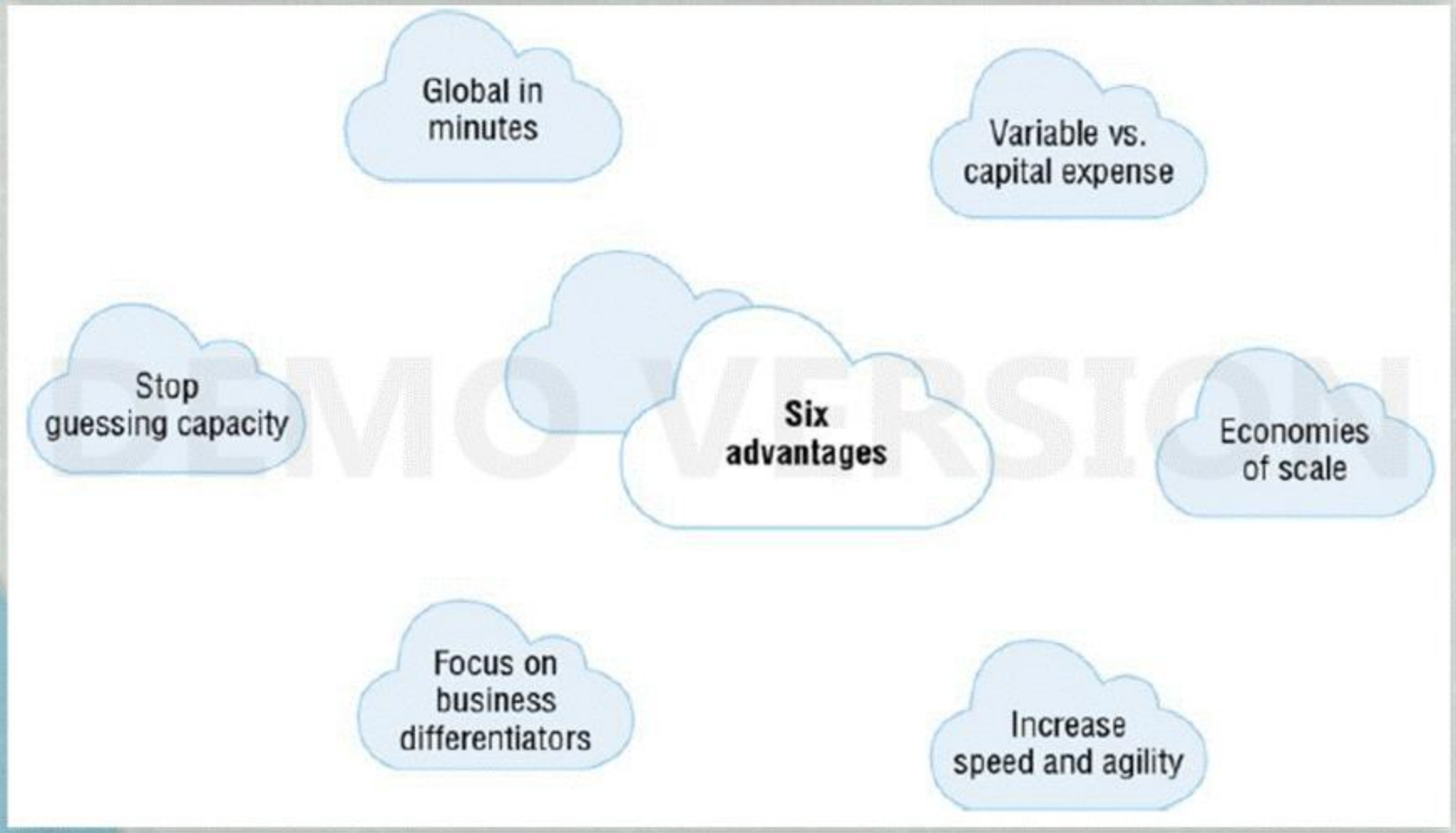
- Infrastructure is not a core business
- Hard to Scale
- Dedicated Infrastructure teams
- Dedicated Datacenters
- Dependency on vendors (servers, switches, cables etc.)
- Underutilized Resources
- High Cost

Traditional DC and 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
- Different storage options
- Optimized use of resources

Traditional DC and why cloud?

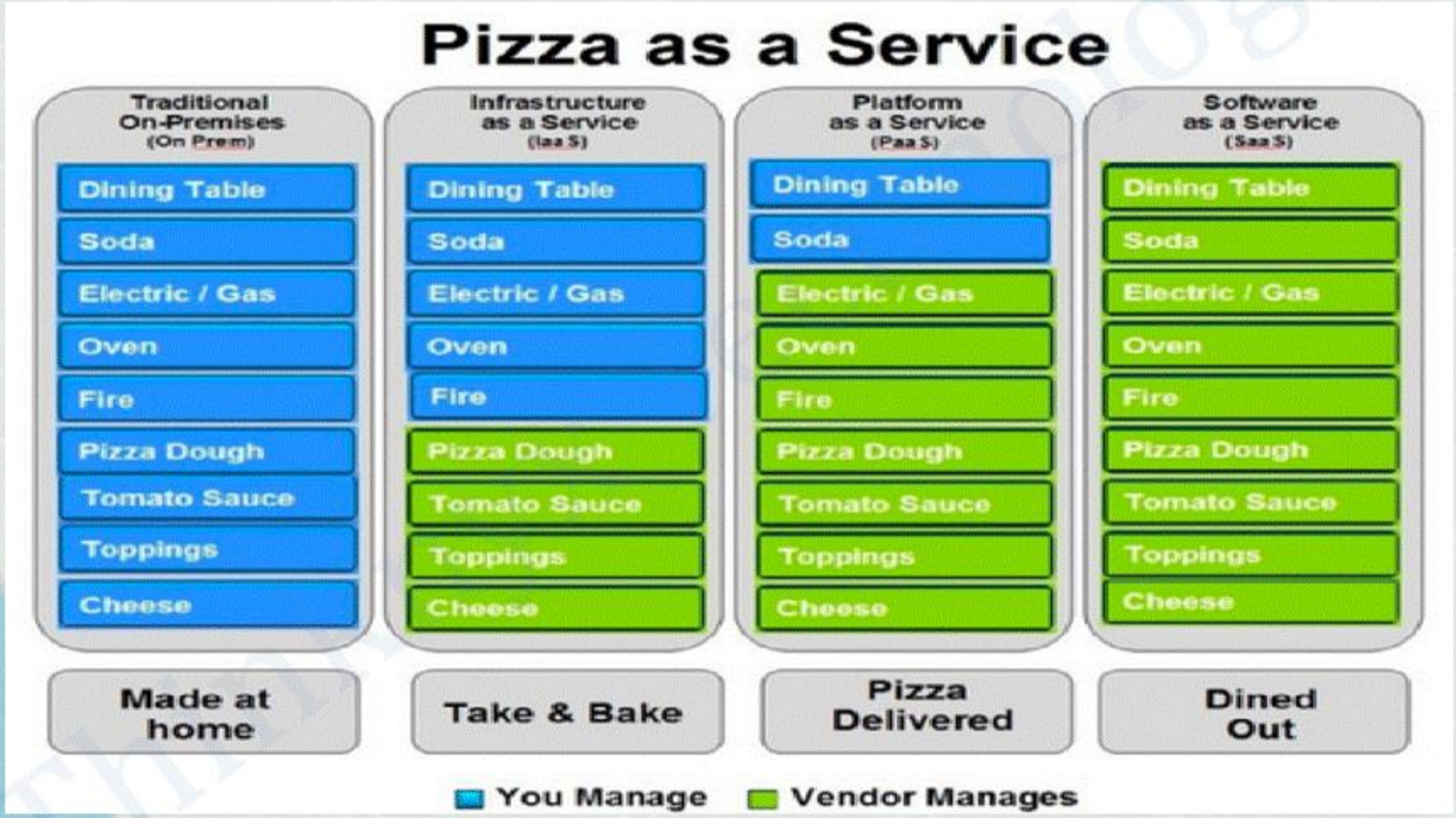


Cloud Service Models

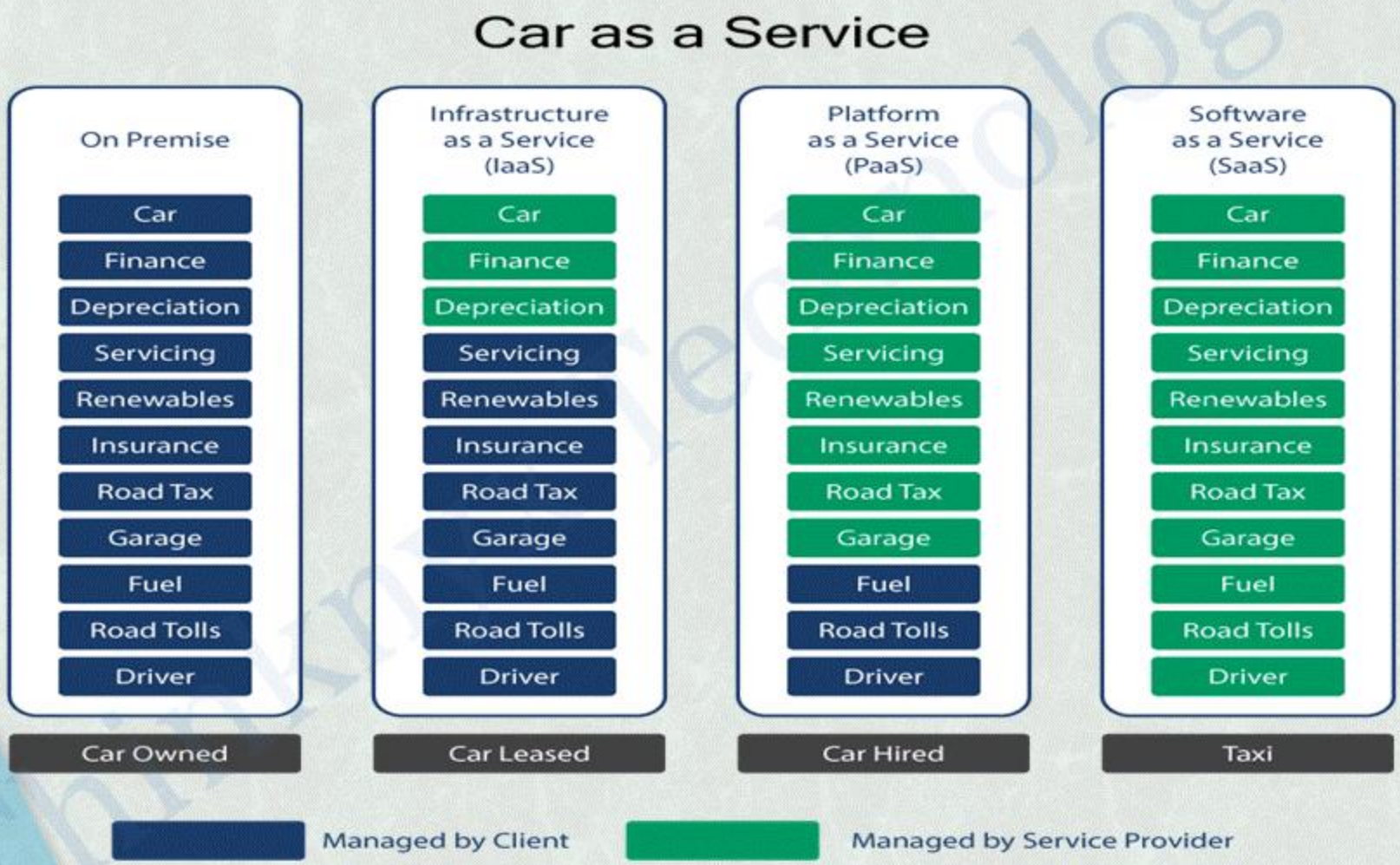
- There are three Cloud Computing Service Models:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

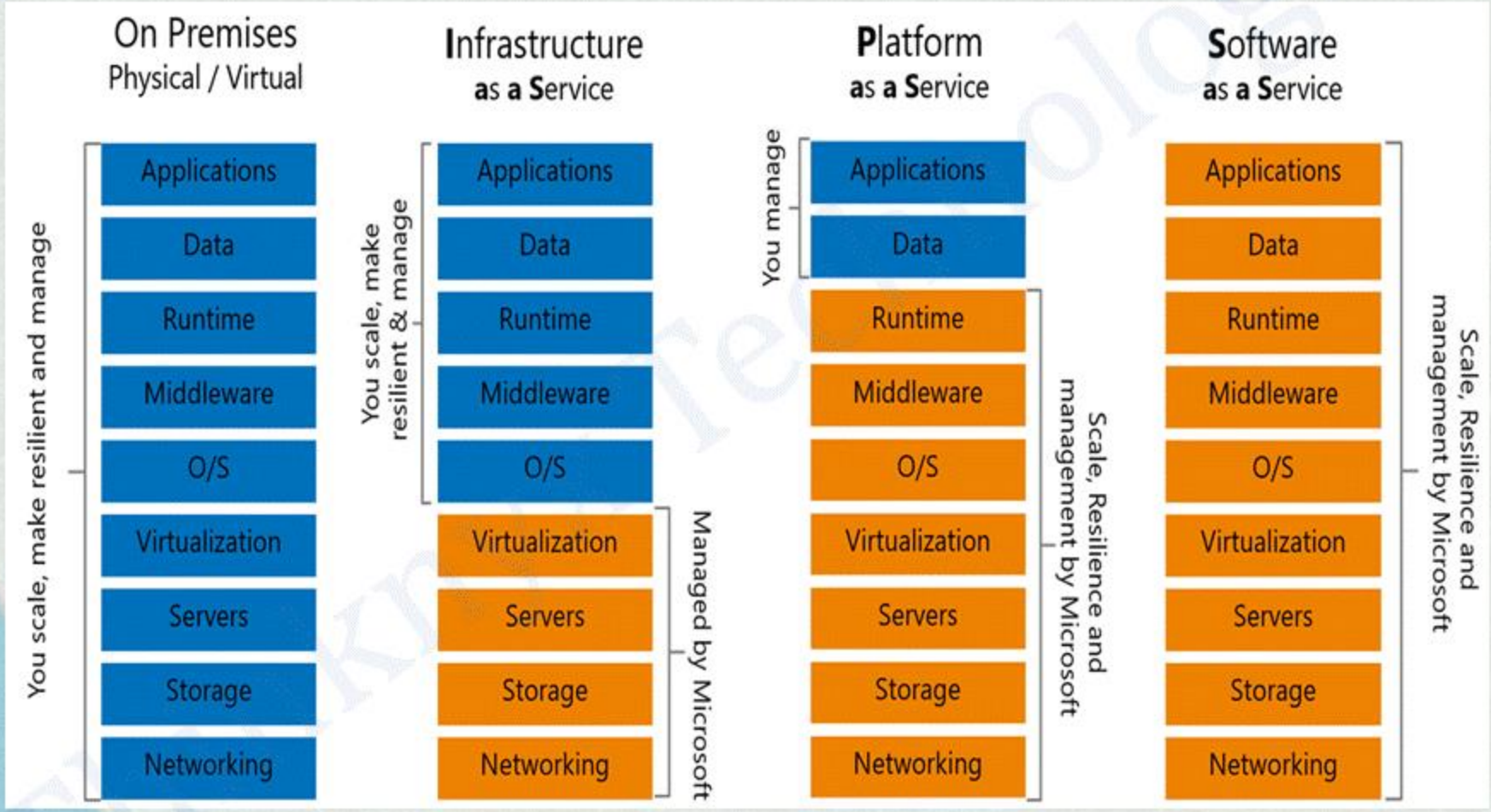
Responsibility- Who owns What?



Responsibility- Who owns What?

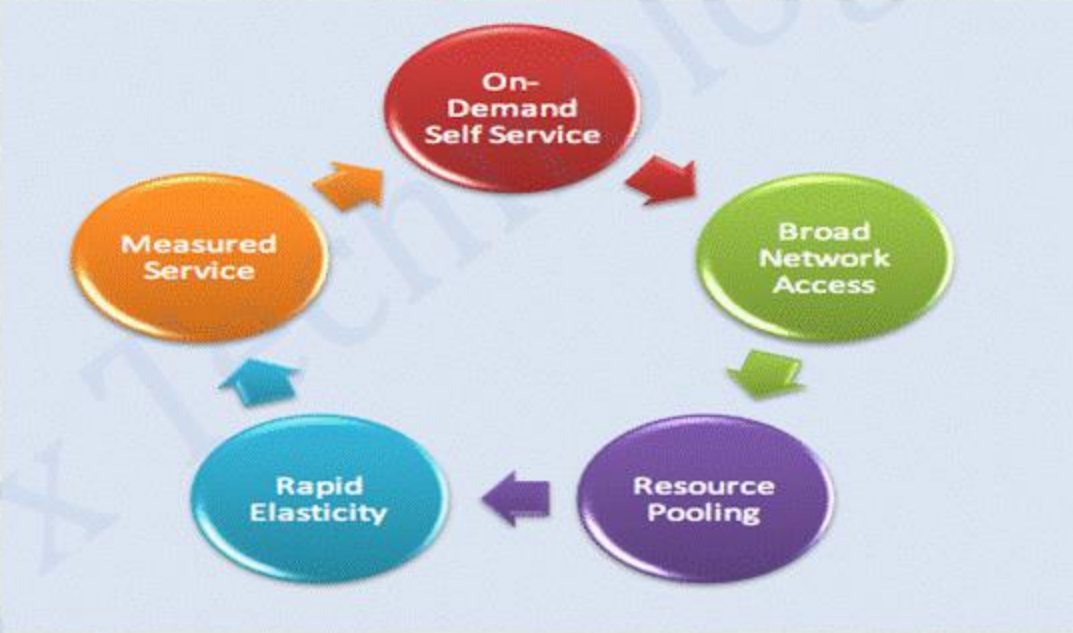


Responsibility- Who owns What?



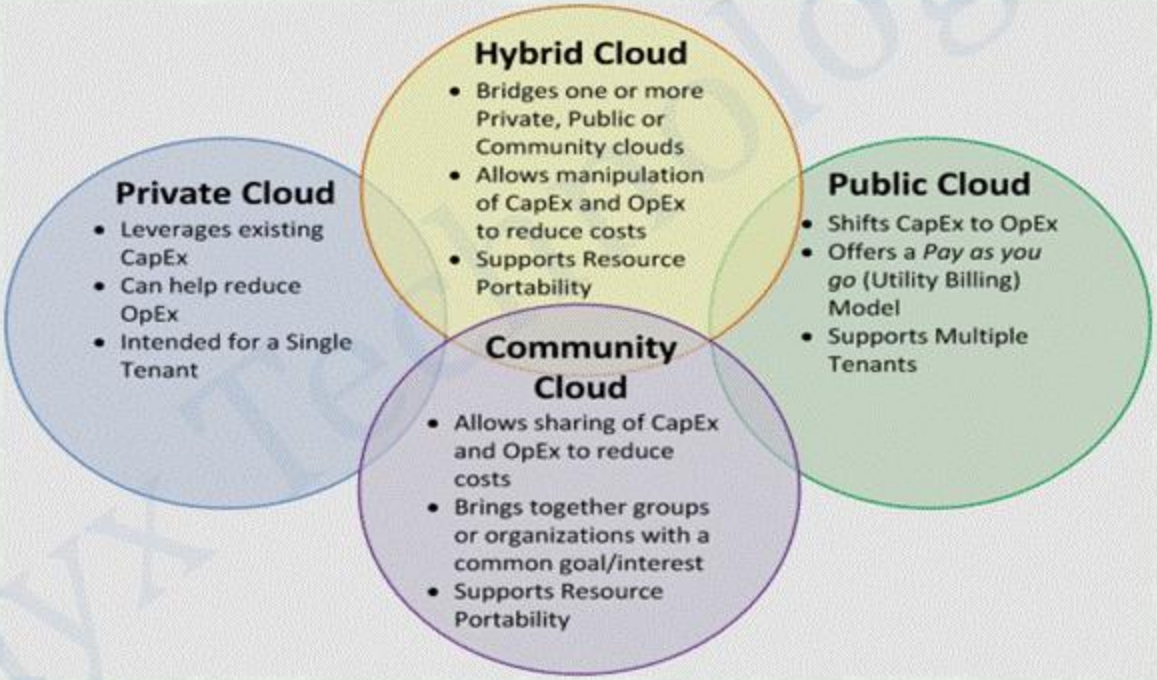
Cloud Essentials Characteristics

- On-demand self-service
- Broad network access
- Resource Pooling
- Rapid Elasticity
- Measured Services



Cloud Deployments Types

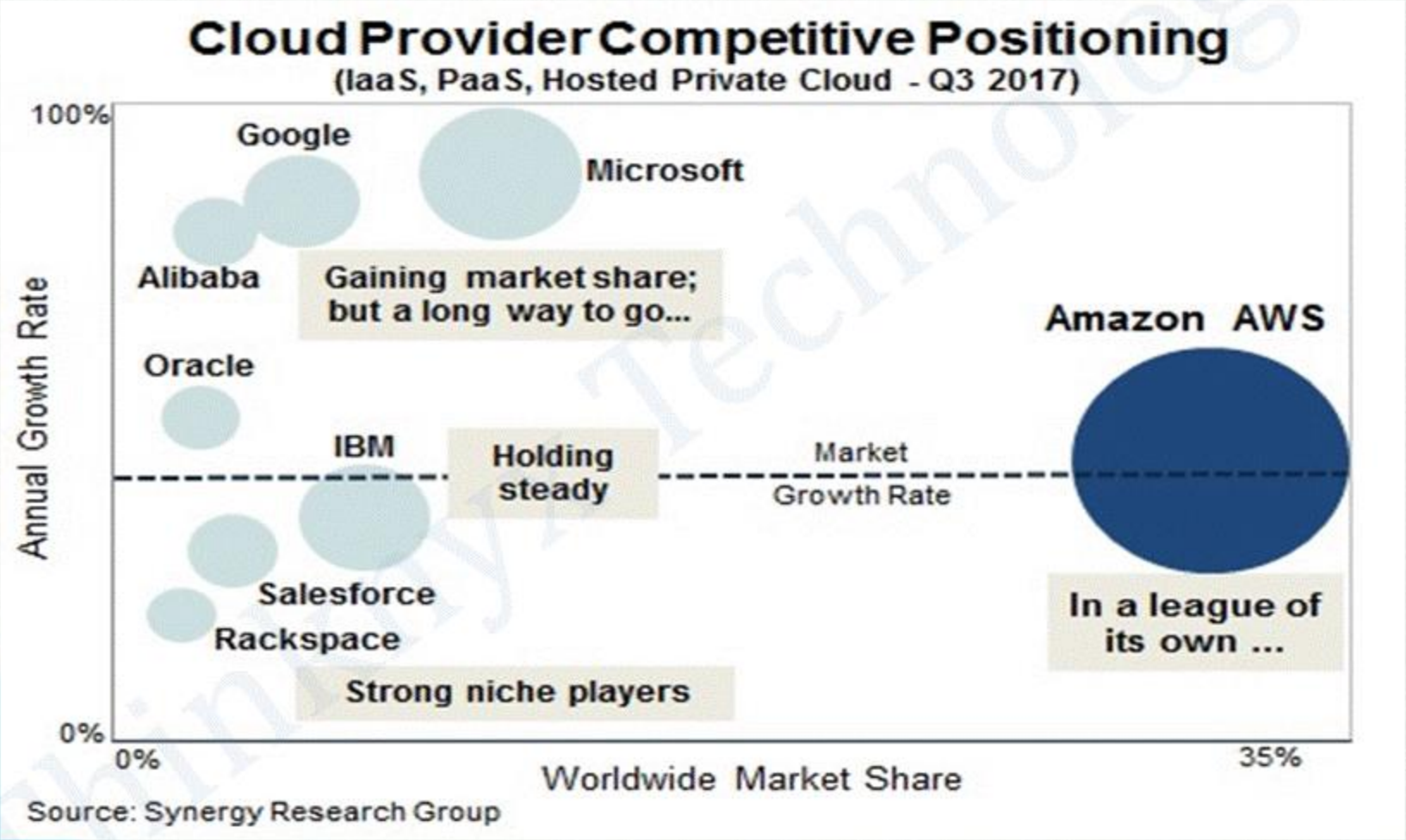
- Public Cloud
- Private Cloud
- Hybrid Cloud
- Community Cloud



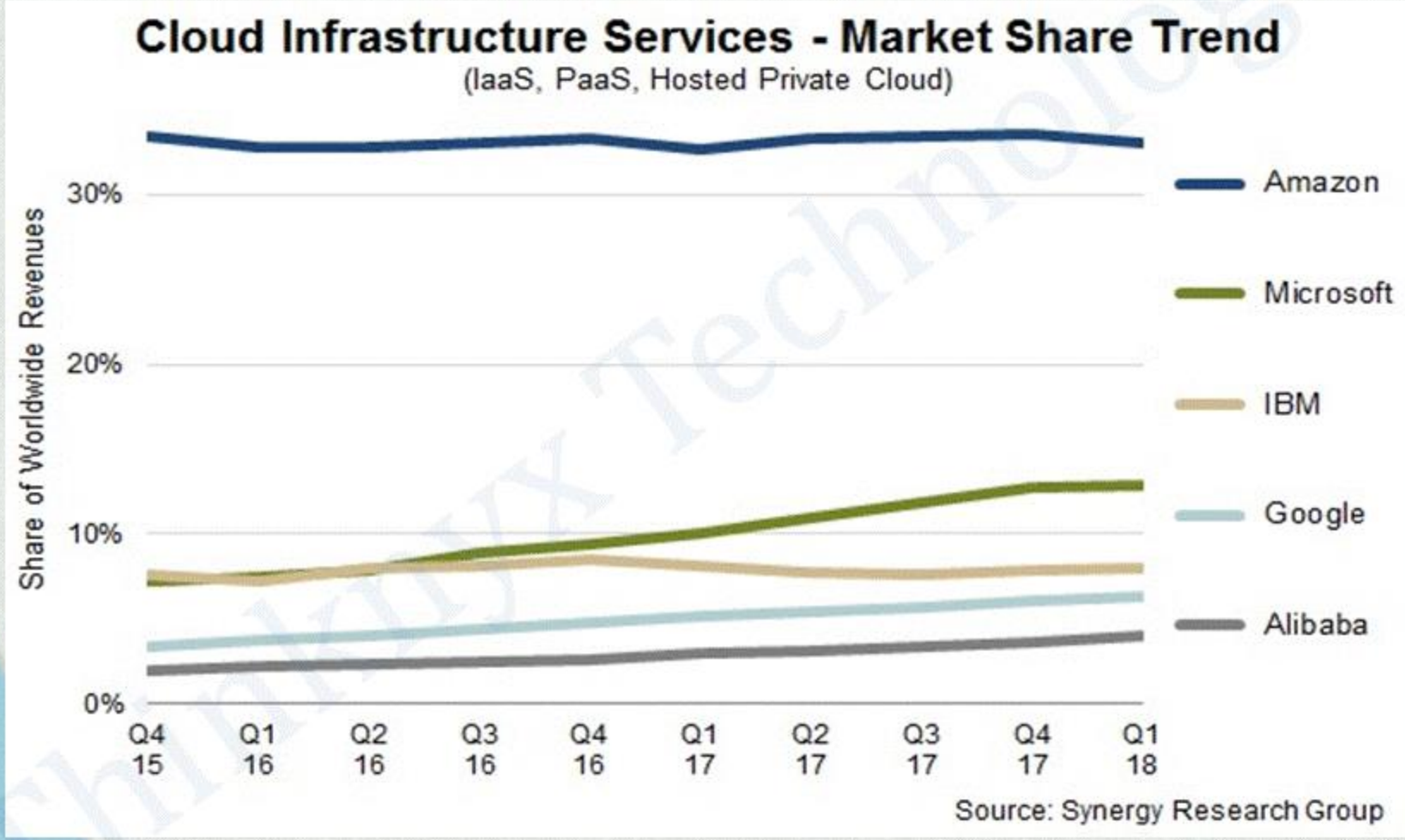
Cloud Vendors

- AWS
- Microsoft Azure
- DigitalOcean
- RackSpace
- IBM Bluemix
- Oracle Cloud
- PackStack
- Pivotal CloudFoundary

Who stands where?



Who stands where?



What is AWS

- AWS (Amazon Web Services) is a group of web services (also known as cloud services) provided by Amazon.
- AWS provide IT infrastructure like CPU, Storage as a service, which means there is no need for any hardware procurement.
- 100's 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.

AWS Core Benefits

- **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 severs, 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 in 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. Infact systems based on AWS are usually more secure than inhouse IT infrastructure systems.

AWS Global Infrastructure

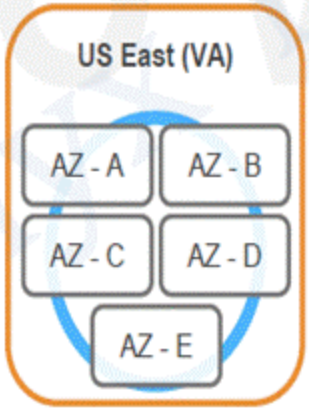
- **AWS Regions:**
- Geographic Locations
- Consists of at least two Availability Zones(AZs)
- All of the regions are independent of each other with separate Power Sources, Colling and Internet connectivity.

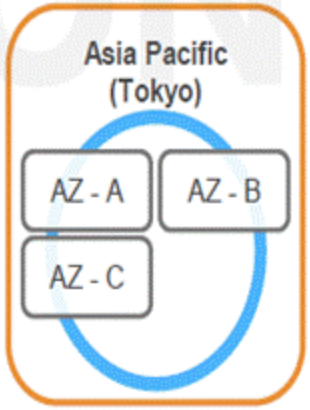
- **AWS Availability Zones**
- AZ is a distinct location within a region
- Each zone is insulated (with low-latency links) from other to support single point of failures
- Each Region has minimum two AZ's

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.

AWS Global Infrastructure

AWS Cloud Computing Platform

Accessing AWS Platform

- AWS Management Console
- AWS Command line interface
- AWS Software Development Kit's

Version Control Systems with GIT

THINKNYX
TECHNOLOGIES

Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

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

Thinknyx

- 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

Types of Version Control Systems

- Majorly VCS is divided into two parts:

- Centralized Version Control System
- Distributed Version Control System

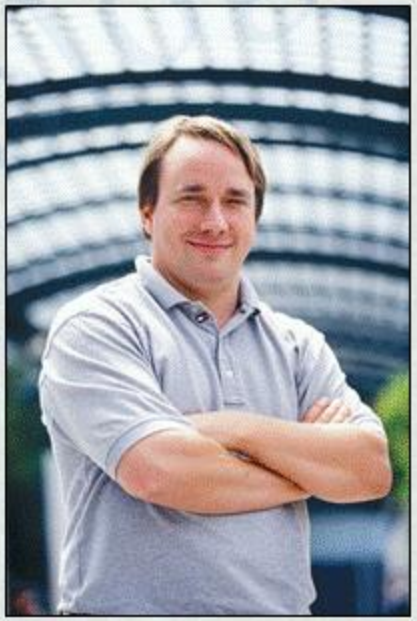
GIT History

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

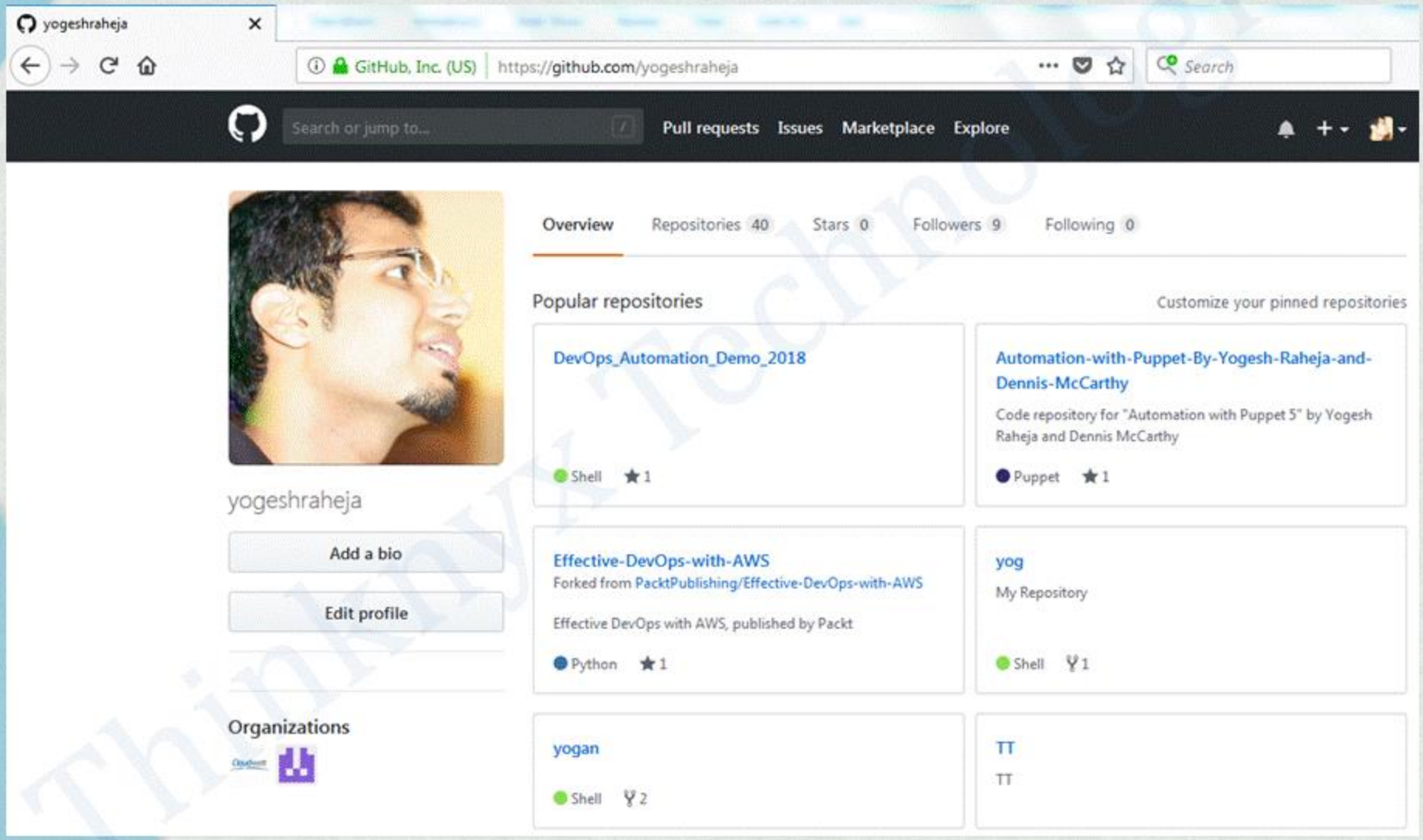
GIT HELP

```
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.
```

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.
- **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

GitHub



Continuous Integration with Jenkins

THINKNYX
TECHNOLOGIES

Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

What is Continuous Integration

- In Simple term Continuous Integration is a term which means “checking the compatibility of a change you have made with the remaining code base or modules of an application or to check the impact of that change on the application functionality”.
- Continuous Integration is not helping us in fixing the bugs but it helps us to Identify those bugs in the early phases of a development lifecycle and with much faster rate.
- Continuous Integration is not the sole responsibility of the developers but every individual working in the team is responsible for the Healthy Continuous Integration.

Why Continuous Integration

- CI helps in reduction of efforts which will increase exponentially with the increase in:
  - Number of components
  - Number of Bugs
  - Parallel development
  - Time from last Integration

Jenkins History

- Jenkins was originally developed as the Hudson project. Hudson's creation started in summer of 2004 at Sun Microsystems. Kohsuke Kawaguchi, the current CTO of CloudBees, "***A Company providing Enterprise Level Support for Jenkins, when he was working with Sun Micro Systems***"

- Certain clashes between Sun Micro Systems and Hudson Community for the Management of the Project

- Today Jenkins is the most used Continuous Integration tool in the market, which provide variety of functions to run with the help of large Jenkins Community.

Brief about Jenkins

- An Open Source CI Server with MIT (Massachusetts Institute of Technology)Licensing Large Community Base
- More than 150,000 installations as on Dec, 2016
- More than 1300 plug-in support which make to compatible with almost every delivery tool available in market
- Annual meet-up with the Jenkins World
- CloudBees is the biggest promoter and Contributor of Jenkins Community and also provide Enterprise Support for Jenkins
- *Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and deploying software.*

What Jenkins can do ..?

- Automated build triggers
- Get source code from repository
- Automated build
- Automated Testing
- Trend Reports
- Bug Reporting
- Automate Notifications
- Automate Deployments
- Automate BAU Tasks like Space Cleanup, Server Monitoring, Patching, Configuration Management run, Containers run etc.

Why Jenkins?

- **Continuous Integration and Continuous Delivery:** As an extensible automation server, Jenkins can be used as a simple CI server or turned into the continuous delivery hub for any project.
- **Easy installation:** Jenkins is a self-contained Java-based program, ready to run out-of-the-box, with packages for Windows, Mac OS X and other Unix-like operating systems.
- **Easy configuration:** Jenkins can be easily set up and configured via its web interface.
- **Plugins:** With hundreds of plugins in the Update Center, Jenkins integrates with practically every tool in the continuous integration and continuous delivery toolchain.
- **Extensible:** Jenkins can be extended via its plugin architecture, providing nearly infinite possibilities for what Jenkins can do.
- **Distributed:** Jenkins can easily distribute work across multiple machines, helping drive builds, tests and deployments across multiple platforms faster.

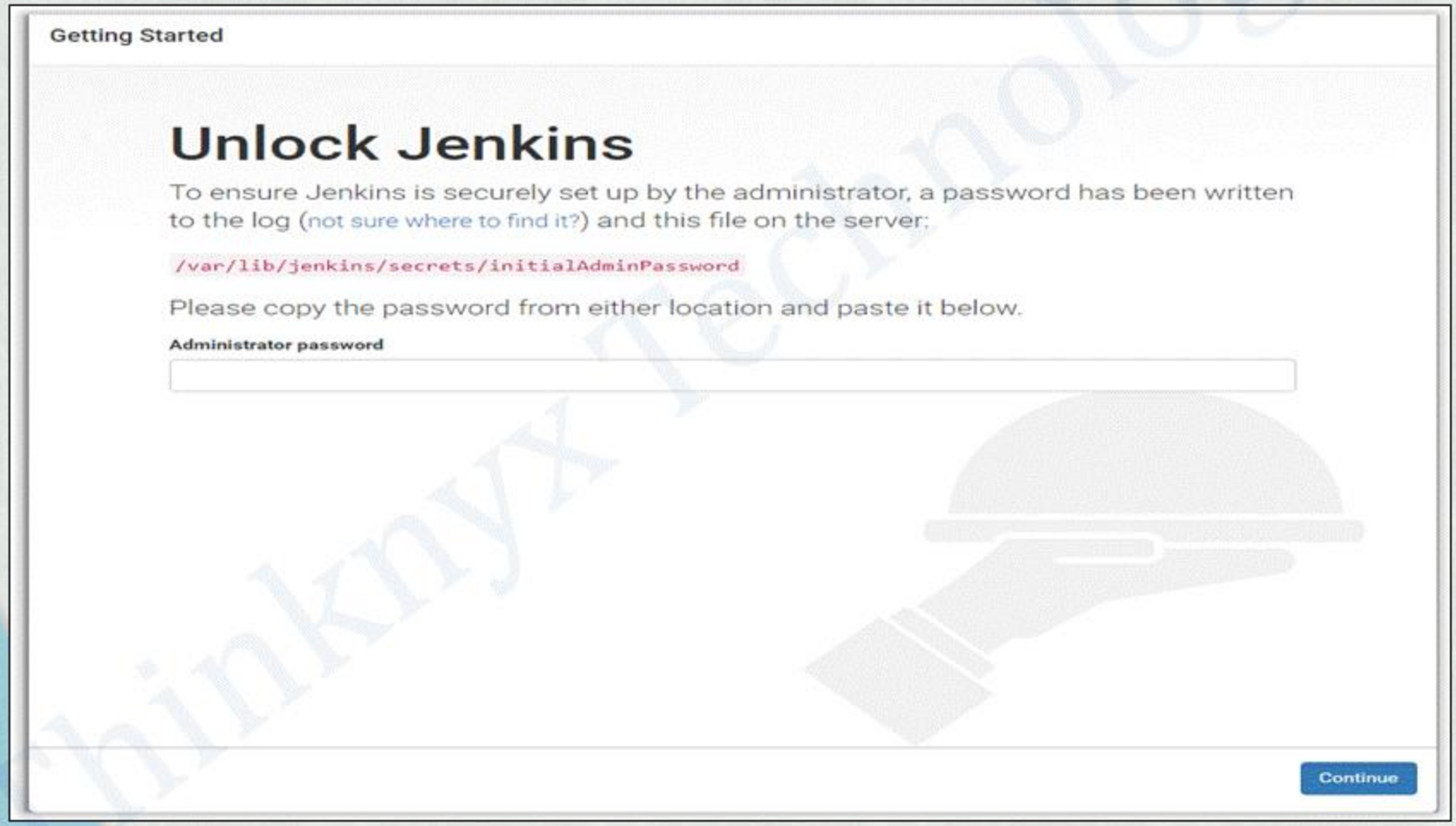
Installation & Configuration Requirements

- Web Server (Webserver like: Tomcat, WebLogic)
- Java 1.8 (Java version 8, either a JRE or Java Development Kit (JDK) is fine)
- Build tools (for development to do packaging)
- VCS Clients (Version Control System)
- Plug-ins Configuration
- 256 MB of RAM, although more than 512MB is recommended
- 10 GB of drive space (for Jenkins and your Docker image)

Installing Jenkins using Yum in CentOS

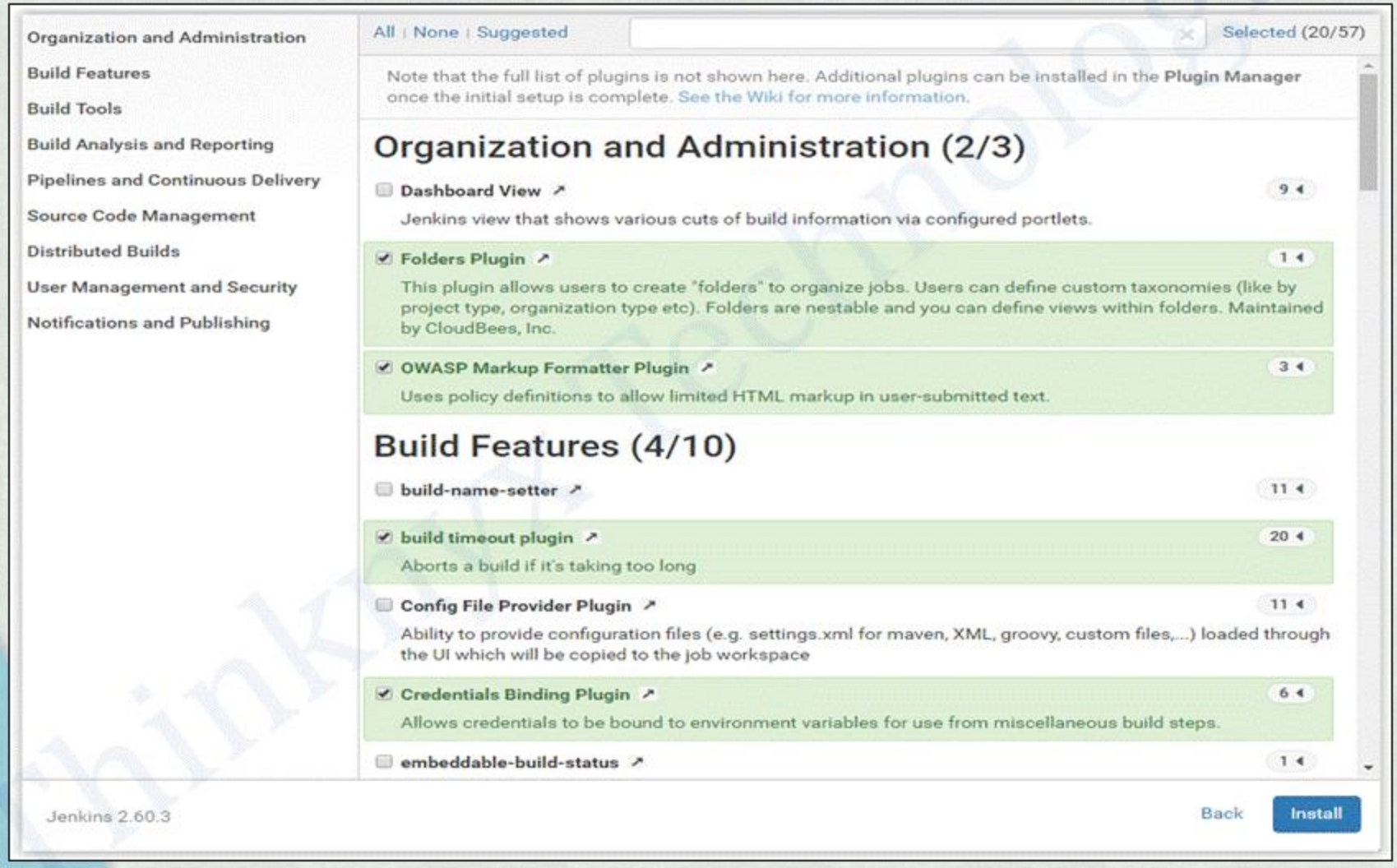
- sudo yum update
- sudo yum install java-1.8.0-openjdk.x86_64
- java –version
- Set JAVA_HOME and JRE_HOME
  - export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk
  - export JRE_HOME=/usr/lib/jvm/jre
- sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo
- sudo rpm --import http://pkg.jenkins-ci.org/redhat-stable/jenkins-ci.org.key
- sudo yum install jenkins
- sudo systemctl start jenkins.service
- sudo systemctl enable jenkins.service
- Once done try to access the Jenkins from http://localhost:8080

Getting Started



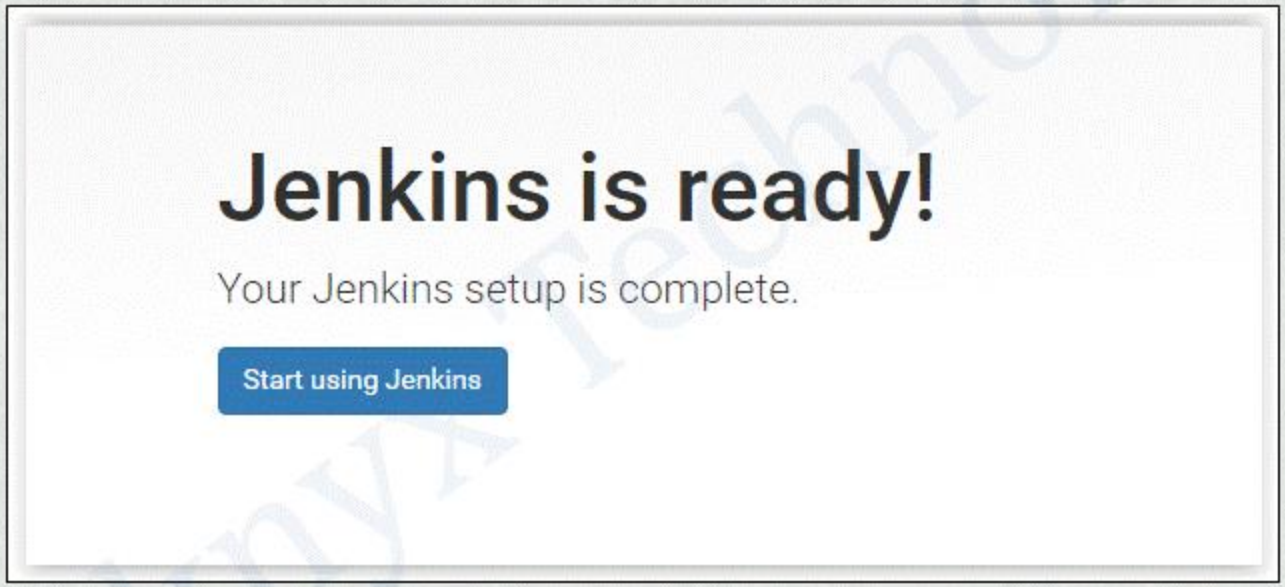
Customize Jenkins

Select Plug-ins



Creating Admin User

We are Ready



Jenkins Dashboard

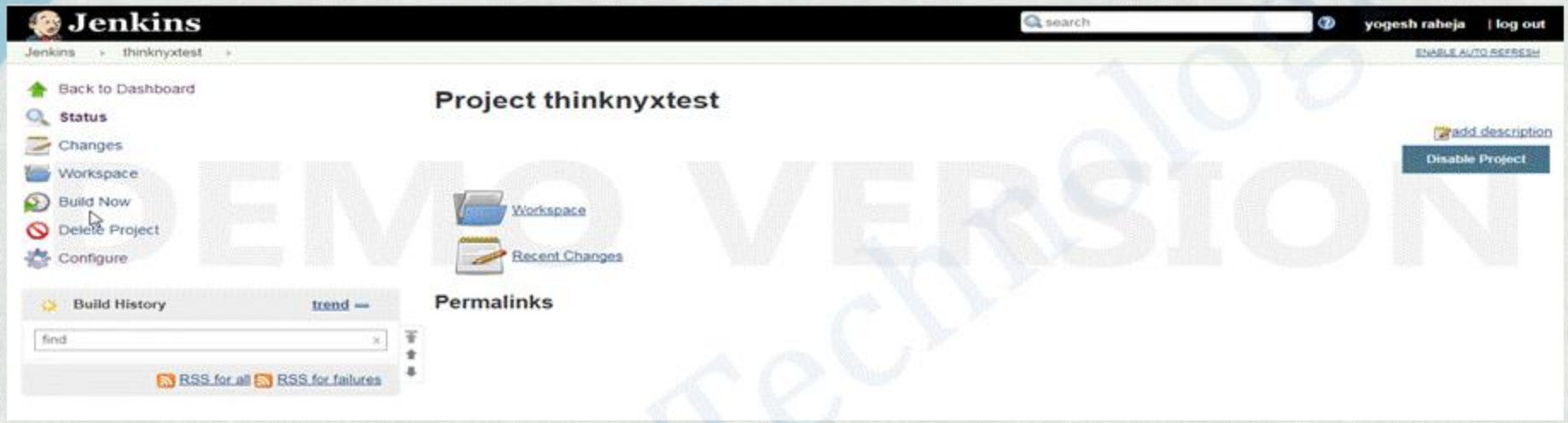
Plug-ins

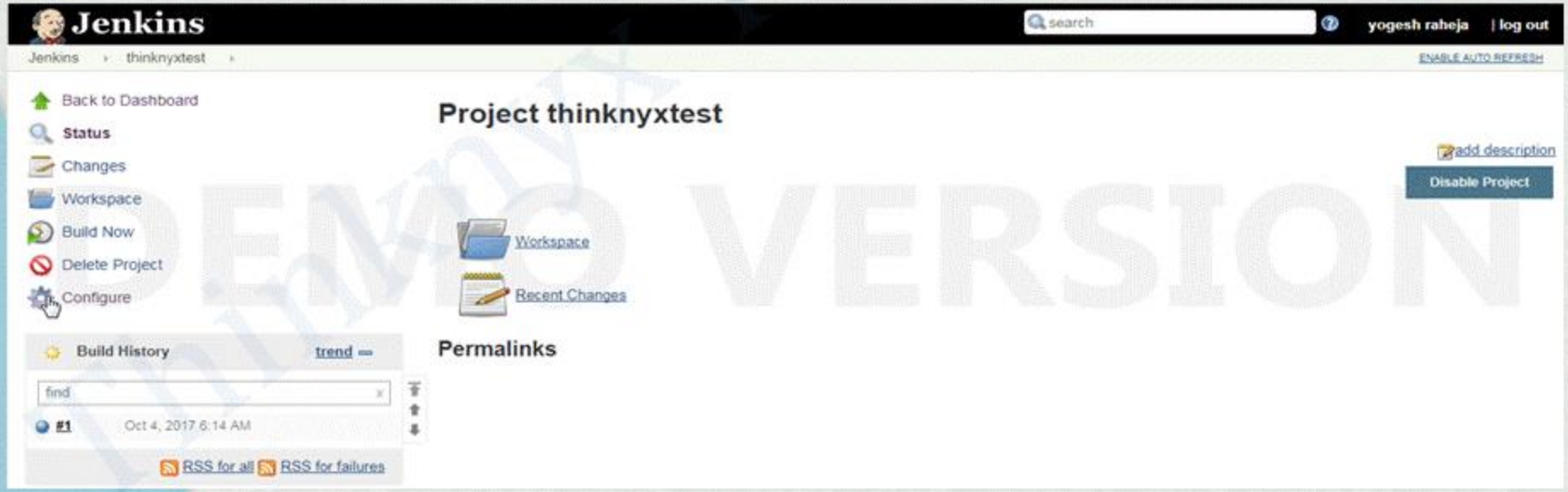
- Maven Release Plug-in Plug-in
- Repository Connector
- Maven Artifact ChoiceListProvider (Nexus)
- Sonar Quality Gates Plugin
- Static Analysis Collector Plug-in
- Build Pipeline Plugin
- Pipeline: Multibranch with defaults
- build-name-setter
- Extended Choice Parameter Plug-In
- Folders Plugin
- Credentials Plugin
- Docker Pipeline
- docker-build-step
- Deploy to container Plugin
- OWASP Dependency-Check Plugin
- Official OWASP ZAP Jenkins Plugin
- Deployment Dashboard Plugin for Jenkins
- OpenShift Deployer Plugin
- Build Monitor View
- mean time to repair plugin
- Logstash
- JIRA Trigger Plugin
- ThinBackup

Creation of First Project

- Click on new item

Executing First Project





Executing First Project

DevOps – Containers

THINKNYX
TECHNOLOGIES

Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

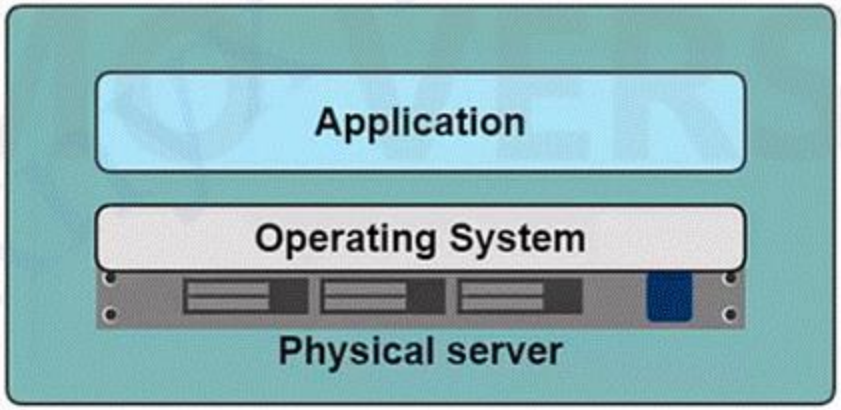
What is Docker

- Before proceeding with the new terms and technologies lets have a look on the history/traditional approaches used for the application since ages.

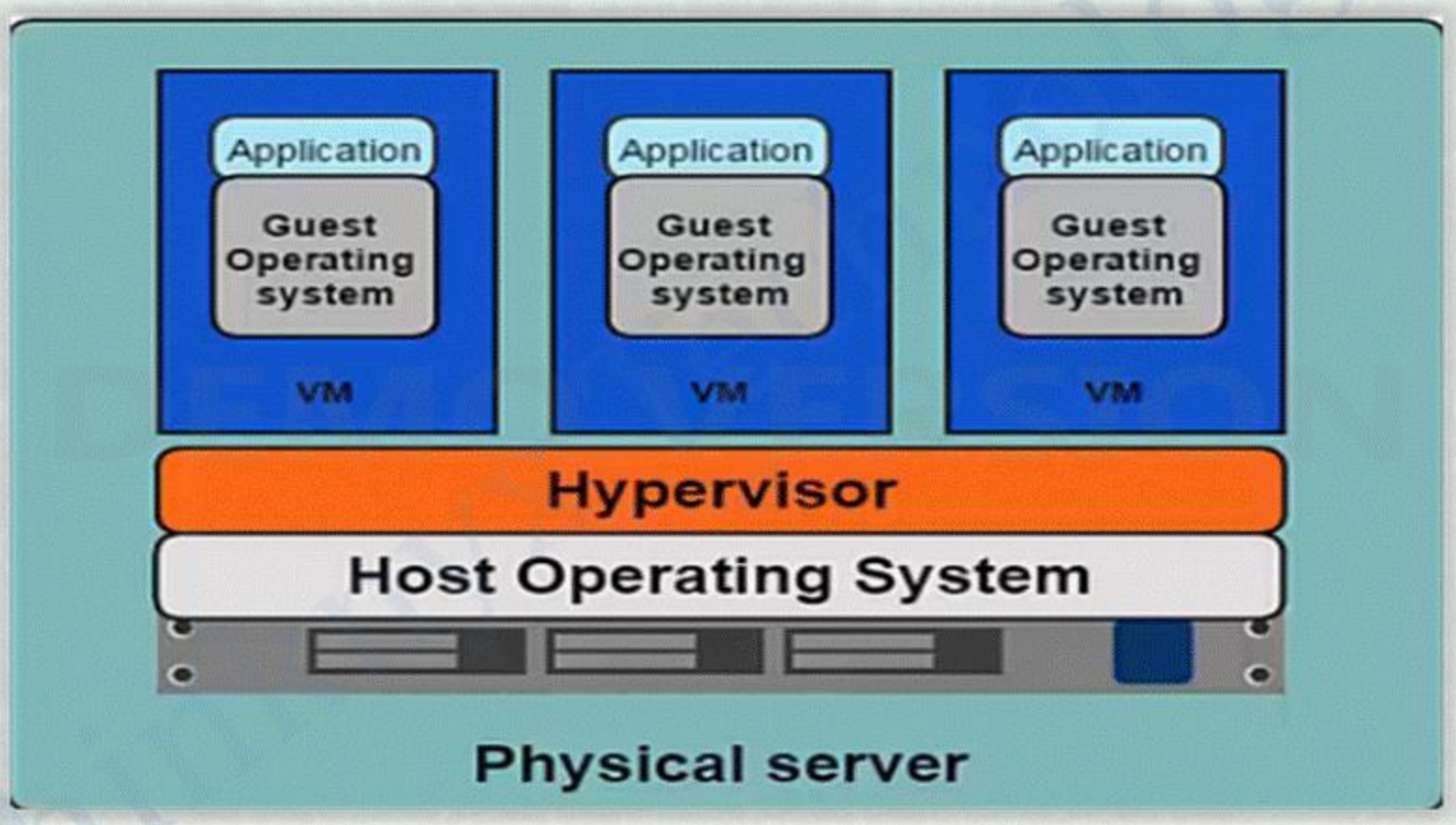
A History Lesson

- In the traditional ages of development and deployment of applications.

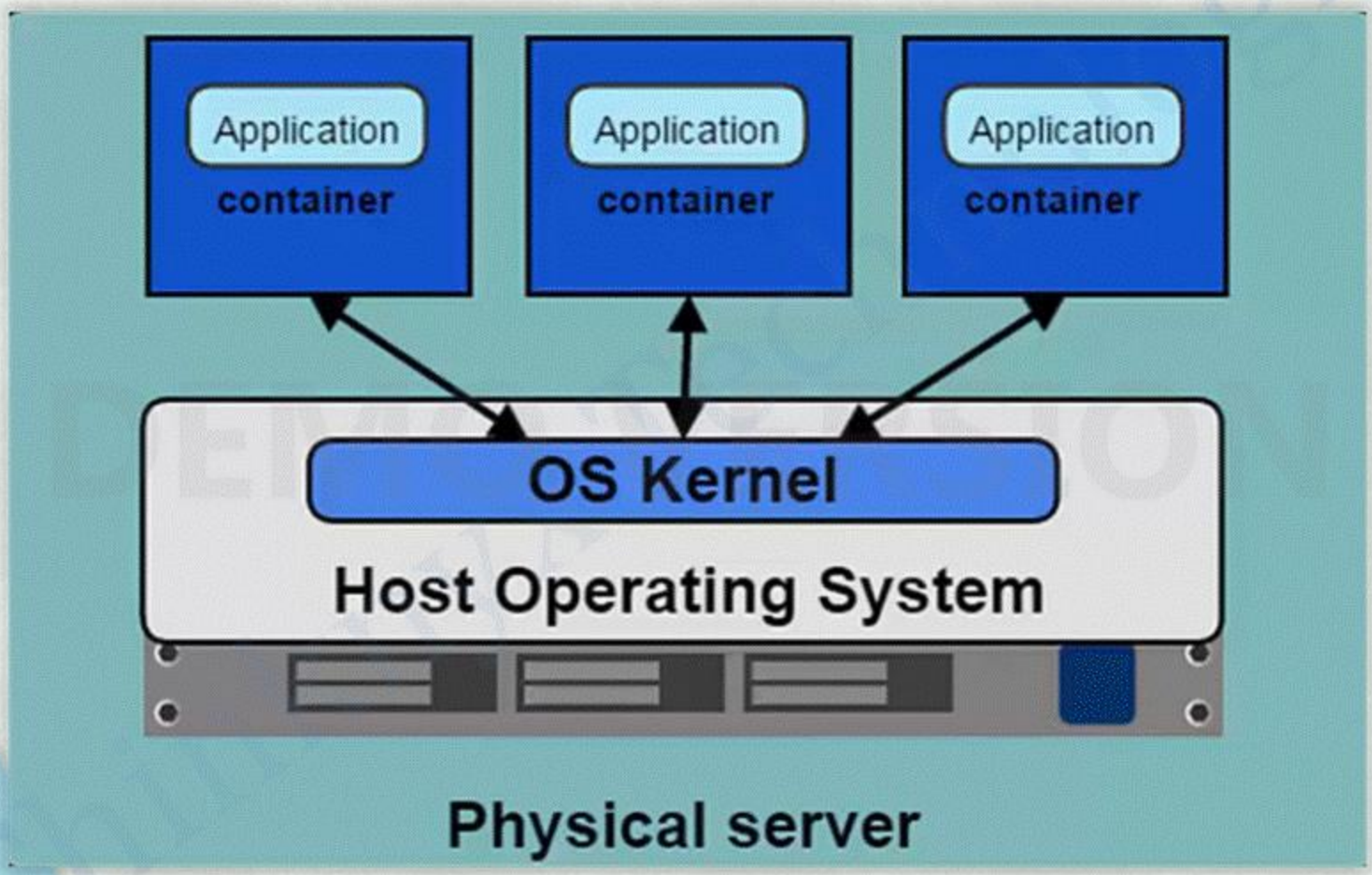
One Application on one physical server



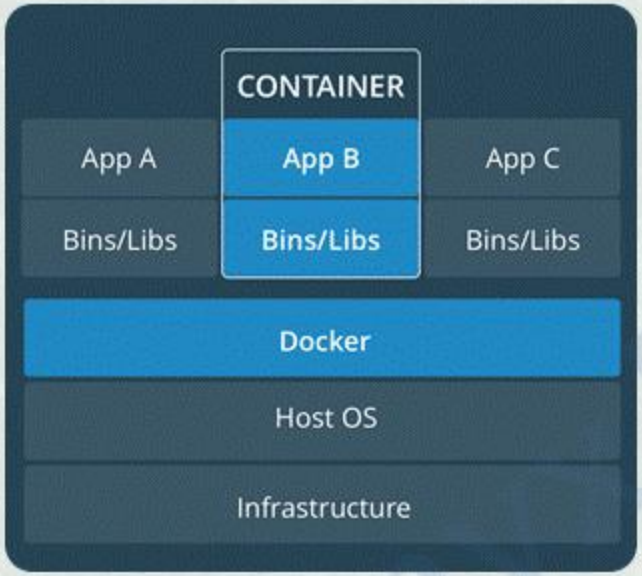
A History Lesson



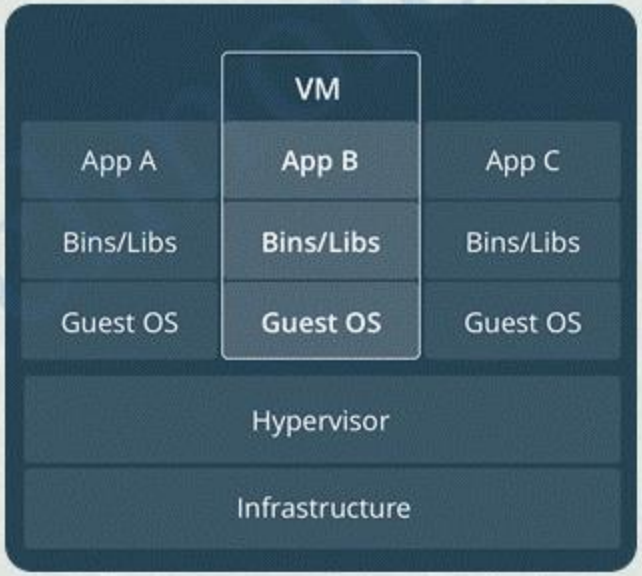
Overview of Containers



Containers VS VM's

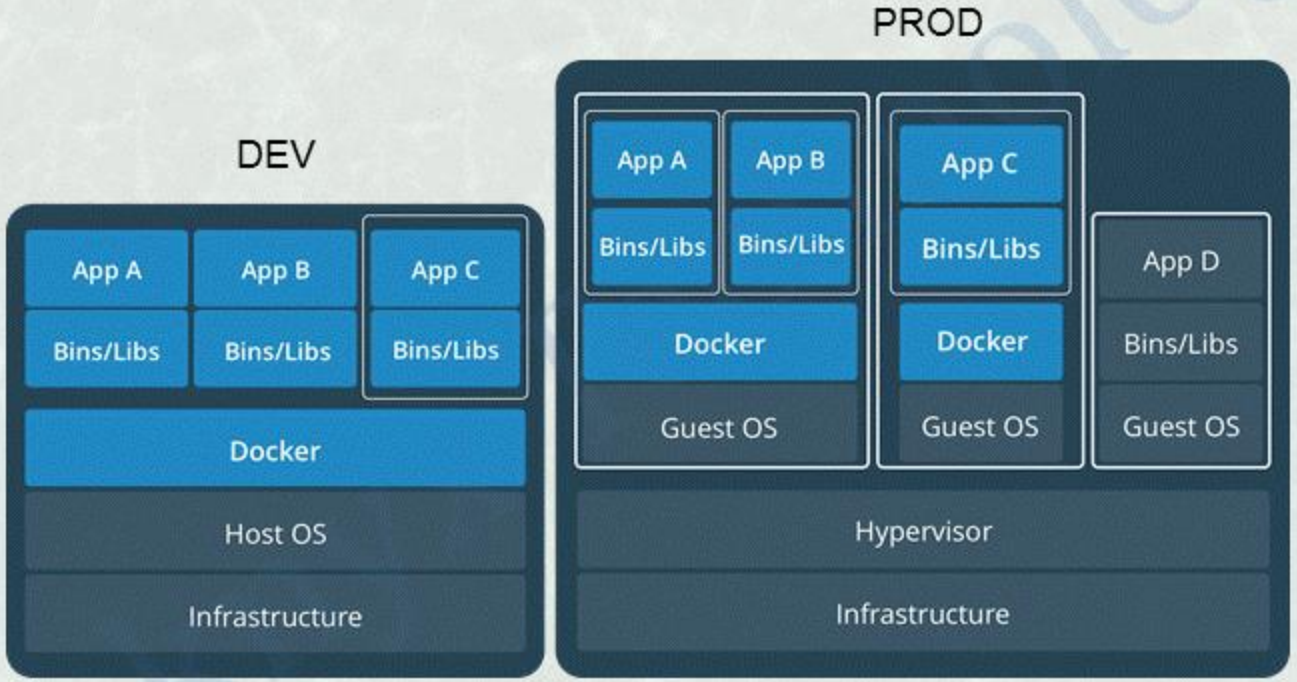


Containers are an app level construct



VMs are an infrastructure level construct to turn one machine into many servers

Containers and VM's together



Containers and VMs together provide a tremendous amount of flexibility for IT to optimally deploy and manage apps.

Results

Thinknyx

Speed

- No OS to boot = applications online in seconds

Portability

- Less dependencies between process layers = ability to move between infrastructure

Efficiency

- Less OS overhead
- Improved VM density

Adoption in Just 4 years

Thinknyx

14M
Docker Hosts

900K
Docker apps

77K%
Growth in Docker job listings

12B
Image pulls Over 390K% Growth

3300
Project Contributors

Docker Components

Docker Overview

Thinknyx

- Docker is an open platform for developing, shipping, and running applications.
- Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.
- With Docker, you can manage your infrastructure in the same ways you manage your applications.
- By using Docker's methodologies for shipping, testing, and deploying, you can reduce time of customer delivery.

Docker Platform

Thinknyx

- Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.

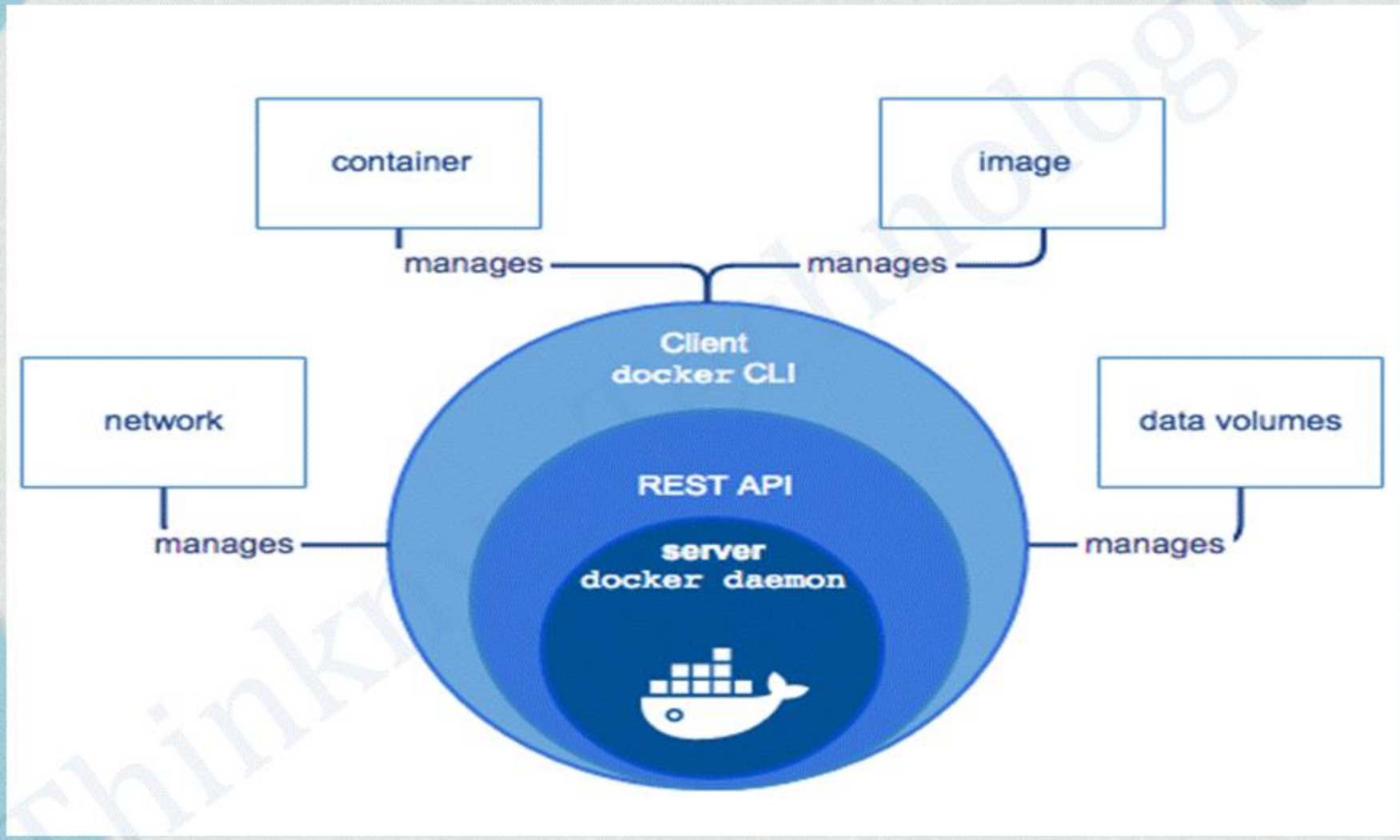
- Because of the lightweight nature of containers, which run without the extra load of a hypervisor, you can run more containers on a given hardware combination than if you were using virtual machines.

Docker Platform

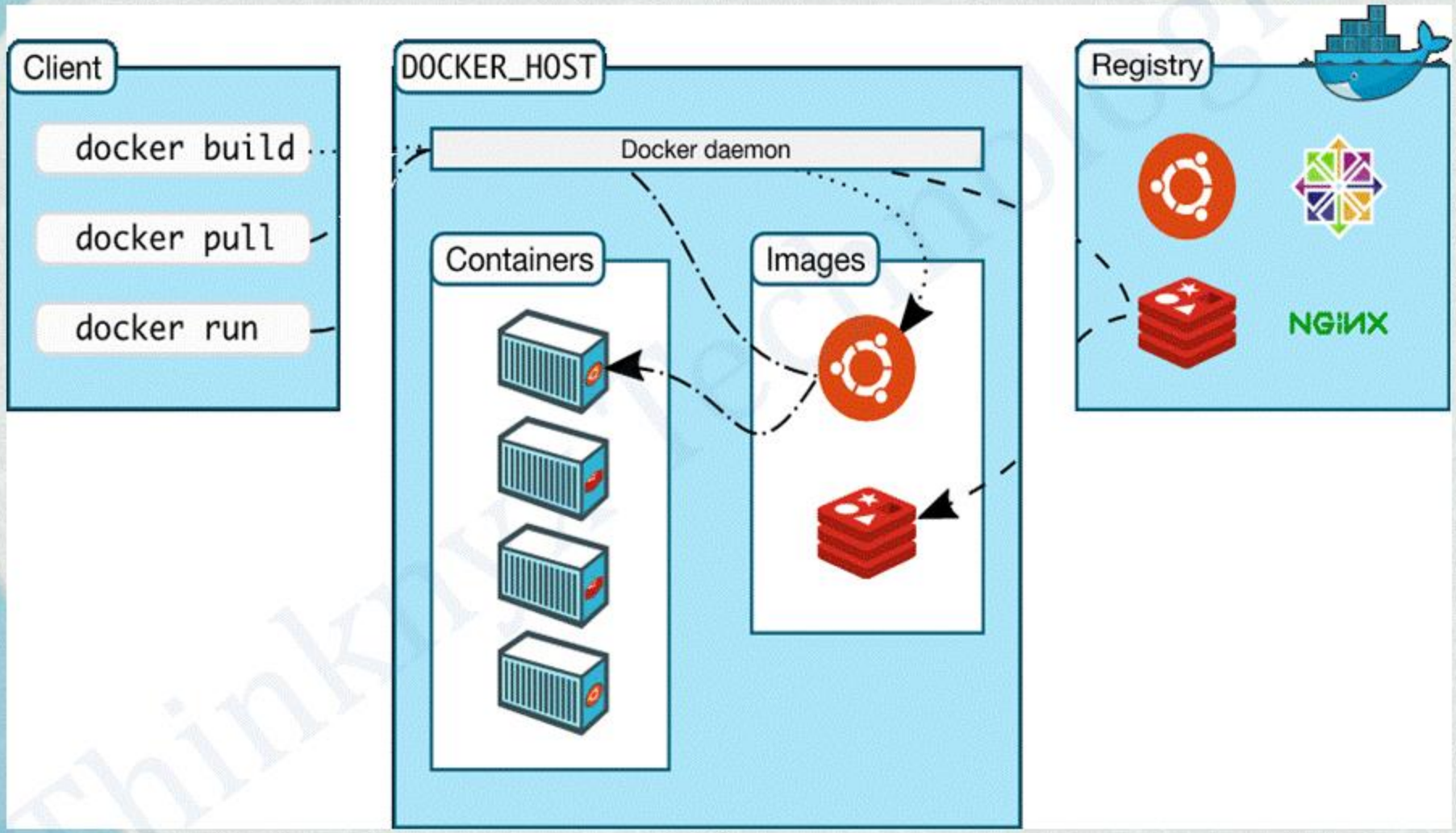
Thinknyx

- Docker provides tooling and a platform to manage the lifecycle of your containers:
  - Encapsulate your applications (and supporting components) into Docker containers
  - Distribute and ship those containers to your teams for further development and testing
  - Deploy those applications to your production environment, whether it is in a local data center or the Cloud

Docker Engine



Docker Architecture



Docker Architecture

Thinknyx

- The Docker Daemon
  - The Docker daemon runs on a host machine. The user uses the Docker client to interact with the daemon.
- The Docker Client
  - The Docker client, in the form of the docker binary, is the primary user interface to Docker.
  - It accepts commands and configuration flags from the user and communicates with a Docker daemon.

Docker Architecture

Image

The basis of a Docker container. The content at rest.

Container

The image when it is 'running.' The standard unit for app service

Engine

The software that executes commands for containers. Networking and volumes are part of Engine. Can be clustered together.

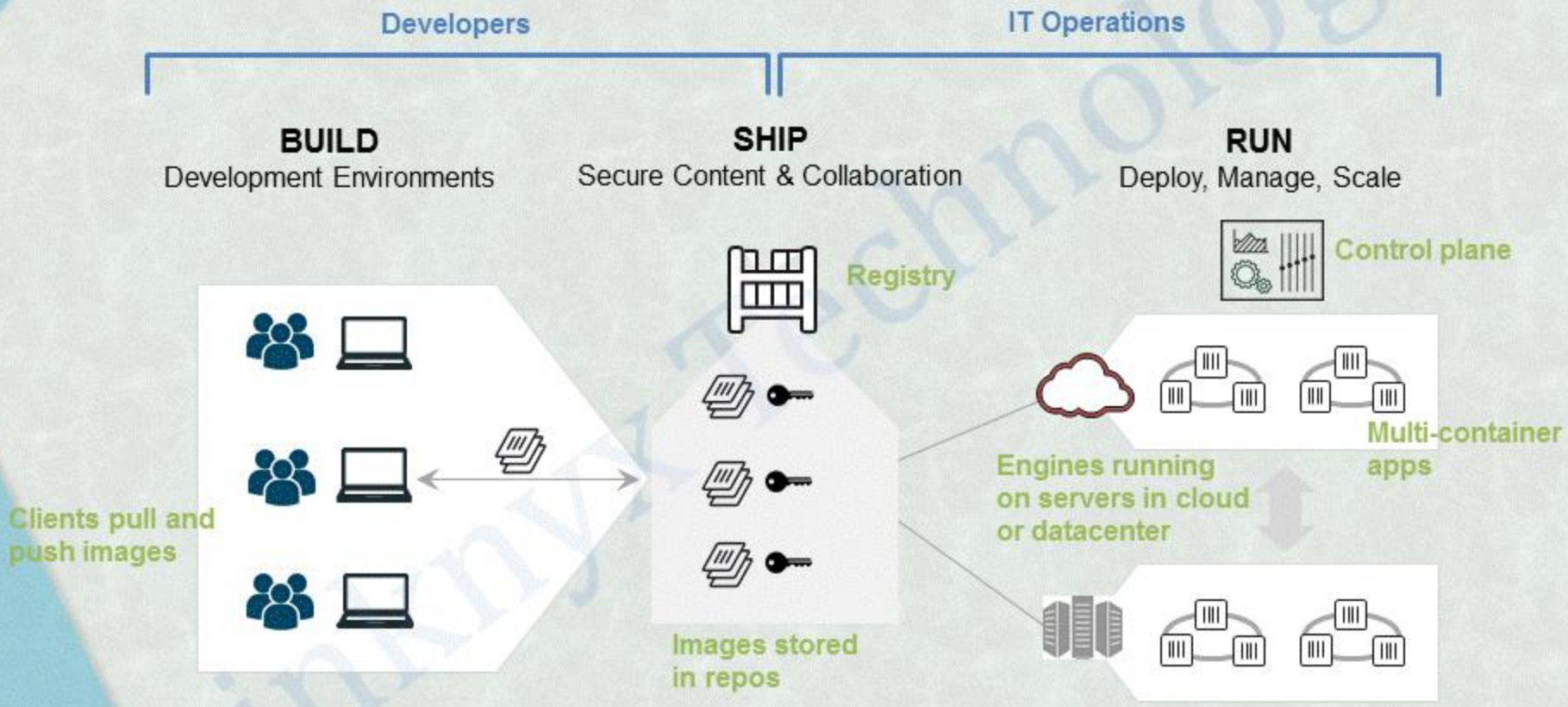
Registry

Stores, distributes and manages Docker images

Control Plane

Management plane for container and cluster orchestration

Container as a Service



Docker Engine Install Demo

- Docker Engine/Client would be installed on Training Environment as demo LAB.

- https://docs.docker.com/engine/installation/linux/centos/

Docker Test

- docker –version
- Test the docker functioning: docker run hello-world

Ansible Fundamentals

THINKNYX
TECHNOLOGIES

Yogesh Raheja
+91-9810344919
yogesh.raheja@thinknyx.com

Introduction to Configuration Management with Ansible

- About Ansible
- Current IT Automation State
- Why Configuration Management?
- Ansible History
- Introducing Ansible
- How Ansible Works
- DataFlow

About Ansible

- IT Automation Software for System Administrators:

- Founded in Feb, 2012
- First commercial product release in 2012
- Multiple in built functional modules
- Multiple Community Members
- 40,000+ Users
- 50,000+ Nodes managed in the largest deployments
- Support for Red Hat, CentOS, Ubuntu, Oracle Linux, MAC, OS, Solaris 10/11, Windows.
- Ansible Controller node Supported on Linux variants only

Current IT Automation State

- **Manually Configure:** Literally logging into every node to configure it.
- **Golden Images:** Creating a single copy of a node's software and replicating that across nodes.
- **Custom One-off Scripts:** Custom code written to address a specific, tactical problem.
- **Software Packages:** Typically all or nothing approach.

Why Configuration Management?

- To provide optimized level of automated way to configure Applications and Software's inside your system.
- Enable you to Discover, Provision, Configure and Manage the systems.
- Developers should be able to use a single command to build and test software in minutes or even in seconds.

Ansible History

- **Michael DeHaan**, the author of the provisioning server application Cobbler and co-author of the Func framework for remote administration, developed the platform called "Ansible" on 20th Feb' 2012.

- It is included as part of the Fedora distribution of Linux, owned by Red Hat Inc., and is also available for Red Hat Enterprise Linux, CentOS, and Scientific Linux via Extra Packages for Enterprise Linux (EPEL) as well as for other operating systems.

- Ansible, Inc. (originally AnsibleWorks, Inc.) was the company set up to commercially support and sponsor Ansible.

- It was acquired by Red Hat in October 2015.

Introducing Ansible

- Ansible is an **open-source IT automation tool** majorly used as Configuration Management tool by system administrators.
- Ansible is a **configuration management system**, capable of maintaining remote nodes in defined states (for example, ensuring that specific packages are installed and specific services are running).
- Ansible is a **distributed remote execution system** used to execute commands and query data on remote nodes, either individually or by arbitrary selection criteria.
- Provides following customer benefits:

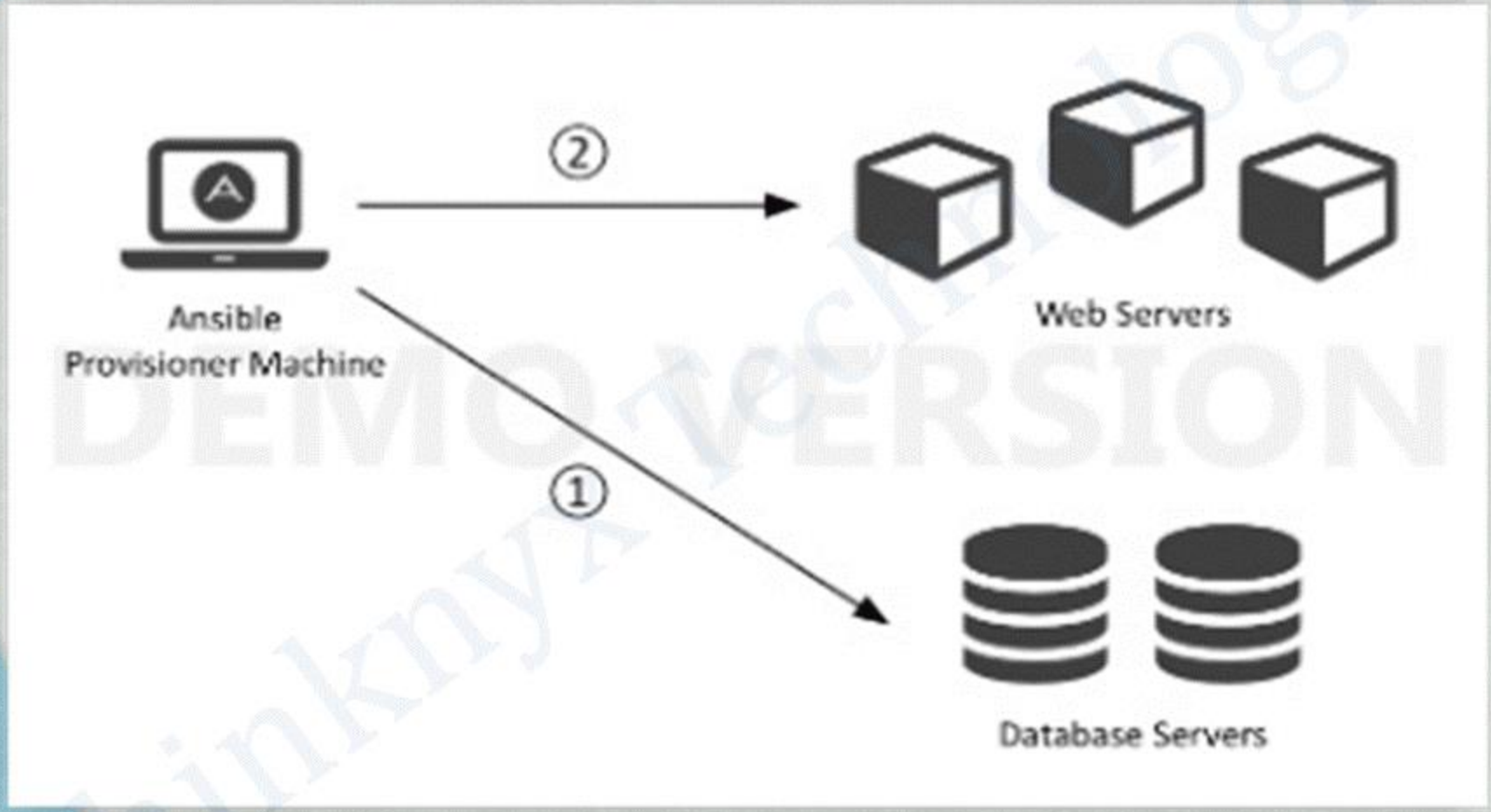
- Productivity / Efficiency
- Simplicity
- Scalability
- Consistency
- Operational efficiency

Ansible Components

- Ansible consists of Agentless Model and majorly have two parts:
- **Controller / Master:** The central configuration server where we will have our all configurations stored.
- **Managed Nodes:** All clients getting configured from Ansible Master.

Note: Python Version is the pre-requisites for Ansible Master.

DataFlow



Ansible Way for Configuration Management

- A light at the end of the tunnel:

```
tasks:
 - name: Creating Yogesh Raheja User
   user: name=yogeshraheja comment="Yogesh Raheja"  state=present
```

```
tasks:
 - name: Creating Raheja Group
   group: name=raheja state=present
```

Ansible Way: Maintaining State

- You provision a node.
- Ansible configures it.
- Ansible maintains the desired state when needed.

Note: You make to sure the state is configured as per environment requirements.

Infrastructure as Code

- Descriptive
- Straightforward
- Transparent

```
[root@yogesh]# cat ntp.yml
---
# This is my Host section
- hosts: localhost
# This is my Task section
 tasks:
 - name: NTP Installation
   yum: name=ntp state=present
 - name: NTP Service
   service: name=ntpd state=started enabled=yes
```

Idempotency

- Ansible enforces in an idempotent way.
- The property of certain operations in mathematics or computer science is that they can be applied multiple times without further changing the result beyond the initial application.
- Able to be applied multiple times with the same outcome.

Ansible Terminology

- **Controller/Master:** The Ansible master is the machine that controls the infrastructure and dictates policies for the servers it manages. It operates both as a repository for configuration data and as the control center that initiates remote commands and ensures the state of your other machines.

- **Managed/Agent Nodes :** The servers that Ansible configures are called Clients/Nodes.

- **Ansible Inventory:** Ansible Inventory represents which machines it should manage using a very simple INI file that puts all of your managed machines in groups of your own choosing.

- **Ansible Adhoc-tasks:** Ansible uses adhoc requests to confirm simple and small tasks on any server right a way without login into the client. The best example is to check the Alive Status for whole managed inventory.

AnsibleTerminology

- **Playbooks:** A structured way to put all of the defines tasks for your application or your whole setup.
- **Modules:** In built functions which executes at the backend to perform underlined tasks in Ansible.
- ***yml files:** describe a set of desired states that a system needs to be in, for example "apache needs to be installed and running".
- **Ansible Tower:** Ansible Tower by Red Hat helps is a web-based solution that makes Ansible even more easy to use for IT teams of all kinds. It's designed to be the hub for all of your automation tasks.

Ansible Terminology

- **Ansible Galaxy:** Ansible Galaxy is a free site for finding, downloading, and sharing community developed roles. Downloading roles from Galaxy is a great way to jumpstart your automation projects..

- **Ansible for Unix/Linux:** The Ansible master communicates and manage Unix/Linux Clients using SSH by default.

- **Ansible for Windows:** Starting in version 1.7, Ansible also contains support for managing Windows machines. This uses native PowerShell remoting, rather than SSH. and uses the "winrm" Python module to talk to remote hosts.

Installation and configuration of Ansible

- Ansible Master/Controller Node
- Installation and Configuration
- Facts
- Ansible Clients
- Ansible Inventory
- Execution Test
- Ansible commands

Ansible Master Role

- The Ansible **master** is the machine that controls the infrastructure and dictates policies for the servers it manages.
- Currently Ansible can be run from any machine with Python 2.6 or 2.7 installed (Windows isn't supported for the control machine).
- This includes Red Hat, Debian, CentOS, OS X, any of the BSDs, and so on.

Ansible Master Role

- To install the Ansible Master, we need to install EPEL repository package:
  - Ansible Repository

http://fedoraproject.org/wiki/EPEL

Note: If internet connectivity is there just do:

- wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-6.noarch.rpm

- Pre-installation
- Assign a hostname to your machine(Master) and make that name persist across reboot.

Ansible Master Installation and Configuration

- Ansible Master would be installed on Training Environment as demo LAB.

Lets do it to get familiar with the Ansible Master Installation Process.

Ansible Master Installation and Configuration

- yum install ansible
- rpm –qa | grep –i ansible
- ansible --version
- Default Configuration file is /etc/ansible/ansible.cfg
- Default Inventory file is /etc/ansible/hosts

Ansible Master Configuration

- Edit /etc/ansible/ansible.cfg for any Master Configurations
- Default options are fine
- All parameters can be overridden in ansible-playbook or with command line flags.

Special Shortcut: cat /etc/ansible/ansible.cfg | grep "^\["

Ansible Clients

- As Ansible uses Agentless mechanism so no packages are required to be installed on any client.

- We need a way to communicate, which is normally ssh. By default this uses sftp. If that's not available, you can switch to scp in ansible.cfg.

- We also need Python 2.4 or later. If you are running less than Python 2.5 on the remotes, you will also need below package to be installed on the server:

  "python-simplejson"

Ansible Authentication

- As Ansible is using SSH by default during communication, this communication connection supports both:

- **Password Based Authentication:** Password based authentication is acceptable if your environment is small and easily manageable. But it become very difficult to work with password based authentications once you scale your environment. Password based authentication is only useful in Engineering Labs or Test Labs or Playbook creations tests.

- **Key Based Authentication:** Key based Authentication is adopted in Enterprise Environments. Here we create one generic user and amend the keys of the generic user in Managed Nodes for Key Based - Password less Authentication. This is a onetime task and can be used with any number of servers.

Ansible Inventory

- Ansible Inventory is a text-based list of individual servers and/or group of multiple servers.
- By default the Ansible Inventory location is **"/etc/ansible/hosts"**.
- You may have multiple Inventory files.
- Ansible Inventory can have Host Name or IP Address or Combination of both.
- Ansible provides the flexibility to pull inventory from Dynamic or Cloud sources with the help of scripts.
- You can specify a different inventory file using the -i <path> option on the command line.

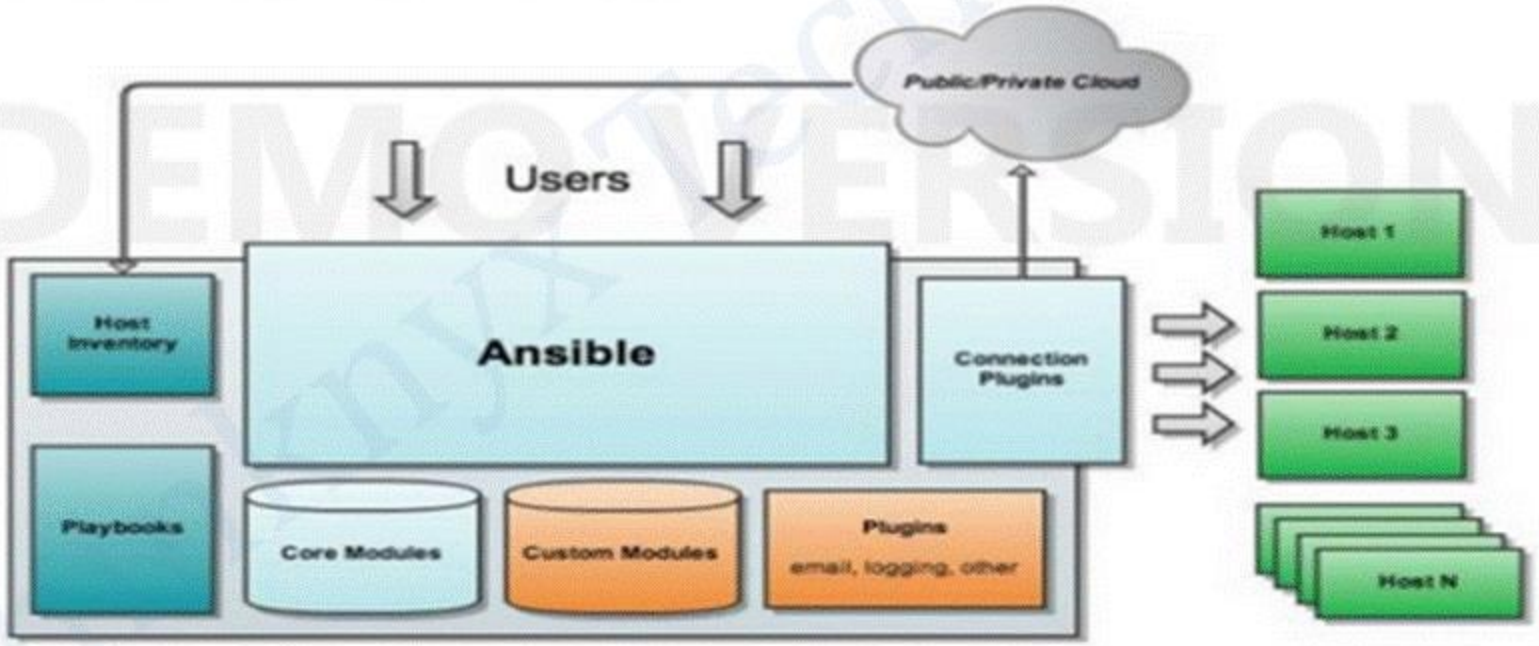
Ansible Inventory File

- cat /etc/ansible/hosts

```
mail.example.com
192.168.163.80

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com

[databases]
db-[a:f].example.com
```

Ansible Communication

Ansible Architecture



Ansible Communication test

- Communication checks with password authentications:

```
[root@yogesh-controller ~]# ansible centos-managed -m ping --ask-pass
SSH password:
centos-managed | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Ansible Communication test

- Communication setup and checks with key based password less authentications:
- Execute ssh-keygen
- Make sure to have 0600 permissions on id_rsa and 0644 on id_rsa.pub.
- Copy id_rsa.pub (public key) to Managed Node for password less authentication at "/root/.ssh/authorized_keys" with permissions 0600.
- Execute the ansible ping test without –ask-pass

Ansible Modules

- Modules are the Basic Building Block of Ansible.
- These are the readymade tools to perform various tasks and operations on "Managed Nodes".
- Modules can be used with Ansible Ad-Hoc Remote Executions and/or Playbooks as a core building blocks.
- Ansible Ships with multiple in-built Modules (approx. 1000+ in Ansible 2.3).
- Can be used for Standalone servers, Virtual Machines and for any Public/Private Cloud Instances.

Ansible Modules

- Two types of Ansible Modules: Core Modules & Custom Modules.
- Robust Module Documentation on website.
- Command line utility on Module information and usage.
- CLI utility ansible-doc on “Controller Node”.
- Execute ansible-doc -l to list all available Modules.
- Execute ansible-doc <module-name> to find all details about Modules.
- For GUI refer **“http://docs.ansible.com/ansible/modules_by_category.html”**

Ansible Adhoc Executions

- Easy to learn ad-hoc command line utility - “ansible”.
- Quick On-demand tasks on “Managed Nodes”.
- 1 to 1 approach, single ad-hoc command is used to perform single operation.
- Multiple ad-hoc operations require multiple “ansible” ad-hoc run.
- Ad-hoc execution syntax.
- Ansible Ping Communication Test with “Managed Nodes”.
- Various real time examples with ad-hoc execution.

Ansible Execution Test

- Let's try executing a remote command, before that make sure you have out an entry of the host in "/etc/ansible/hosts"

- Connect to the master and type:

ansible <host-name/IP> -m ping --ask-pass

ansible "*" -m ping --ask-pass

- First argument = target client
- Second argument = function to execute
- Other arguments = params for the function

Ansible Execution Test

- There are a bunch of predefined :

«**Ad-Hocmodules**»

«**execution modules**»

«**Ad-Hocmodules**»: ansible all/"Client or Group“ -a “<adhoc-command>" --ask-pass

«**execution modules**» ansible all/"Client or Group“ -m <module_name> -a <arguments>

- Note: To list all Ansible Modules run below command:

ansible-doc -l

Ansible Execution Test

- For example, executing a shell commands:

ansible 192.168.74.51 -a "ls -l /tmp" --ask-pass

ansible 192.168.74.51 -a "uname -a" --ask-pass

ansible 192.168.74.51 -a "cat /etc/redhat-release" --ask-pass

ansible 192.168.74.51 -a 'service ntpd status' --ask-pass

Ansible Execution Test

- For example, executing Ansible Modules:

ansible 192.168.74.51 -m ping --ask-pass

ansible 192.168.74.51 -m user -a "name=rahejayogesh state=present" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/tmp/yogeshraheja mode=777 group=rahejayogesh state=touch" --ask-pass

ansible 192.168.74.51 -m service -a "name=ntpd state=stopped" --ask-pass

ansible 192.168.74.51 -m file -a "path=/var/yog/rah/test mode=777 state=directory" --ask-pass

Ansible Ad-Hoc Labs

- Check the uptime using Ansible Ad-hoc execution with password-less authentication
- Check OS release using Ansible Ad-hoc execution with password-less authentication
- Install a package named “telnet” on managed host
- Create a user named “yogeshraheja” with bash shell having user id of 9999 on managed host
- Create a file named “/tmp/myfile” with all permissions and user and owner root on managed host

Ansible Ad-Hoc Labs

- Start a service called “nfs” on managed host

- Run below simple script using Ansible host:

- cat /var/tmp/test.sh
- #!/bin/sh
- useradd testuser
- touch /tmp/testfile
- mkdir /tmp/testdir
- echo “my test output”

Ansible Commands

- **Ansible**: Ansible is an extra-simple tool/framework/API for doing 'remote things' over SSH. This is the adhoc command that allows for a 'single task playbook' run.
- **Ansible-doc**: Ansible-doc displays information on modules installed in Ansible libraries. It displays a terse listing of modules and their short descriptions provides a printout of their DOCUMENTATION strings, and it can create a short "snippet" which can be pasted into a playbook..
- **Ansible-playbook:** Ansible playbooks are a configuration and multinode deployment system. Ansible-playbook is the tool used to run them.
- **Ansible-vault:** ansible-vault can encrypt any structured data file used by Ansible.
- **Ansible-galaxy**: Ansible Galaxy is a shared repository for Ansible roles. The ansible-galaxy command can be used to manage these roles, or for creating skeleton framework for roles you'd like to upload to Galaxy.

THANK YOU

For any queries or questions, please contact:

support@thinknyx.com
yogesh.raheja@thinknyx.com
yogeshraheja07@gmail.com