



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

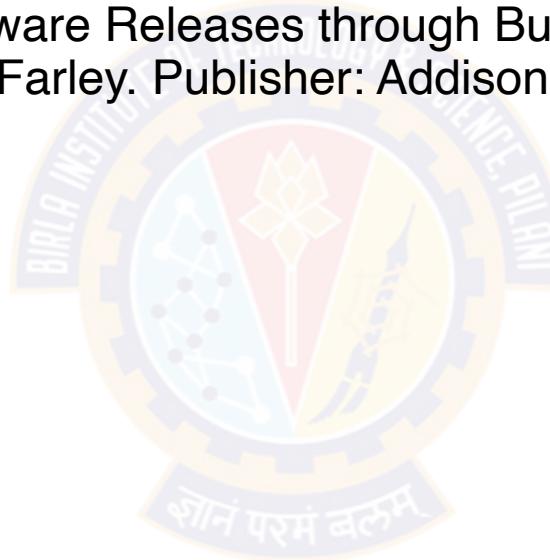
---

Assistant Professor  
BITS Pilani

# Text Books

## T1 & T2

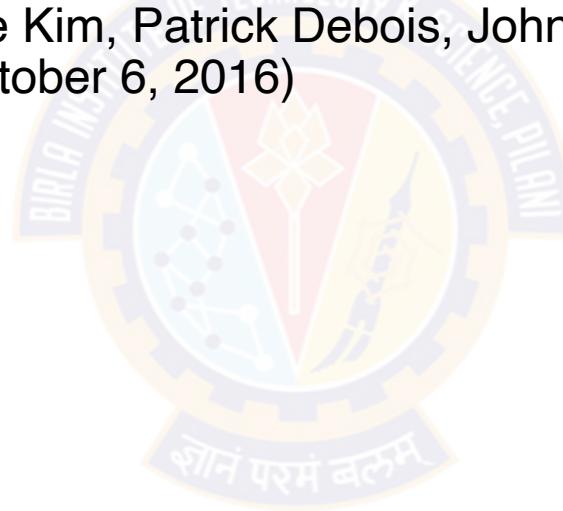
- DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu , Publisher: Addison Wesley (18 May 2015)
- Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Publisher: Addison Wesley, 2011



# Reference Books

## R1 & R2

- Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis , Ryn Daniels. Publisher: O'Reilly Media, June 2016
- The DevOPS Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations by Gene Kim, Patrick Debois, John Willis, Jez Humble, John Allspaw. Publisher: IT Revolution Press (October 6, 2016)



# Agenda

## Introduction

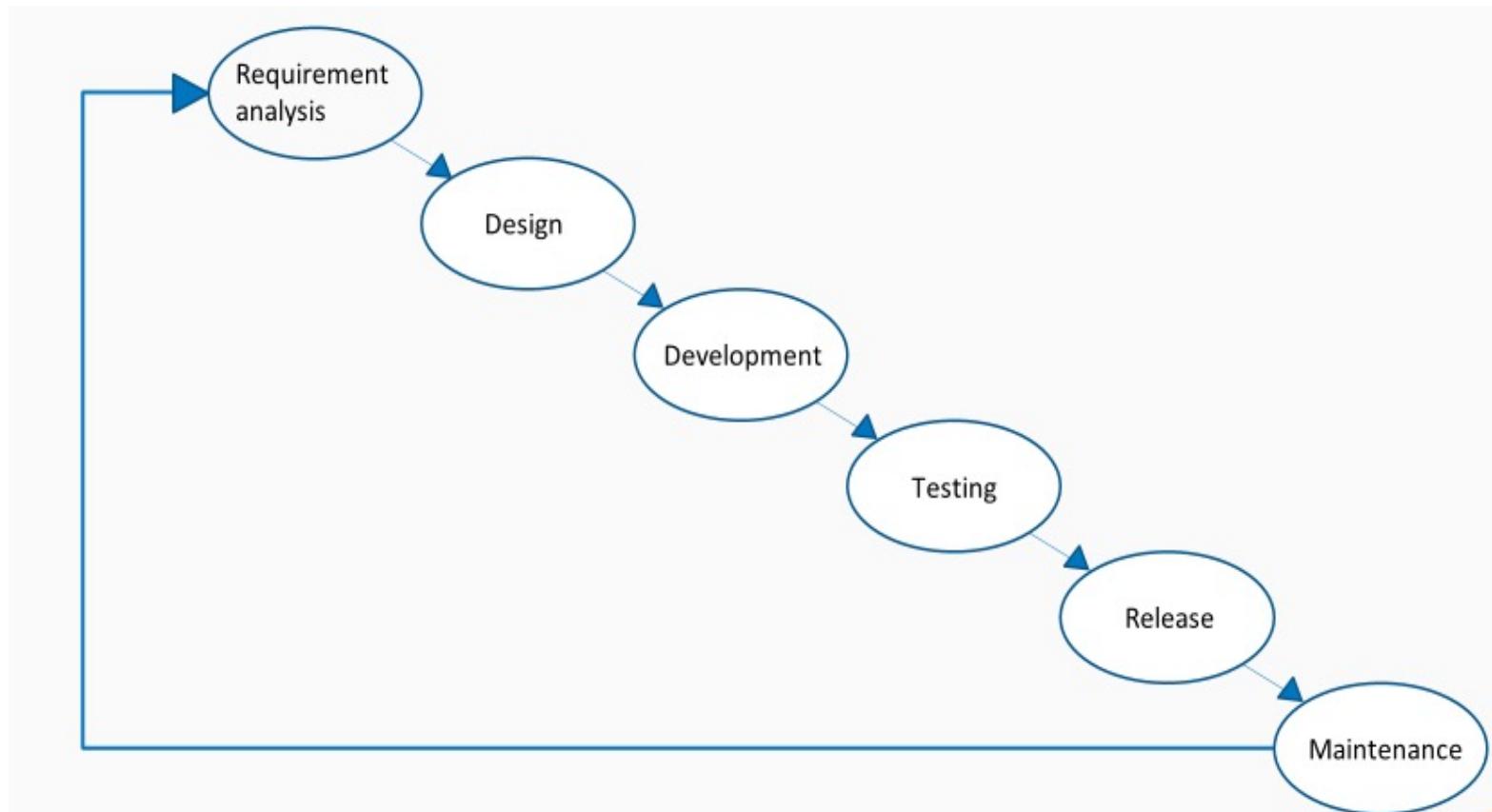
- Software development lifecycle
- The Waterfall approach : Advantages & Disadvantages
- Agile Methodology
- Operational Methodologies: ITIL



# Software Development Life Cycle

## SDLC Phases

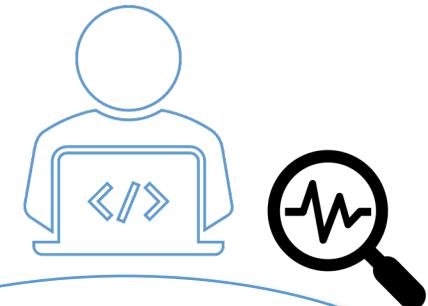
Diagram of our day-to-day activity as software engineers



# Software Development Life Cycle

## Requirement Analysis

- Encounter majority of problems
- Find common language between people outside of IT and people in IT
- Leads to different problems around terminology
- Business flow being capture incorrectly
- Iterative approach



Requirement Analysis

# Software Development Life Cycle

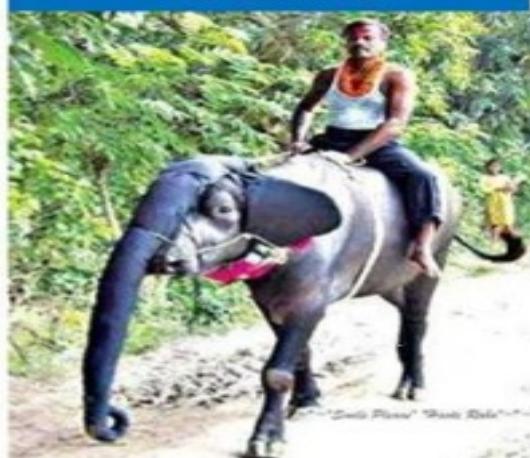
## Requirement Analysis Contd..

Customer requirement



1. Have one trunk
2. Have four legs
3. Should carry load both passenger & cargo
4. Black in color
5. Should be herbivorous

Our Solution



1. Have one trunk
2. Have four legs
3. Should carry load both passenger & cargo
4. Black in color
5. Should be herbivorous

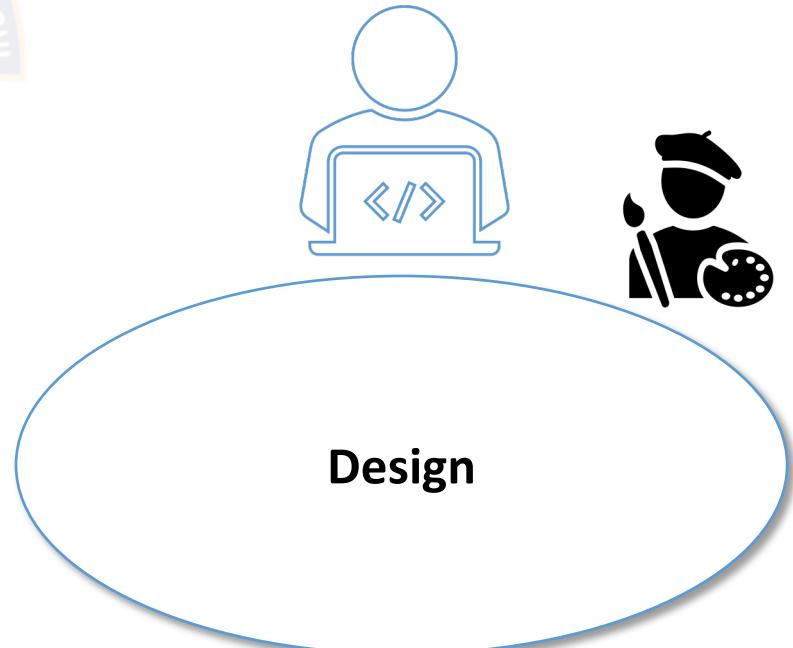
Our Value add:

Also gives milk 😊

# Software Development Life Cycle

## Design

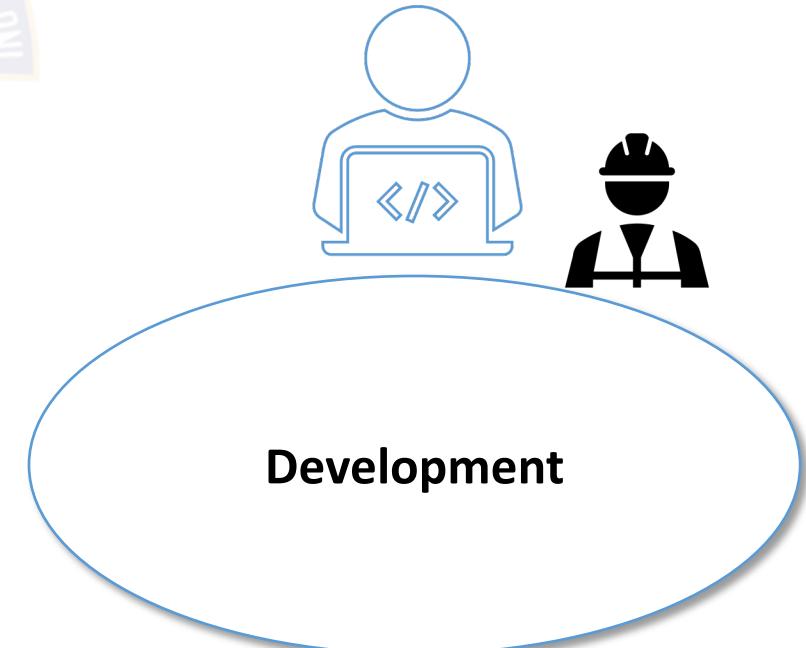
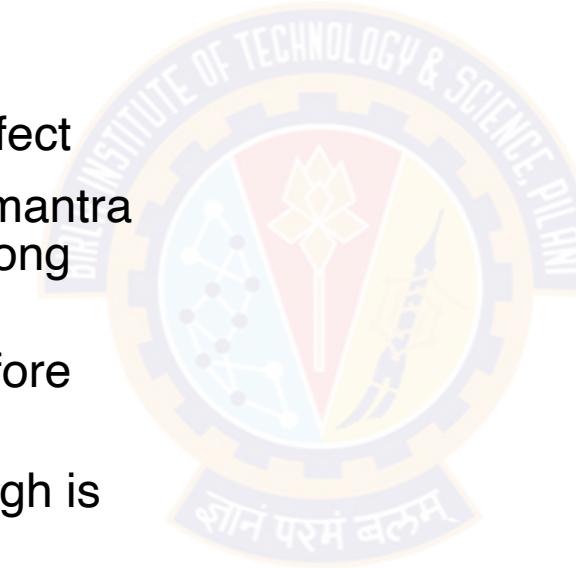
- Design our flows in language that IT crowd can understand straight away
- Overlaps with requirement analysis
- Desirable as diagrams are perfect middle language that we are looking for
- Minimal Viable Product



# Software Development Life Cycle

## Development

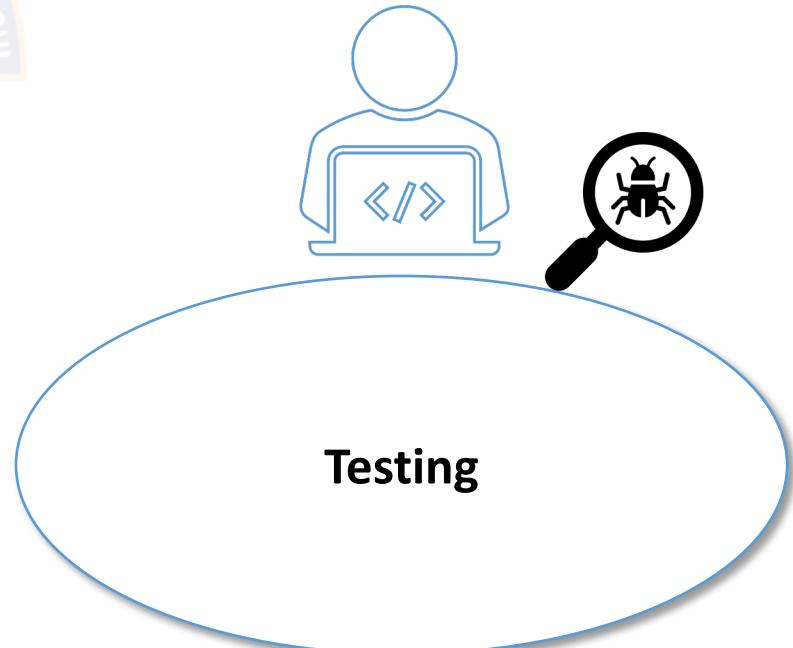
- Software is built
- Builds technical artifacts that work and according to potentially flawed specification
- Our software is going to be imperfect
- Deliver early and deliver often is mantra followed to minimize impact of wrong specification
- Stakeholders can test product before problem is too big to be tackled
- Involving stakeholders early enough is good strategy
- No matter what we do, our software has to be modular so we can plug and play modules in order to accommodate new requirements



# Software Development Life Cycle

## Testing

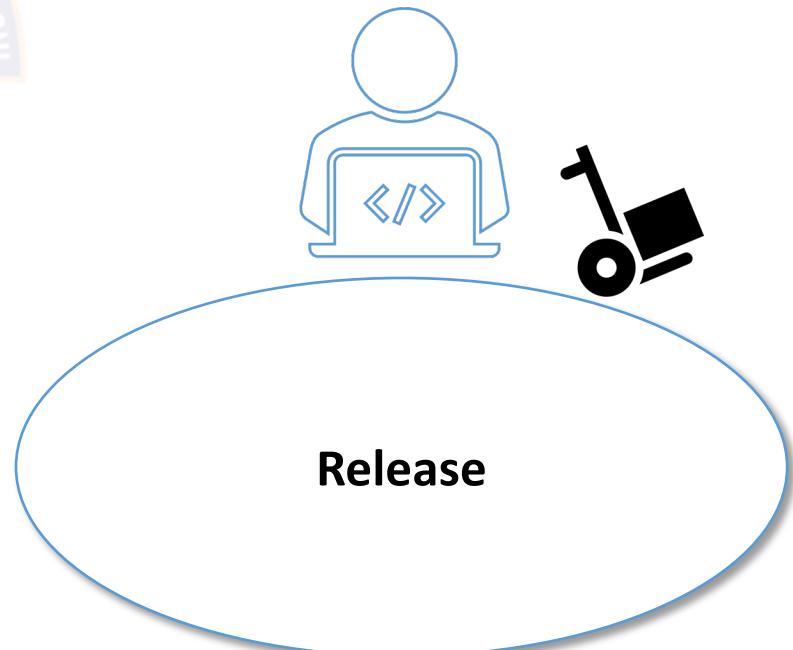
- Continuous Integration server will run testing and inform us about potential problems in application
- Depending on complexity of software, testing can be very extensive
- Continuous integration server focuses on running integration and acceptance



# Software Development Life Cycle

## Release

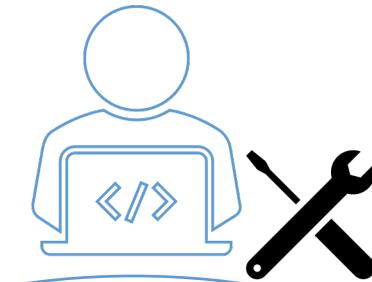
- Deliver software to what we call production
- There will be bugs and reason why we planned our software to be able to fix problems quickly
- Create something called as Continuous delivery pipelines
- Enables developers to execute build-test-deploy cycle very quickly
- Deploy = Release



# Software Development Life Cycle

## Maintenance

- There are two types of maintenance evolutive and corrective
- Evolutive maintenance – evolve software by adding new functionalities or improving business flows to suit business needs
- Corrective maintenance – One where we fix bugs and misconceptions
- Minimize latter but we can not totally avoid

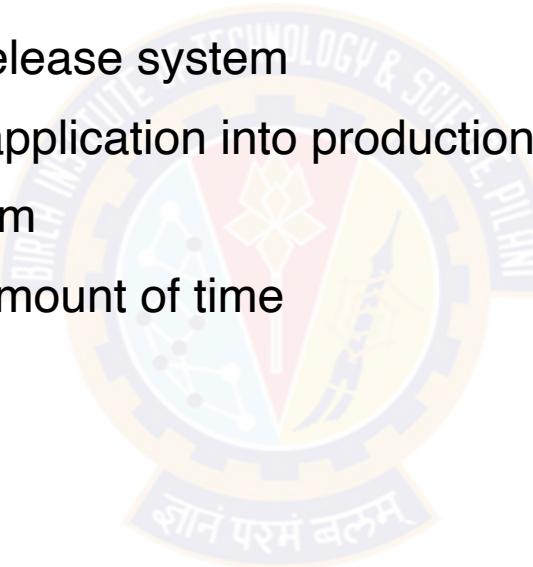


**Maintenance**

# Case Study

## The Power of Automated Deployment from Continuous Delivery Book

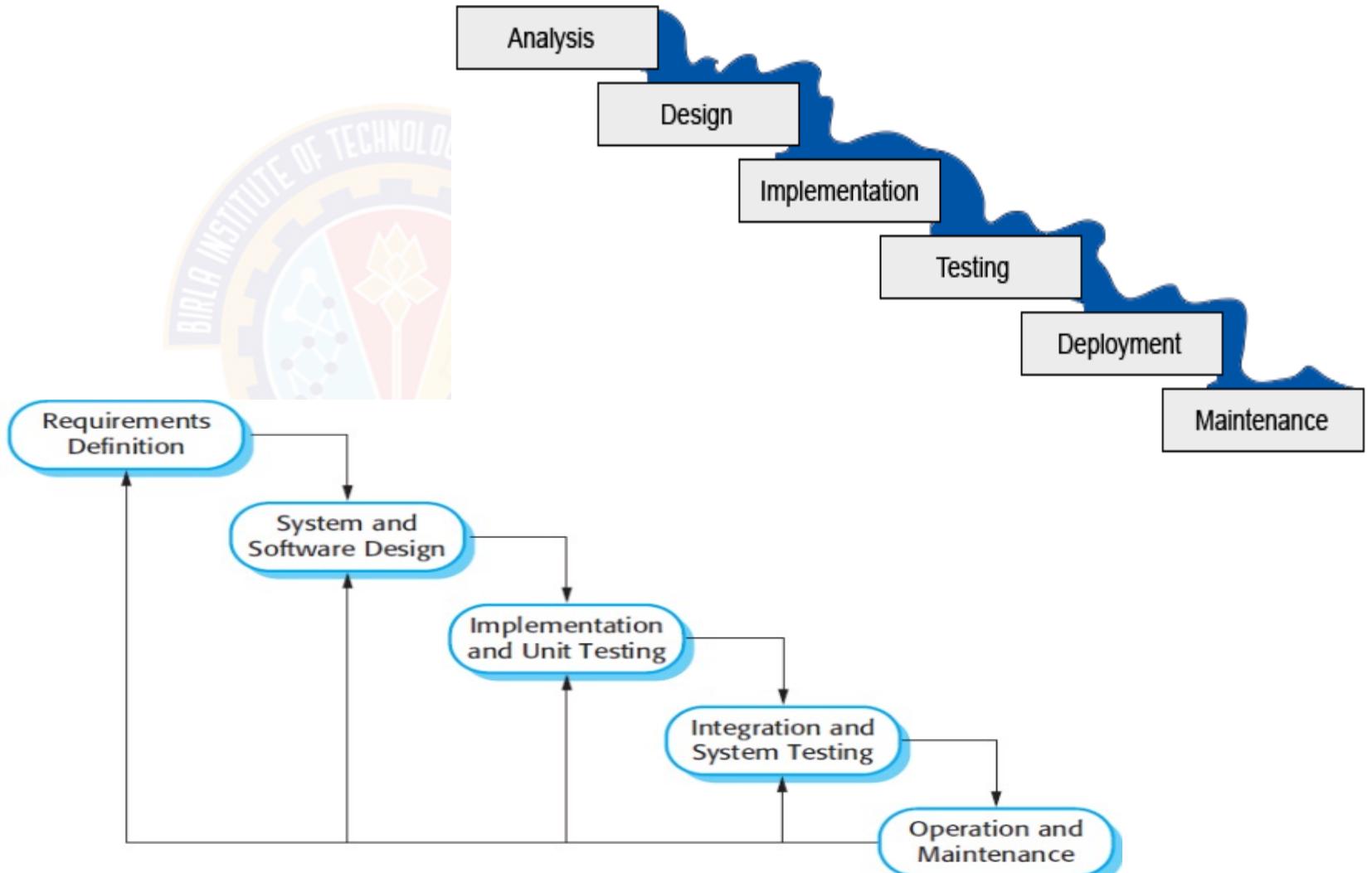
- Large team dedicated to release
- High level of intervention
- Automated build, deploy, test and release system
- Only seven seconds to deploy the application into production
- Successful deployment of the system
- Roll back the change in the same amount of time



# Waterfall Model

## Waterfall Model and Feedback Amendment in Waterfall Model

- Classical Life cycle /Black Box Model
- Sequential in Nature
- Systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software



# Waterfall Model Contd..

## Advantages

- Easy to use and follow
- Cost effective
- Each phase completely developed
- Development processed in sequential manner, so very less chance of rework
- Easy to manage the project
- Easy documentation



# Waterfall Model Contd..

## Waterfall Model Problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway
- In principle, a phase has to be complete before moving onto the next phase
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process
  - Few business systems have stable requirements
- Does Waterfall ends????
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work

# Waterfall Model Contd..

## When to apply Waterfall?

**Stable Requirements  
Fixed Schedule  
Fixed Budget**

**Waterfall  
(Linear)**



# Need of Agile

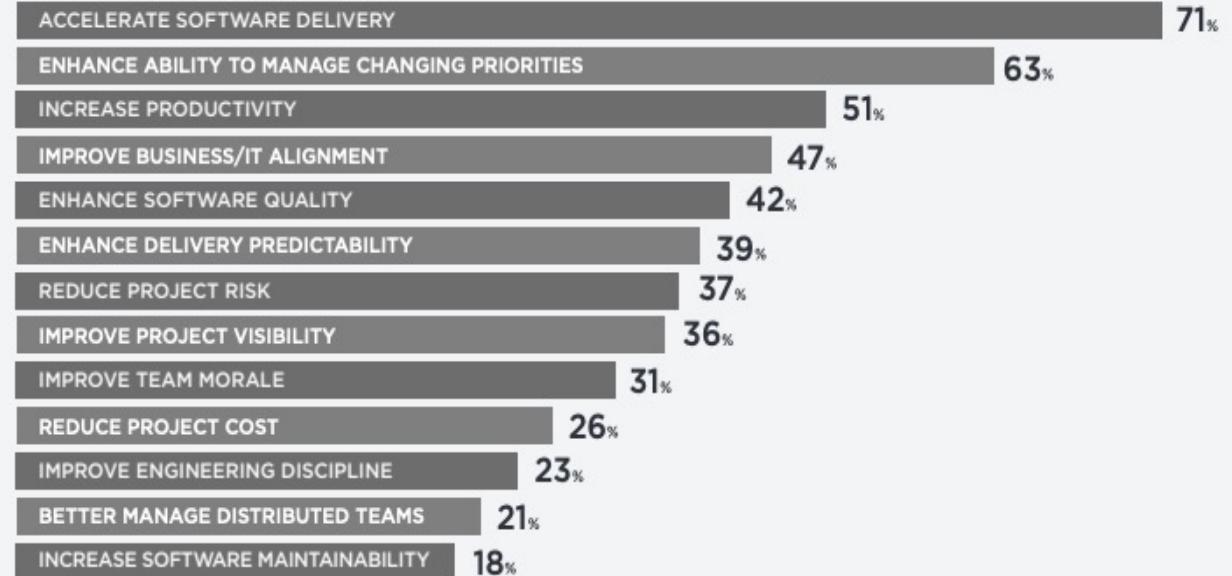
## Why Agile?

- The project will produce the wrong product
- The project will produce a product of inferior quality
- The project will be late
- We'll have to work 80 hour weeks
- We'll have to break commitments
- We won't be having fun
- Storm called Agile
- According to VersionOne's State of Agile Report in 2017 says 94% of organizations practice Agile, and in 2018 it reported 97% organizations practice agile development methods



### REASONS FOR ADOPTING AGILE

Accelerating software delivery and enhancing ability to manage changing priorities remain the top reasons stated for adopting Agile. Respondents indicated this year that reasons for adoption were less about reducing project cost (26% compared to 41% last year), and more about reducing project risk (37% compared to 28% last year).



\*Respondents were able to make multiple selections

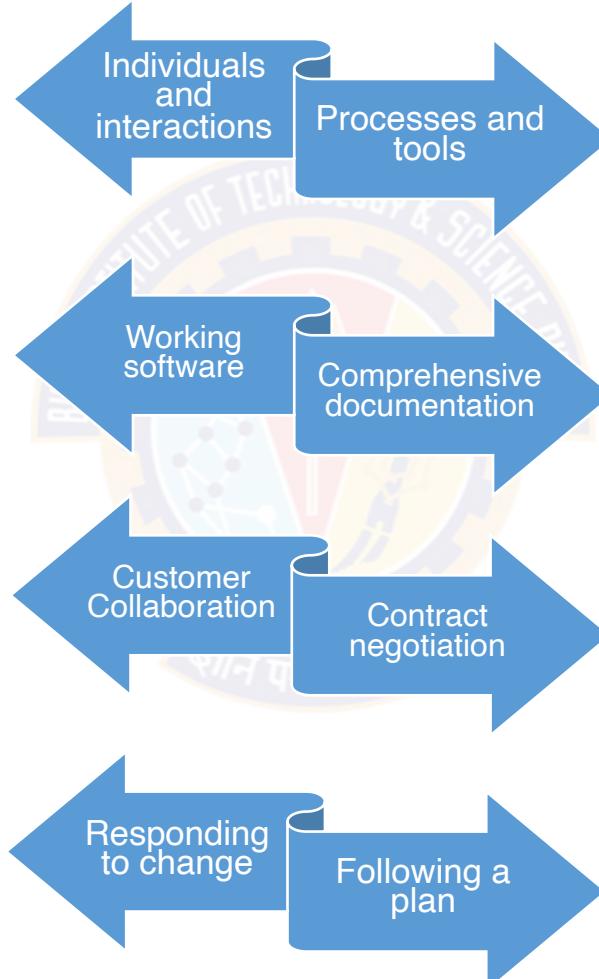
# Principles of Agile Methodology

## Twelve Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
4. Business people and developers must work together daily throughout the project
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
7. Working software is the primary measure of progress
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity--the art of maximizing the amount of work not done--is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

# Pillars of Agile Methodology

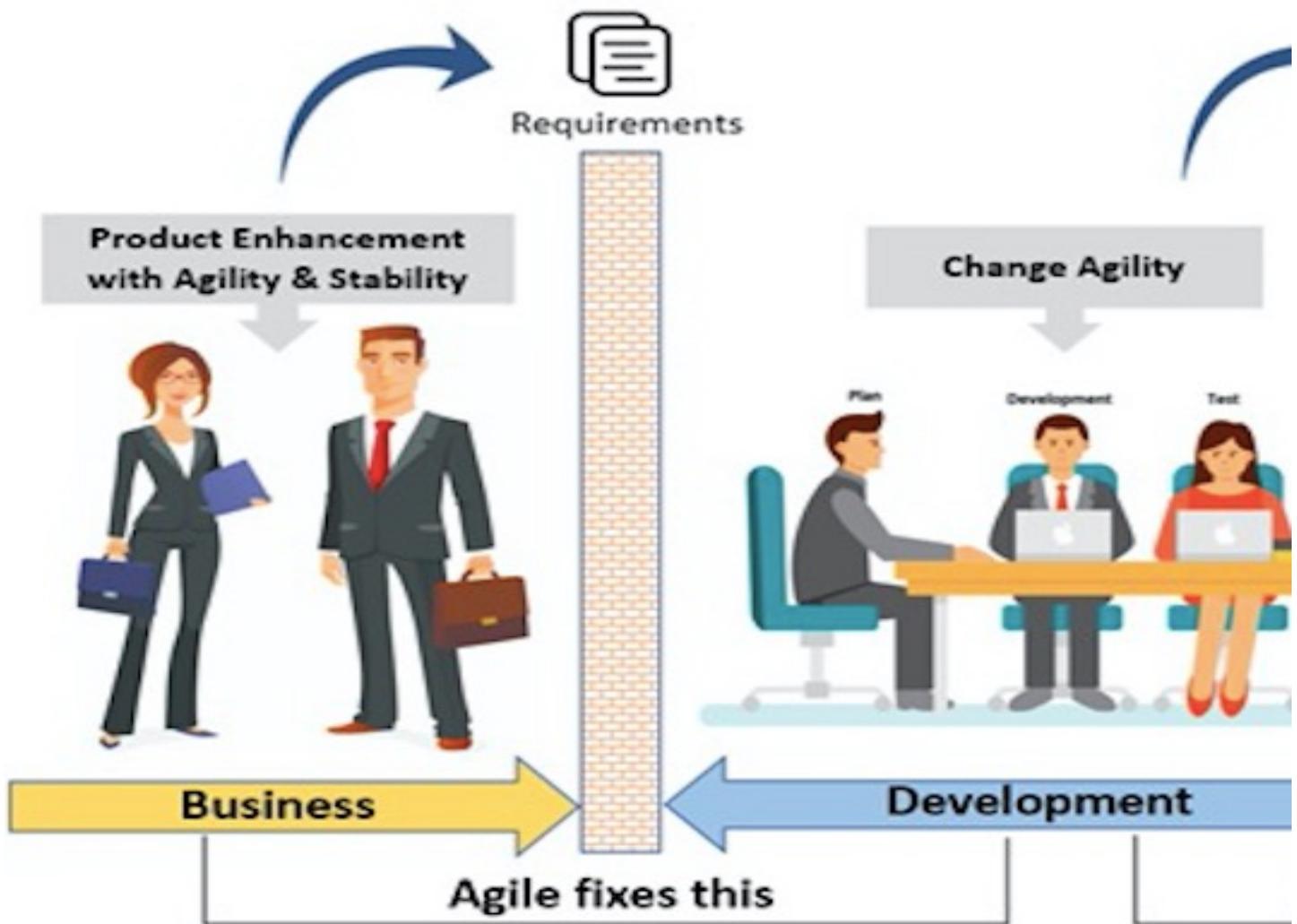
Agile focuses on



# Agile Brings What??

## Being Agile

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed



# Agile Methodologies

## Few of Agile Methodologies

- Scrum
- Extreme Programming [XP]
- Test driven Development [TDD]
- Feature Driven Development [FDD]
- Behavior-driven development [BDD]



# Roles in Agile

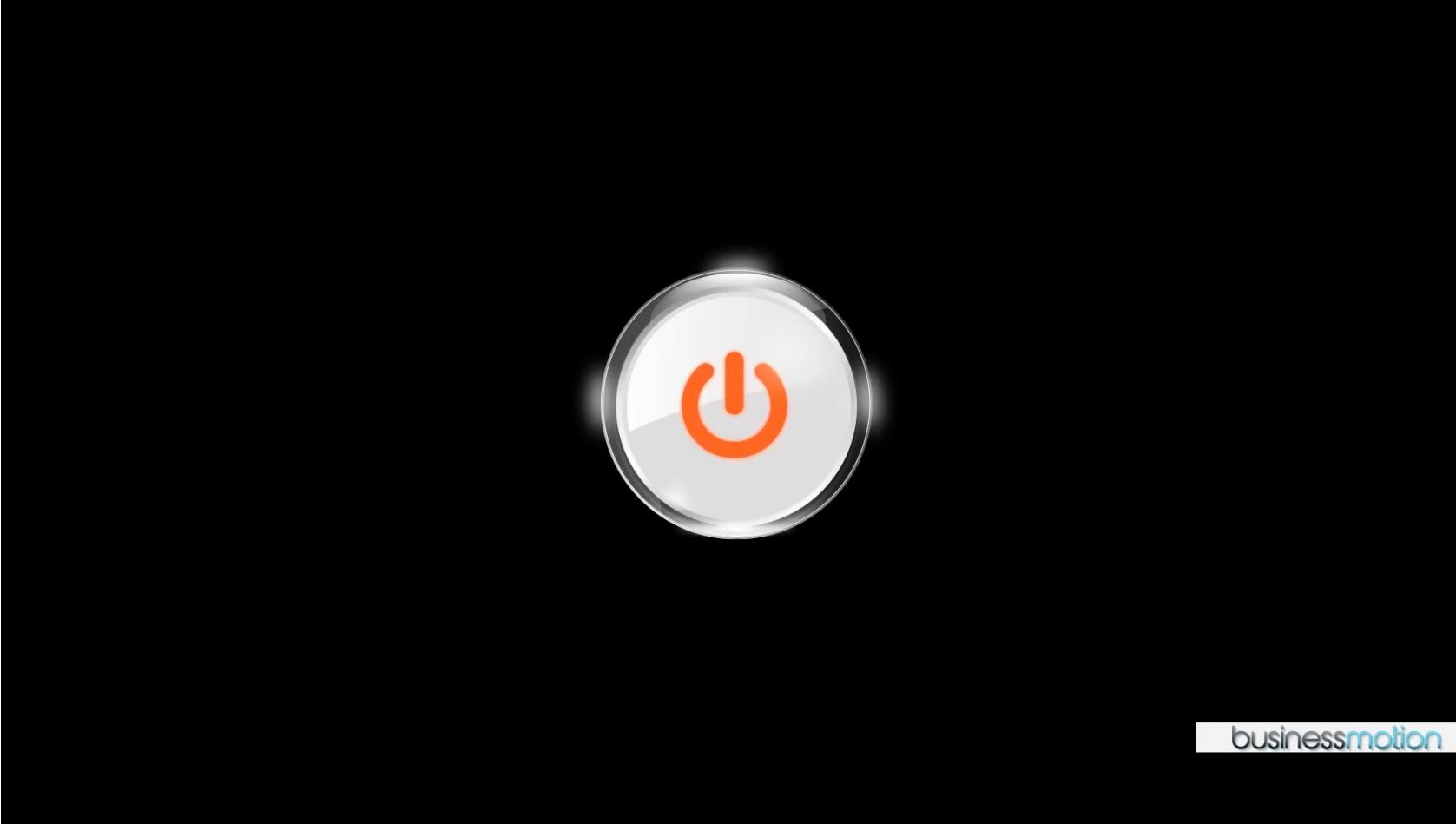
## Basic roles involved

- User
- Product Owner
- Software Development Team



# Agile Methodology

## Summary

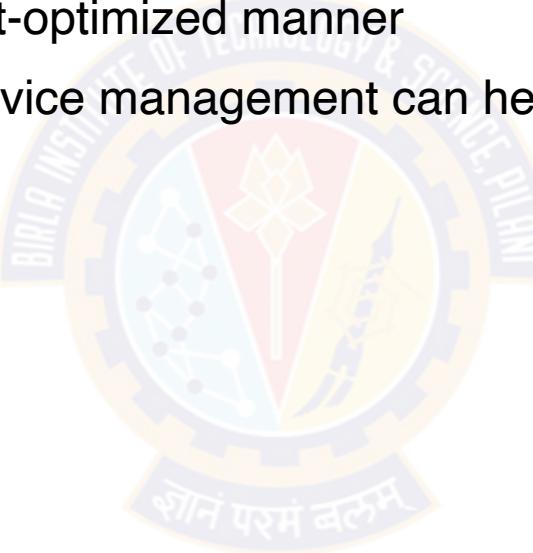


<https://www.youtube.com/watch?v=1iccpf2eN1Q>

# Operational Methodology

## ITIL

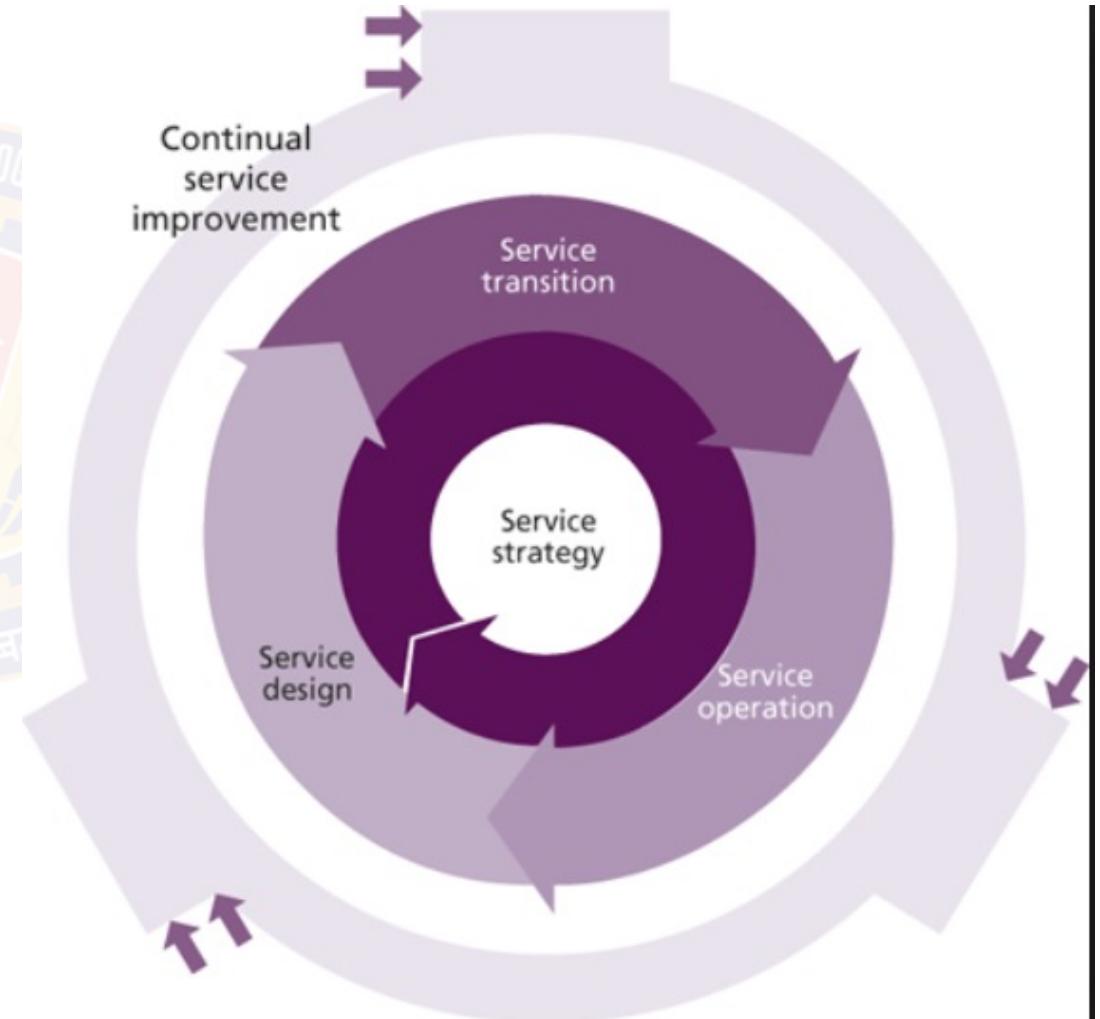
- ITIL is a framework of best practices for delivering IT services
- The ITIL processes within IT Service Management (ITSM) ensure that IT Services are provided in a focused, client-friendly and cost-optimized manner
- ITIL's systematic approach to IT service management can help businesses
  - Manage risk
  - Strengthen customer relations
  - Establish cost-effective practices
  - And build a stable IT environment
- that allows for
  - Growth
  - Scale and
  - Change



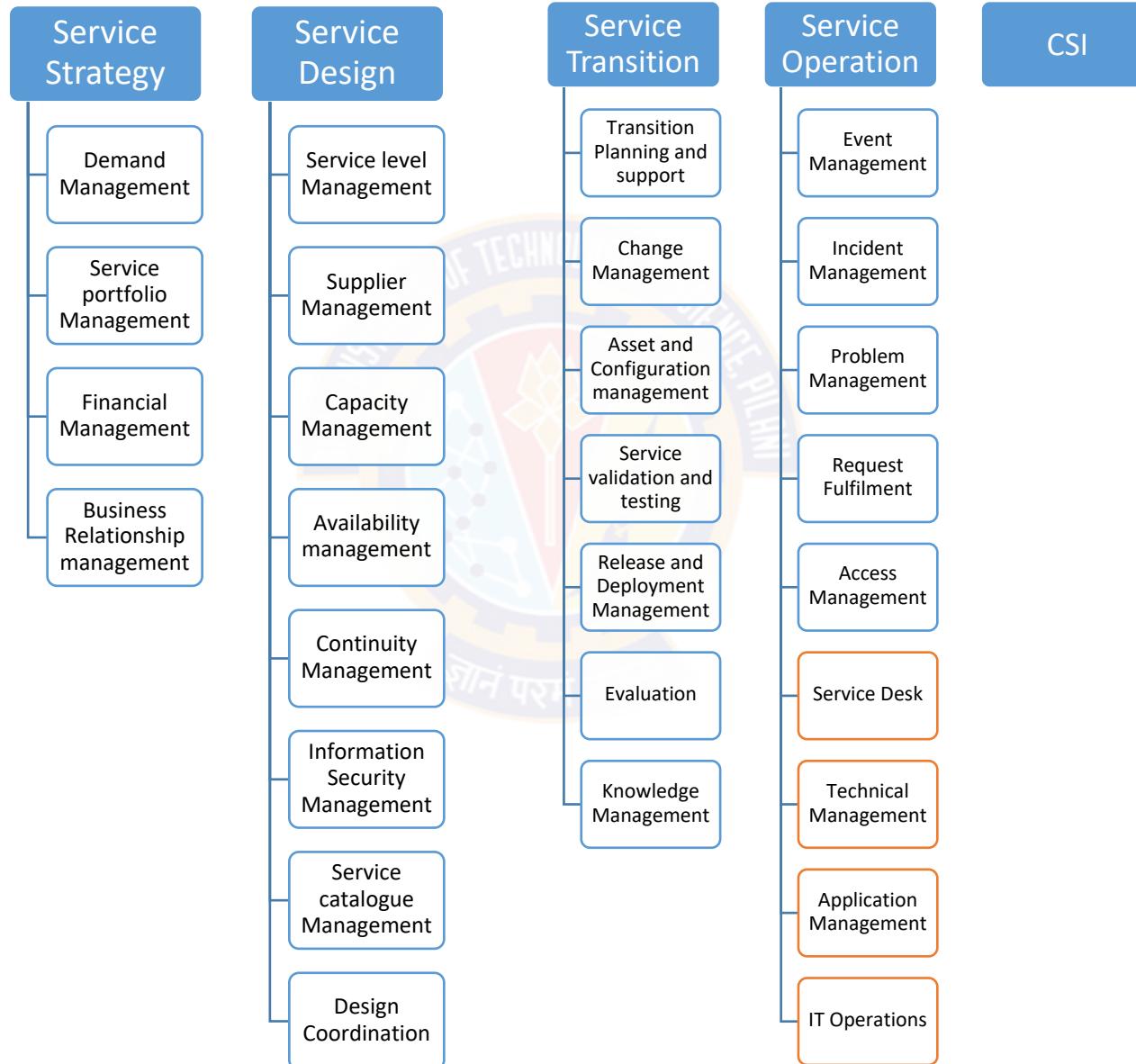
# IT Service Management (ITSM)

## Lifecycle

- ITIL views ITSM as a lifecycle
- Five Phases:
  - Service Strategy
  - Service Design
  - Service Transition
  - Service Operation
  - Service Improvement [create and maintain value]



## Framework

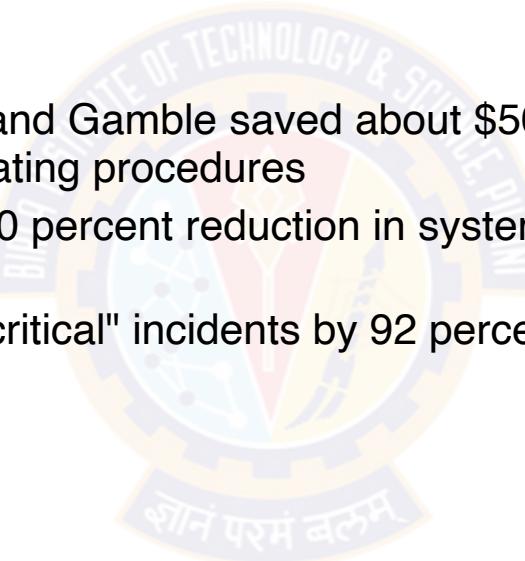


# ITIL

## Is a Project???



- ITIL is not a “Project”
- ITIL is ongoing journey
- Benefits of ITIL:
  - Pink Elephant reports that Procter and Gamble saved about \$500 million over four years by reducing help desk calls and improving operating procedures
  - Nationwide Insurance achieved a 40 percent reduction in system outages and estimates a \$4.3 million ROI over three years
  - Capital One reduced its "business critical" incidents by 92 percent over two years



## Lets Review



<https://www.youtube.com/watch?v=FSfgovmPH08>



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Introduction

- About DevOps
- Problems of Delivering Software
- Principles of Software Delivery
- Need for DevOps
- DevOps Practices in Organization
- The Continuous DevOps Life Cycle Process
- Evolution of DevOps
- Case studies



# DevOps

## Definition

- “DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality”

Implications of this definition

- Practices and tools
- Do not restricted scope of DevOps to testing and development



# DevOps

## Perspective involved



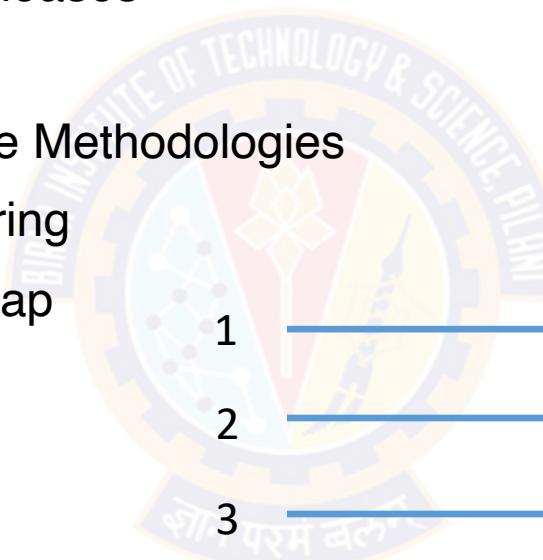
- Quality:
- Checklist
  - Approval
  - Release Note
  - Audit
  - Compliance



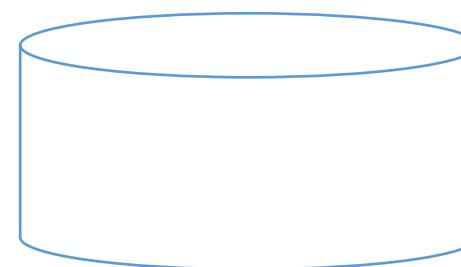
Trust

# Problems of Delivering Software

- Converting Idea to Product / Service?
- Reliable, rapid, low-risk software releases
- Ideal Environment
- Generic Methodologies for Software Methodologies
- More focus on Requirement Gathering
- Understanding the Value Stream Map



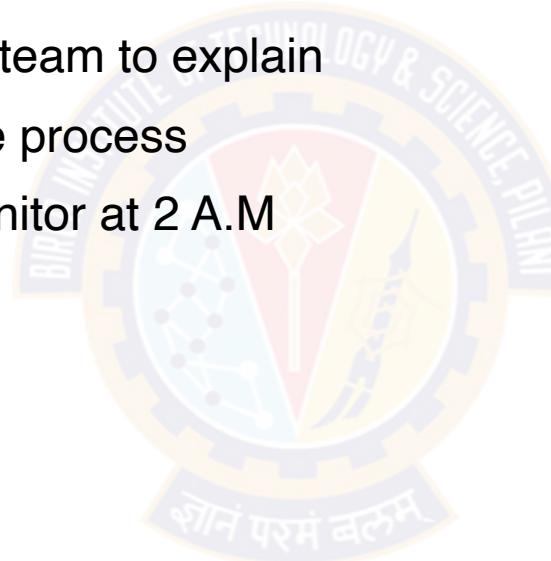
1  
2  
3  
4



# Common Release Antipatterns

## Deploying Software Manually

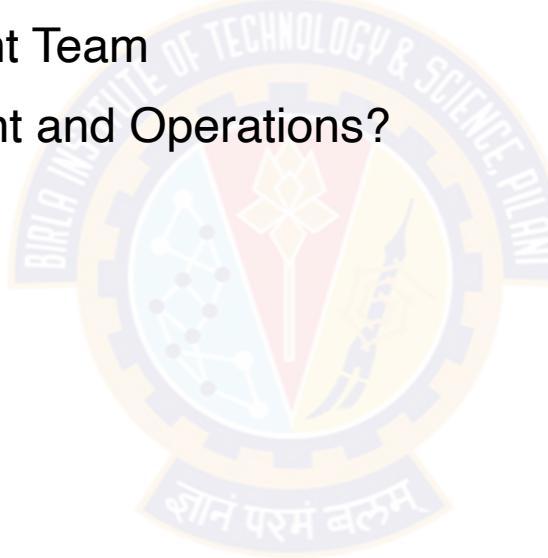
- Extensive and detailed documentation
- Reliance on manual testing
- Frequent calls to the development team to explain
- Frequent corrections to the release process
- Sitting bleary-eyed in front of a monitor at 2 A.M



# Common Release Antipatterns

## Deploying to a Production-like Environment Only after Development Is Complete

- Tester tested the system on development machines
- Releasing into staging is the first time that operations people interact with the new release
- Who Assembles? The Development Team
- Collaboration between development and Operations?



# Common Release Antipatterns

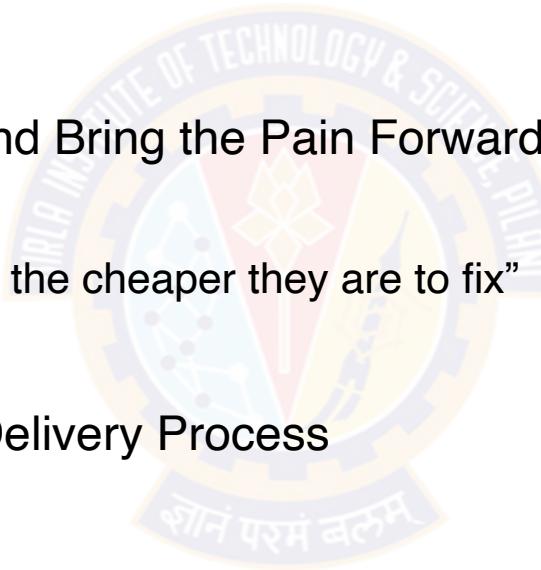
## Manual Configuration Management of Production Environments

- Difference in Deployment to Stage and Production
- Different host behave differently
- Long time to prepare an environment
- Cannot step back to an earlier configuration of your system
- Modification to Configuration Directly



# Principles of Software Delivery

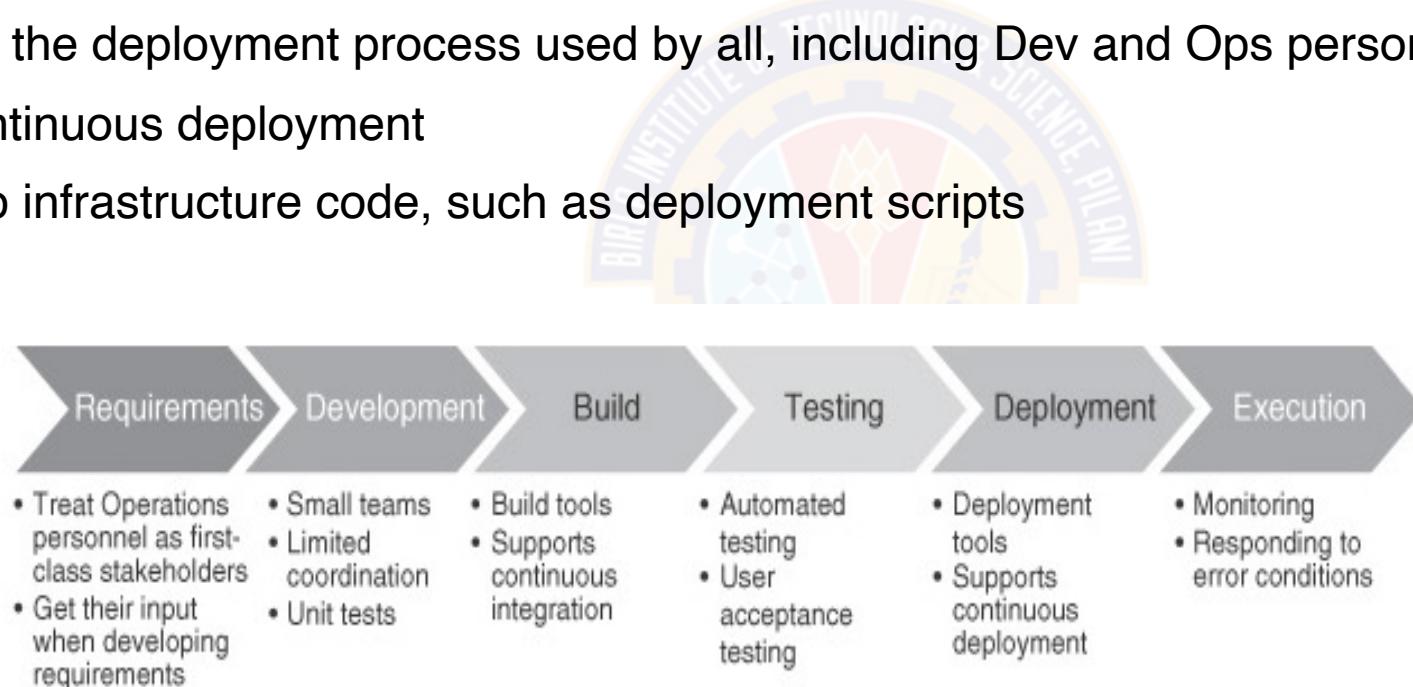
- Create a Repeatable, Reliable Process for Releasing Software
- Automate Almost Everything
- Keep Everything in Version Control
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Build Quality In
  - “The Earlier you catch the defects, the cheaper they are to fix”
- Done, Means Released
- Everybody Is Responsible for the Delivery Process
- Continuous Improvement



# DevOps Practices

## Five different categories of DevOps practices

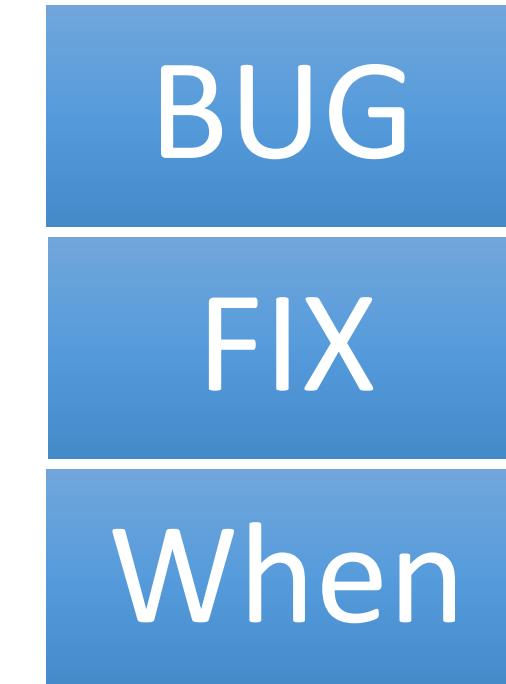
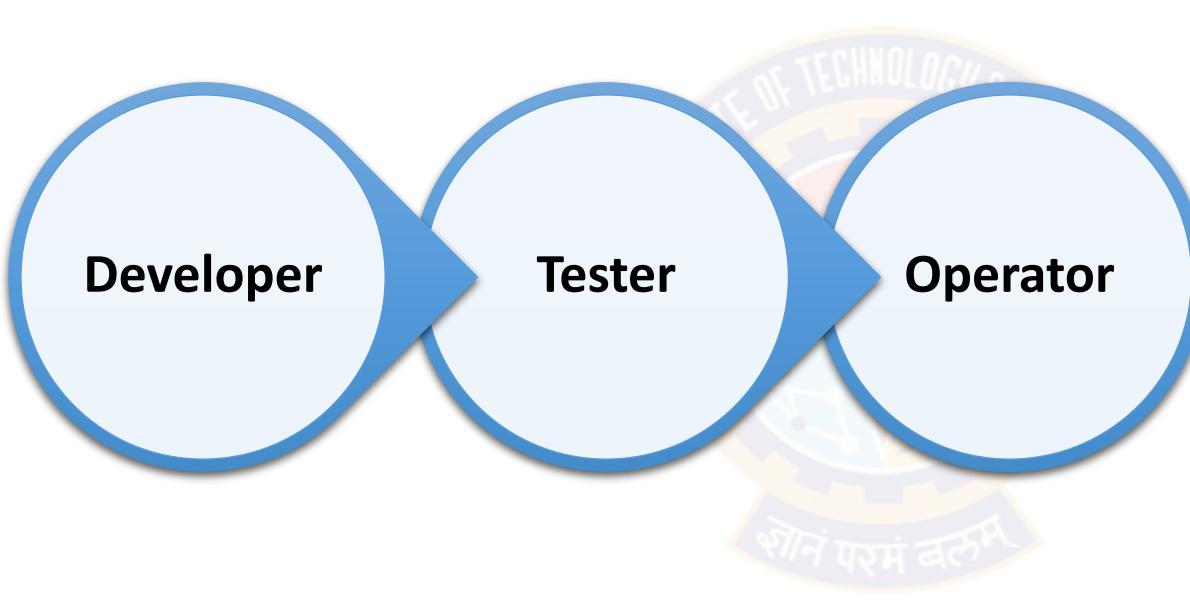
- Treat Ops as first-class citizens from the point of view of requirements
- Make Dev more responsible for relevant incident handling
- Enforce the deployment process used by all, including Dev and Ops personnel
- Use continuous deployment
- Develop infrastructure code, such as deployment scripts



**FIGURE 1.1** DevOps life cycle processes [Notation: Porter's Value Chain]

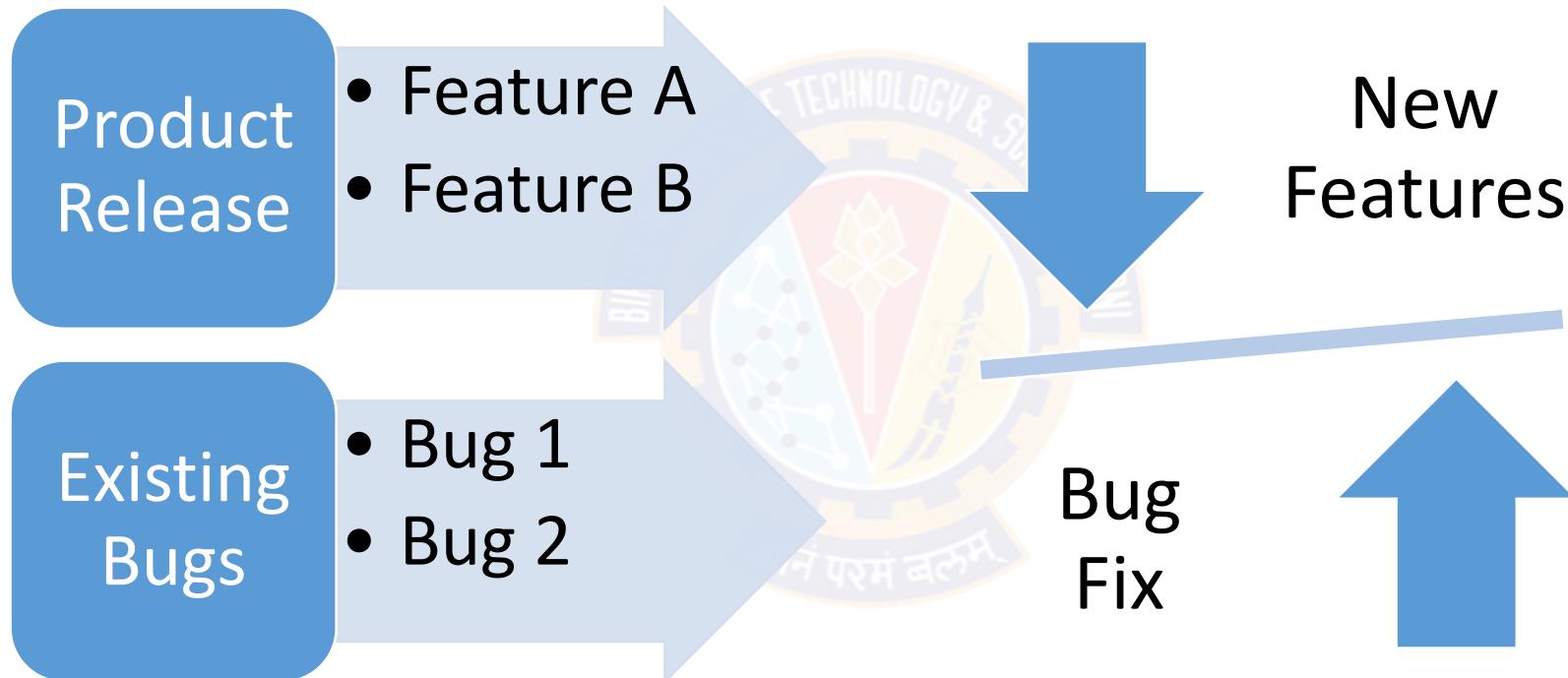
# Need for DevOps

## Timelines



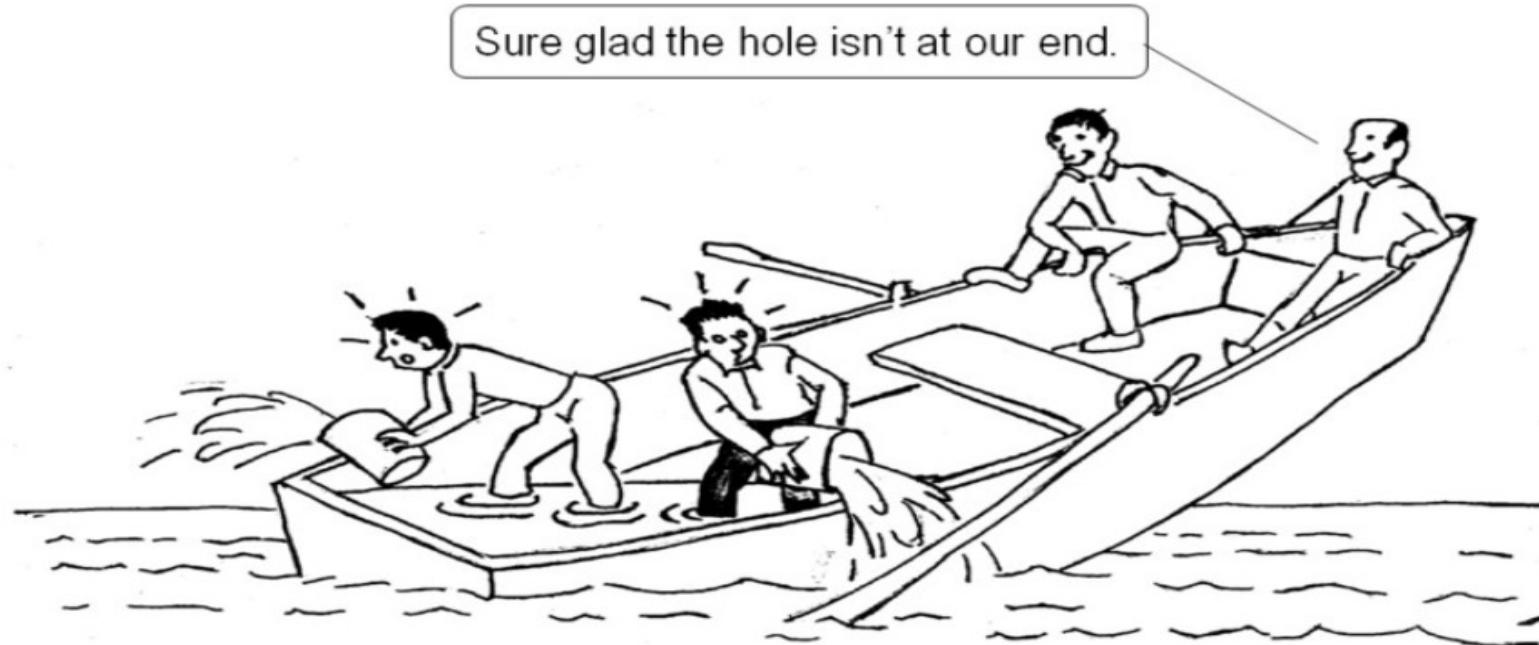
# Need for DevOps

## Imbalance



# Need for DevOps

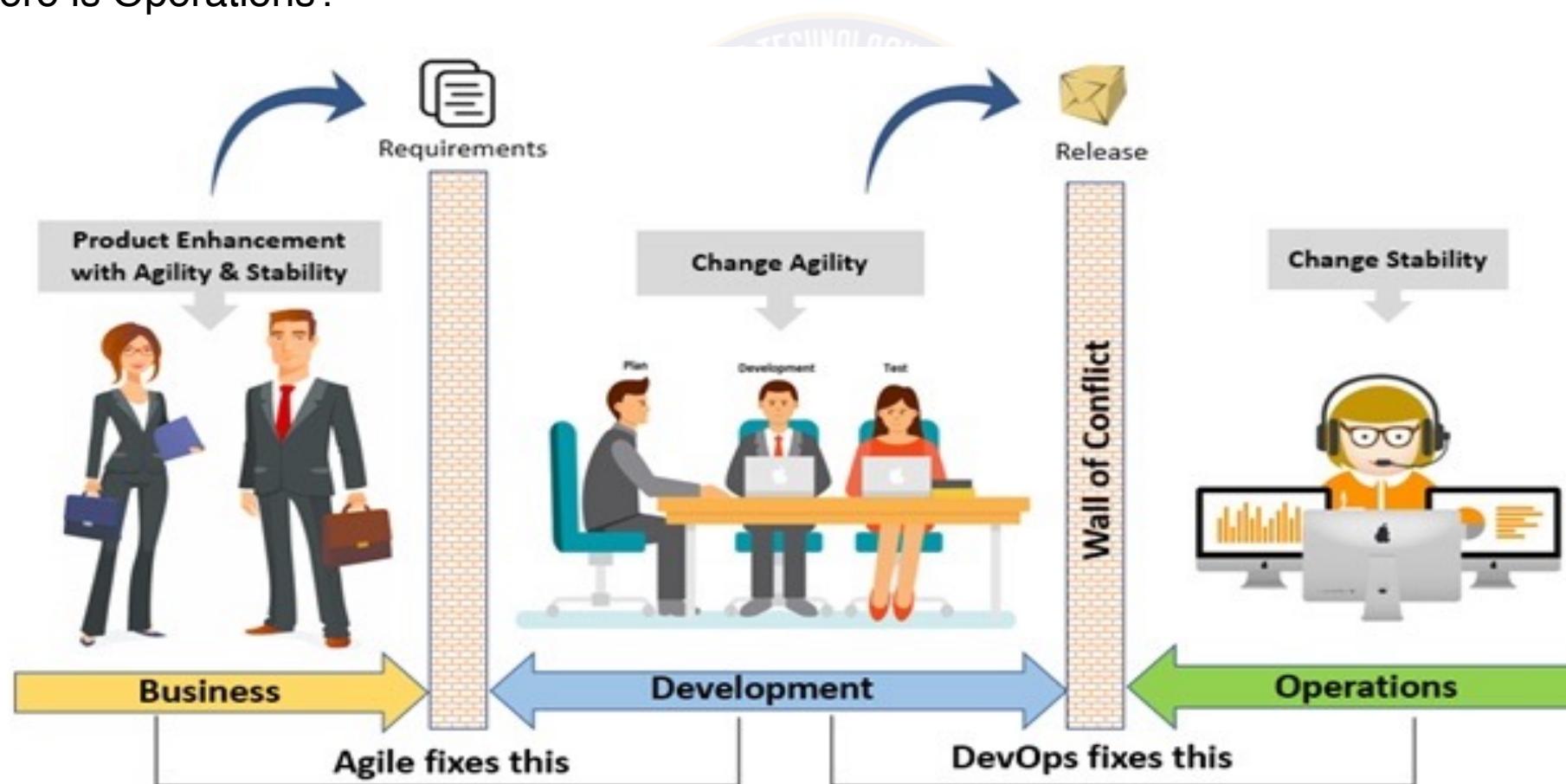
## Blame Game



# Need for DevOps

## Where is Operations?

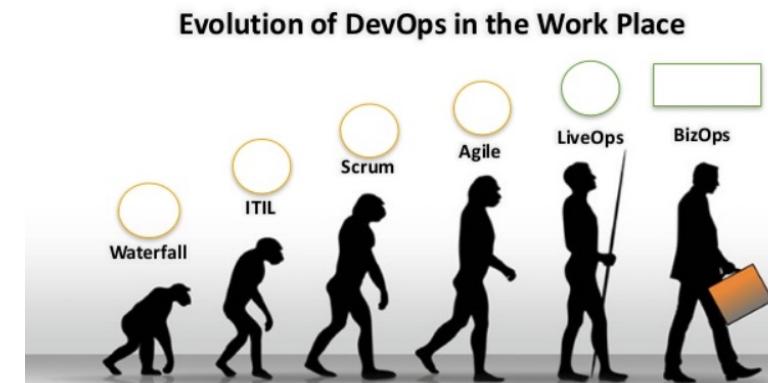
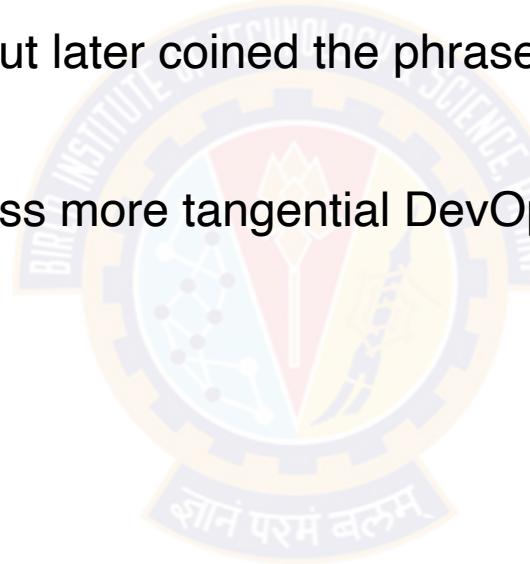
- Development is All Well (Waterfall, Agile)
- Where is Operations?



# The evolution of DevOps

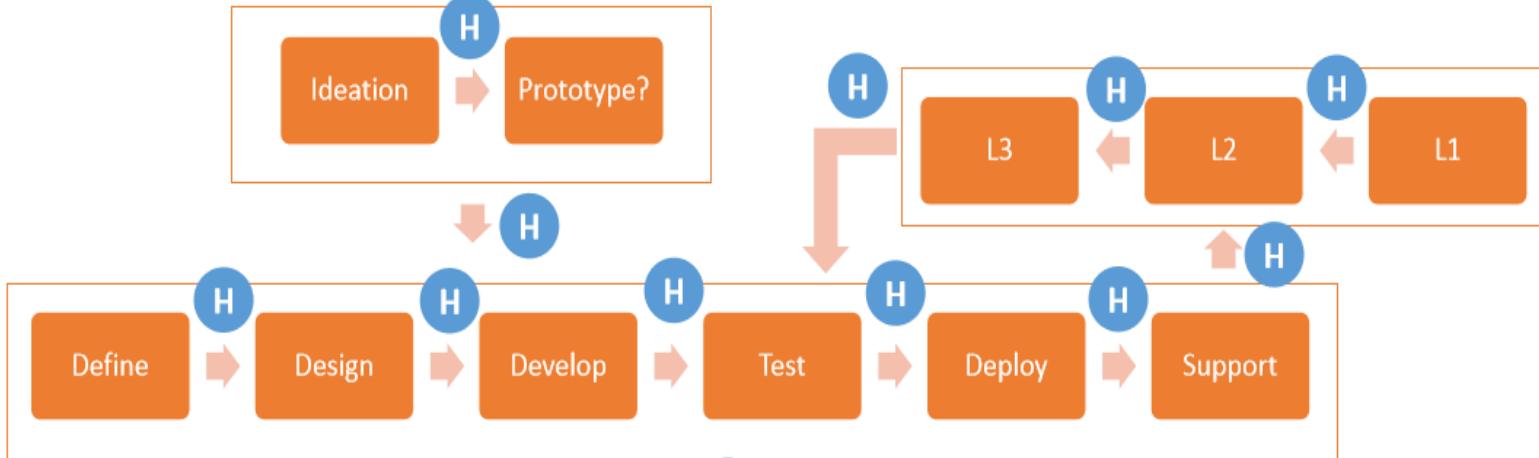
## History

- Back in 2007
- Patrick Debois [Belgian Engineer]
- Initially it was Agile Infrastructure but later coined the phrase DevOps
- Velocity conference in 2008
- And if you see you may come across more tangential DevOps Initiative
  - WinOps
  - DevSecOps
  - BizDevOps



# The old world before DevOps

## More Handshakes

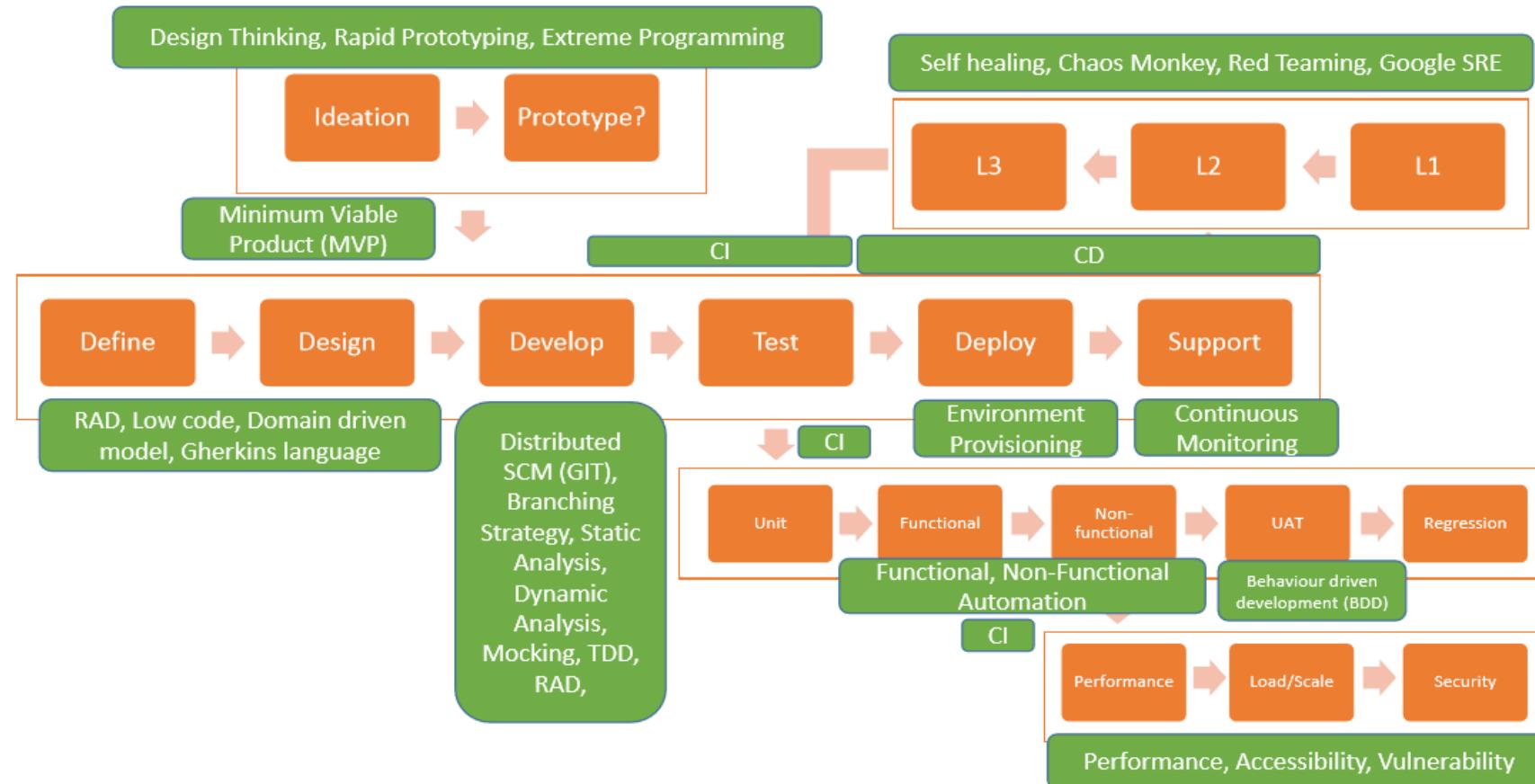


### Problems

- People dependent
- Information leakage
- Competency leakage
- Accountability issues
- Too many touchpoints for business
- Cultural Issues
- Higher cost
- Longer Go-To-Market (Losing competitive advantage, Opportunity loss)

# Evolution of DevOps :: New world

## No More Handshakes



# Case Study

## Flickr / Yahoo

- Flickr was able to reach
  - 10+ Deploy per day after adopting DevOps
- In 2009
- You May Also Refer:
- YouTube Link: <https://www.youtube.com/watch?v=LdOe18KhtT4>



**flickr**

# Case Study

## Netflix

- Netflix's streaming service is a large distributed system hosted on Amazon Web Services (AWS)
- So many components that have to work together to provide reliable video streams to customers across a wide range of devices
- Netflix engineers needed to focus heavily on the quality attributes of reliability and robustness for both server- and client-side components
- Achieved this with DevOps by introducing a tool called Chaos Monkey
- Chaos Monkey is basically a script that runs continually in all Netflix environments, causing chaos by randomly shutting down server instances
- Thus, while writing code, Netflix developers are constantly operating in an environment of unreliable services and unexpected outages
- Unique opportunity to test their software in unexpected failure conditions



# References

## CS 1 & 2

- 4th chapter from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- 1st chapter Continuous Delivery by Jez Humble and David Farley and 5th Chapter from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- For ITIL: <https://www.simplilearn.com/itil-key-concepts-and-summary-article>,



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Understanding DevOps

- DevOps Misconception
- DevOps Anti-Patterns
- Three Dimensions of DevOps
  - Process
  - People
  - Tools



# DevOps Misconception

## DevOps Myths & Misconception

- DevOps is a Team
- CI/CD is all about DevOps
- Non-Compliant to Industry Standards



An additional team is likely to cause more communication issues



Scalability  
Consistency  
Reliability

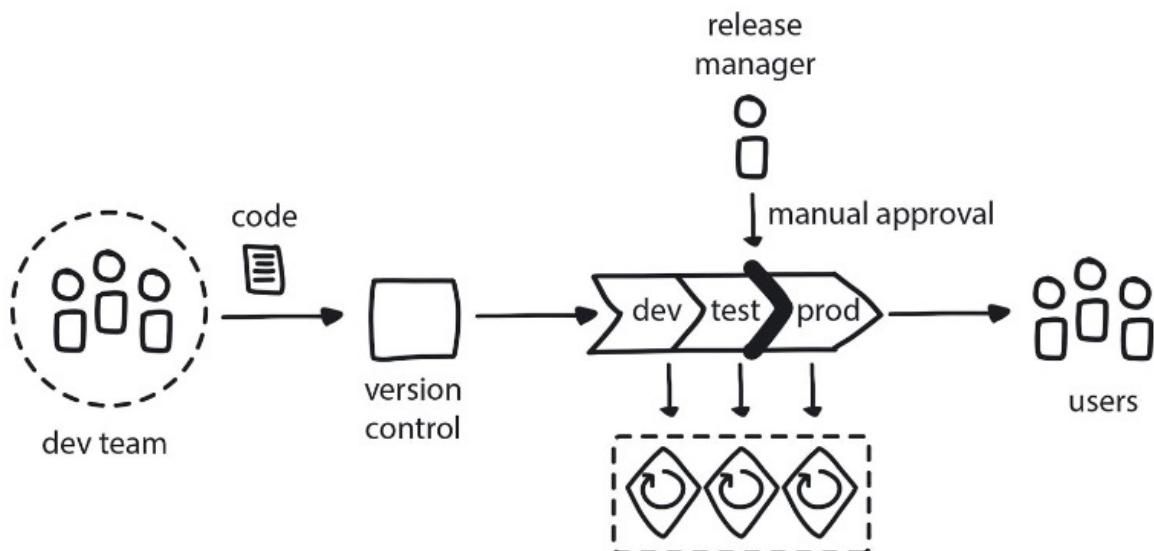


Regulations implement control in order to be compliant  
Controls reduce the risk that may affect the confidentiality, integrity, and availability of information

# DevOps Misconception

## DevOps Myths & Misconception

- Automation eliminates ALL bottlenecks
- “Universal” Continuous Delivery pipeline
- All about Tools
- Release as fast as Amazon / Facebook



It increases feedback loops  
However Hand off processes can often add to process wait states



Processes to fit organizational and busin. needs — not vice-versa



Mandating tools developers can use.  
Prioritizing tools over people

Empowered smaller self-sufficient teams and Continuous Delivery Engineer

Improve the partnership b/w dev test & Ops

# DevOps Anti-Patterns

## Anti-Patterns

- Blame Culture
  - A blame culture is one that tends toward blaming and punishing people when mistakes are made, either at an individual or an organizational level
- Silos
  - A departmental or organizational silo describes the mentality of teams that do not share their knowledge with other teams in the same company
- Root Cause Analysis
  - Root cause analysis (RCA) is a method to identify contributing and “root” causes of events or near-misses/close calls and the appropriate actions to prevent recurrence
- Human Errors
  - Human error, the idea that a human being made a mistake that directly caused a failure, is often cited as the root cause in a root cause analysis

# Lessons of History

## Example for Silos

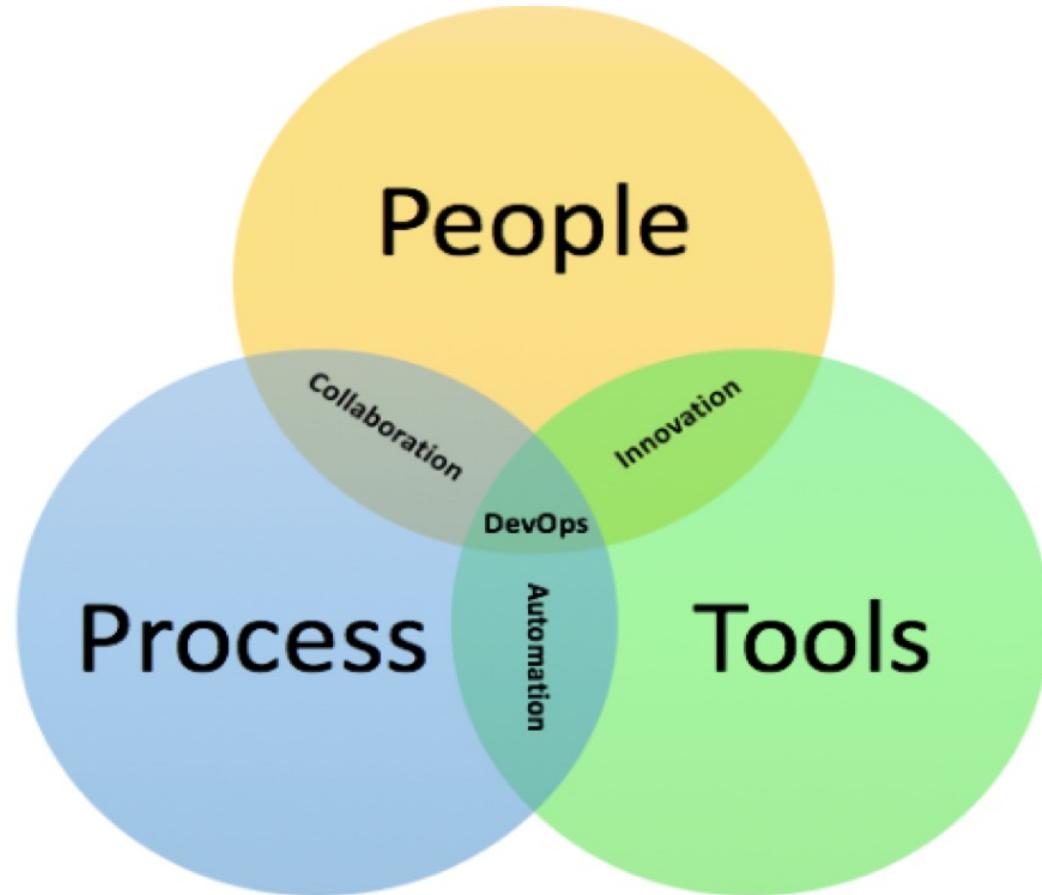
- TVS India
  - T. V. Sundaram Iyengar
  - Founded in 1978



# DevOps Dimensions

## Three dimensions of DevOps

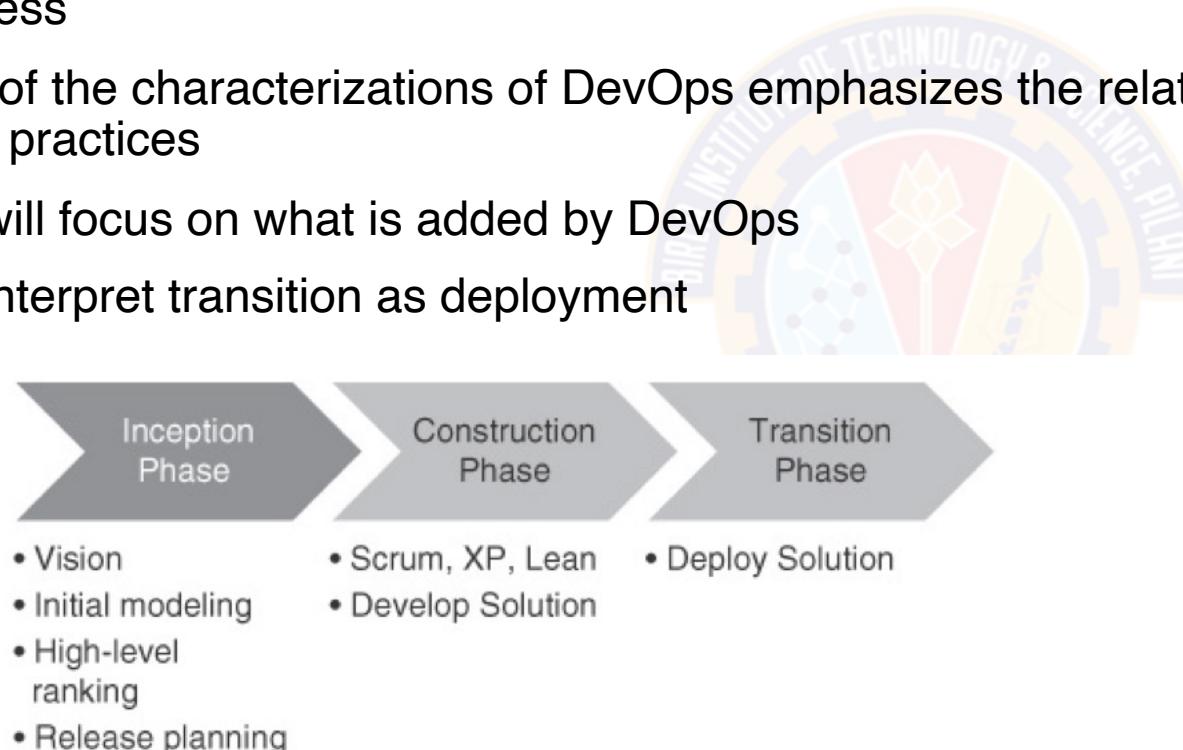
- People
- Process
- Tools / Technology



# DevOps - Process

## DevOps and Agile

- We need processes and policies for doing things in a proper way and standardized across the projects so the sequence of operations, constraints, rules and so on are well defined to measure success
- One of the characterizations of DevOps emphasizes the relationship of DevOps practices to agile practices
- We will focus on what is added by DevOps
- We interpret transition as deployment



**FIGURE 1.2** Disciplined Agile Delivery phases for each release. (Adapted from *Disciplined Agile Delivery: A Practitioner's Guide* by Ambler and Lines) [Notation: Porter's Value Chain]

# DevOps and Agile

## DevOps practices impact all three phases

- Inception phase : During the inception phase, release planning and initial requirements specification are done
  - Considerations of Ops will add some requirements for the developers
  - Release planning includes feature prioritization but it also includes coordination with operations personnel
- Construction phase: During the construction phase, key elements of the DevOps practices are the management of the code branches,
  - the use of continuous integration
  - continuous deployment
  - incorporation of test cases for automated testing
- Transition phase: In the transition phase, the solution is deployed and the development team is responsible for the deployment, monitoring the process of the deployment, deciding whether to roll back and when, and monitoring the execution after deployment



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## DevOps : Process Dimension

- DevOps and Agile
- Team Structure
- Agile Methodology
  - TDD
  - BDD
  - FDD



# DevOps – Process

## Process



vs



Extreme  
Programming

SCRUM

Feature Driven  
Development

Kanban

Hybrid Agile

Distributed  
Agile

# DevOps – Process

## Team Structure

- Choosing appropriate team structure (Resource planning)

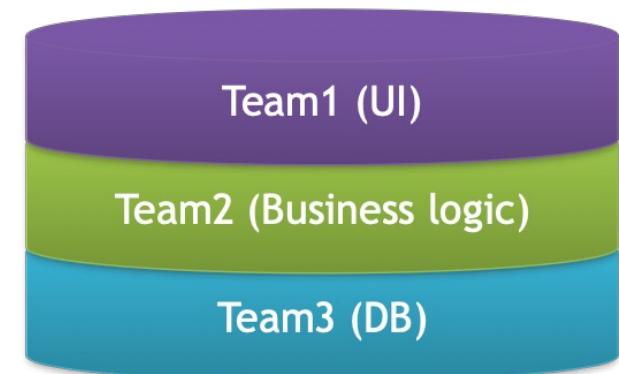
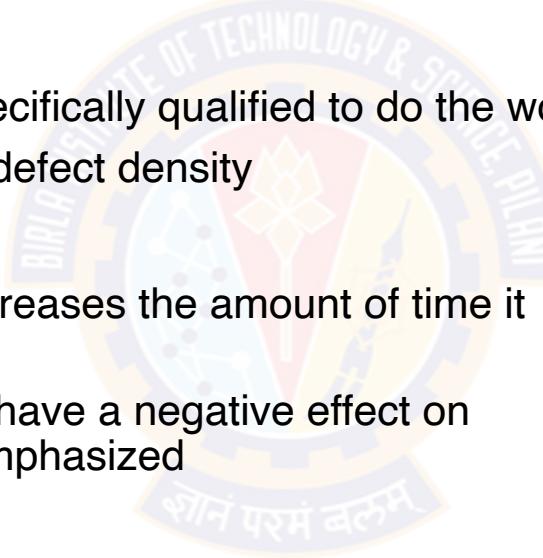
# Component Team Structure

# Featured Team Structure

# DevOps – Process

## Component teams

- Made up of experts that specialize in a specific domain
- Architecture diagram can be represented as layer of the system
- Advantages:
  - work is always done by people specifically qualified to do the work
  - work is done efficiently with a low defect density
- Disadvantages
  - Working as Component Teams increases the amount of time it takes to deliver
  - working in Component Teams will have a negative effect on productivity that cannot be over emphasized

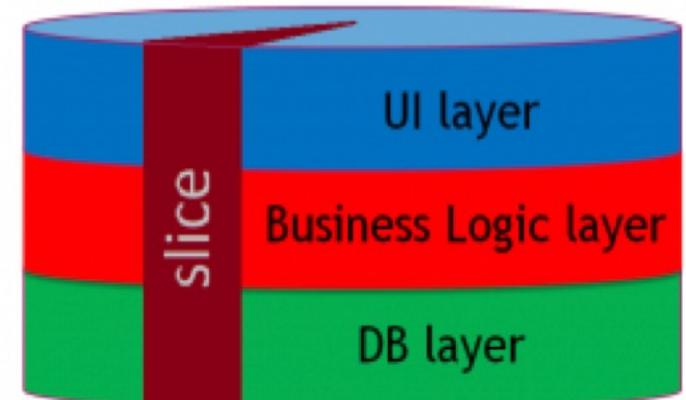
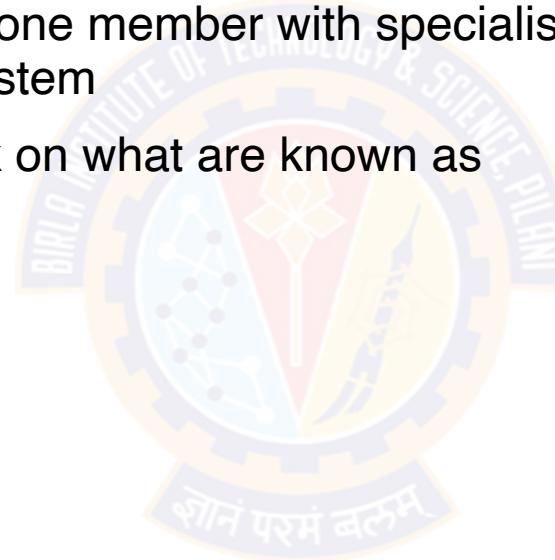


**Architecture diagram**

# DevOps – Process

## Feature teams

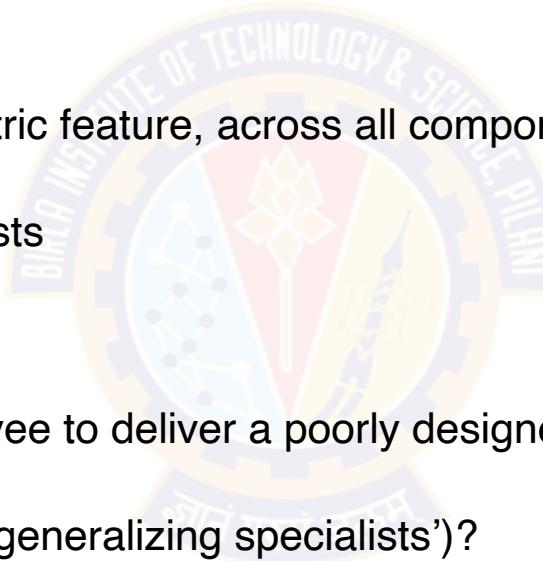
- Feature Teams contain multi-disciplined individuals that have the ability and freedom to work in any area of the system
- Feature Team usually has at least one member with specialist knowledge for each layer of the system
- This allows Feature Teams to work on what are known as 'vertical slices' of the architecture



# DevOps – Process

## Feature teams

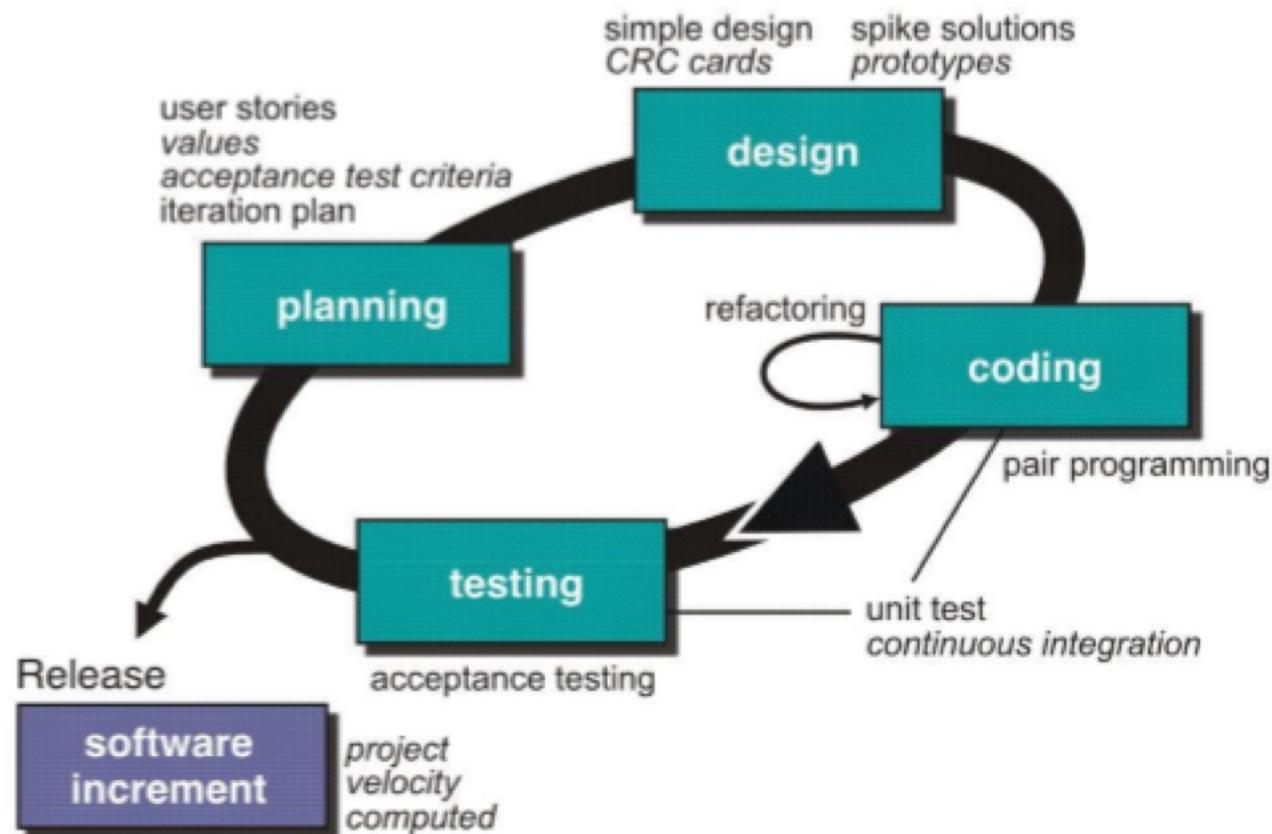
- The characteristics of a feature team are listed below:
  - long-lived—the team stays together so that they can ‘jell’ for higher performance; they take on new features over time
  - co-located
  - work on a complete customer-centric feature, across all components and disciplines (analysis, programming, testing, ...)
  - composed of generalizing specialists
  - in Scrum, typically  $7 \pm 2$  people
- Disadvantages
  - untrained or inexperienced employee to deliver a poorly designed piece of work into a live environment
  - Where do you get the personnel ('generalizing specialists')?



# DevOps: Process

## TDD

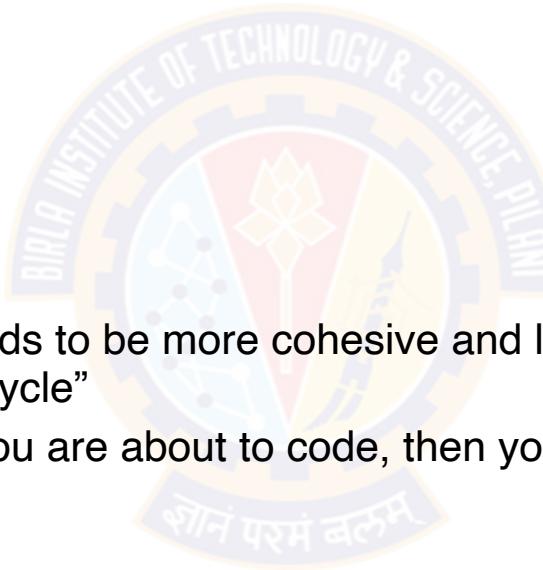
- Prerequisite :: XP



# TDD

## Understanding

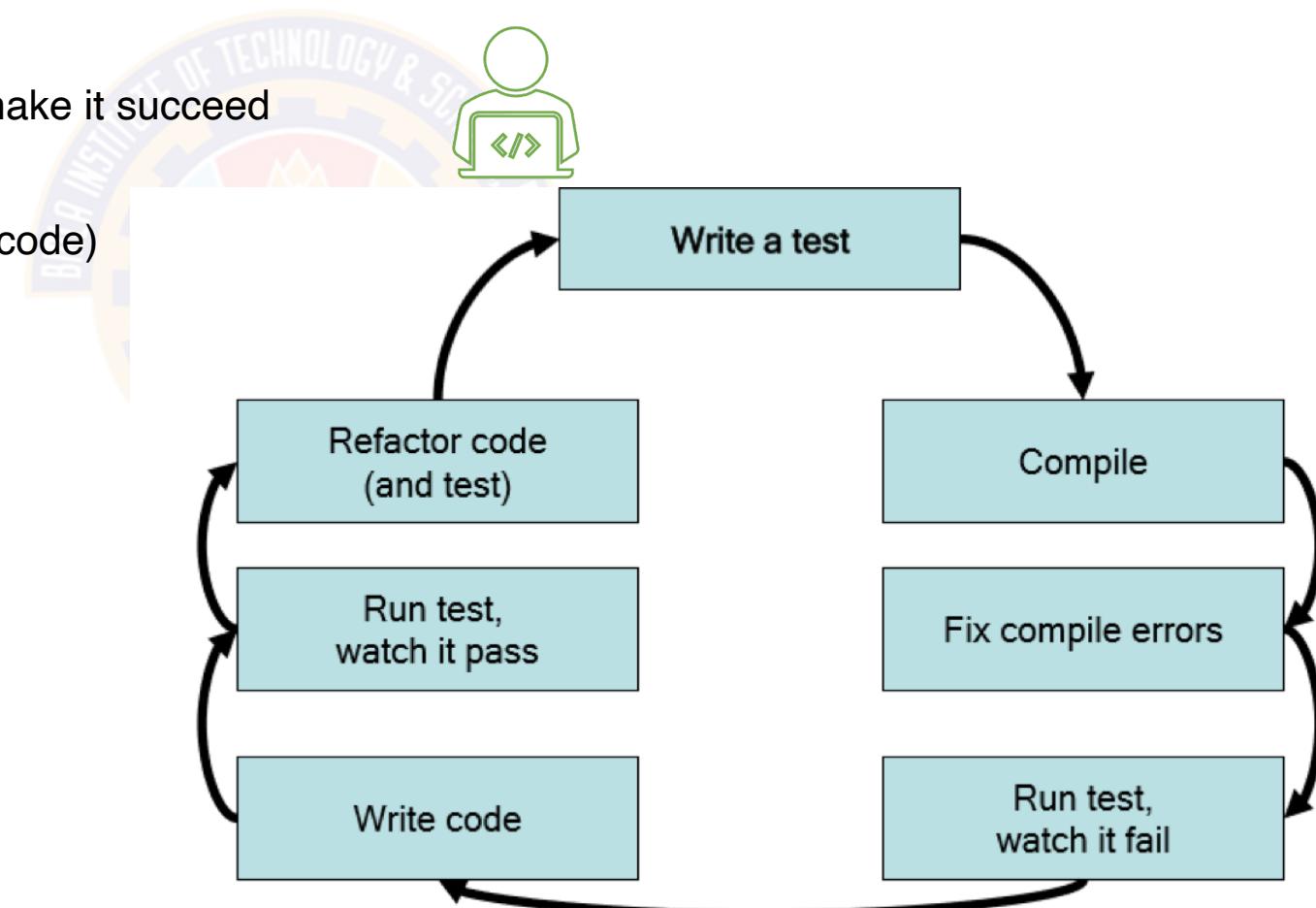
- Many Names
  - Test driven development
  - Test drive design
  - Emergent design
  - Test first development
- TDD Quotes
  - Kent Beck said “Test-first code tends to be more cohesive and less coupled than code in which testing isn’t a part of the intimate coding cycle”
  - “If you can’t write a test for what you are about to code, then you shouldn’t even be thinking about coding”



# TDD

## TDD Three steps

- RED
  - Write a new TEST which fails
- GREEN
  - Write simplest possible code to make it succeed
- REFACTOR
  - Refactor the code (including test code)



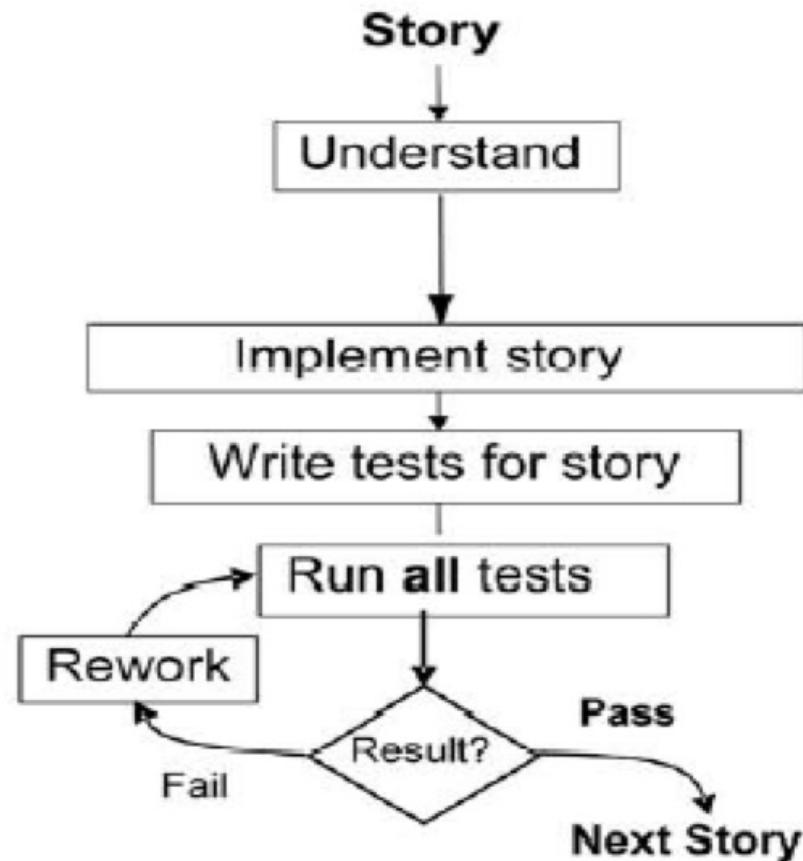
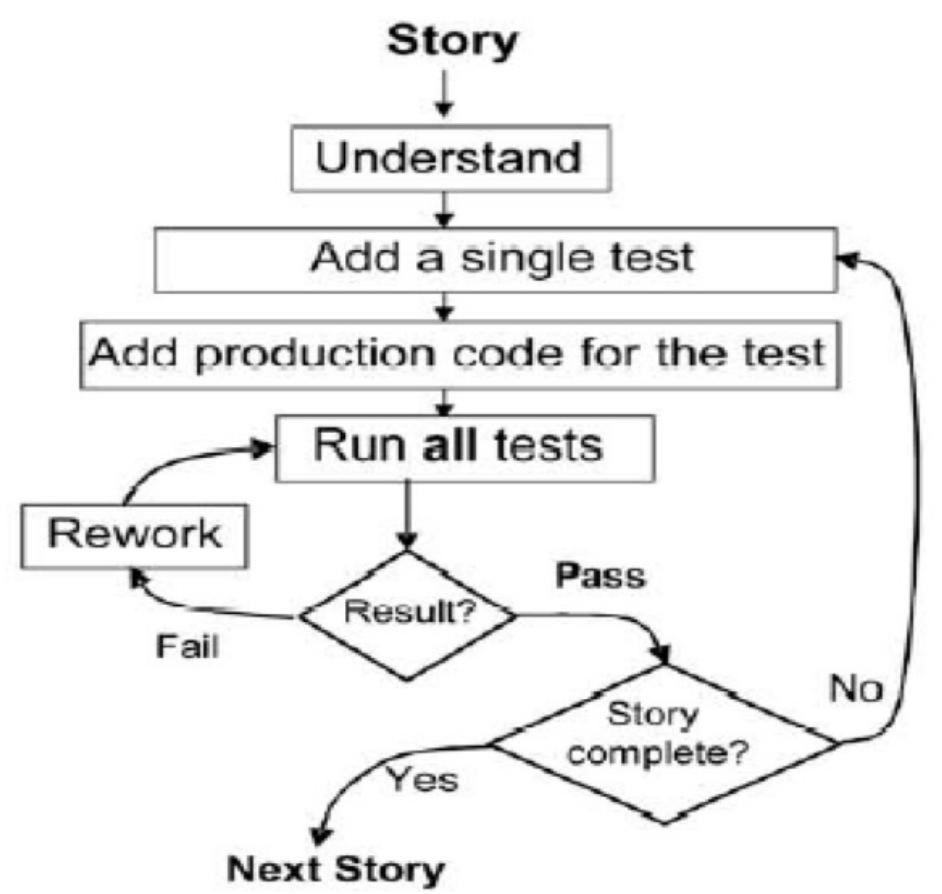
## Why TDD

- TDD can lead to more modularized, flexible, and extensible code
- Clean code
- Better code documentation
- More productive
- Good design



# TDD

## Test First vs. Test Last



# TDD Cycle

## Red, Green and Refactor

- Example (payment gateway task)

1. Choose a small task
2. Write a failing test (RED Test)
3. Write simplest code to make the test pass (GREEN Test)
4. REFACTOR
5. Repeat



Run all test



## Implementation Code

```
payment_gateway = function () {  
  credit_card ();  
  netbanking ();  
  upi_payment (); ————— all_upi_payment ();  
}
```

## Tests

.....  
.....  
.....



```
new_test = function () {  
  payment_gateway_includes_upi_option ();  
}
```

# TDD

## Lets Have TDD Overview



<https://www.youtube.com/watch?v=uGaNkTahrlw&t=132s>

# DevOps: Process

## FDD

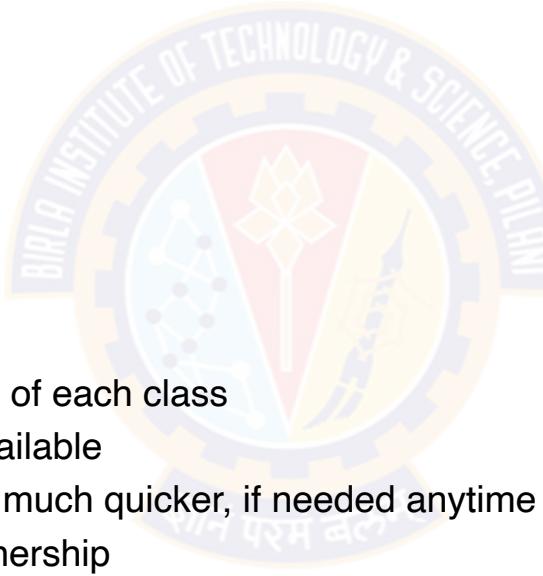
- What is a Feature?
  - Definition: small function expressed in client-valued terms
  - FDD's form of a customer requirement



# FDD

## FDD Primary Roles

- Project Manager
- Chief Architect
- Development Manager
- Domain Experts
- Class Owners
  - This concept differs FDD over XP
  - Benefits
    - Someone responsible for integrity of each class
    - Each class will have an expert available
    - Class owners can make changes much quicker, if needed anytime
    - Easily lends to notion of code ownership
    - Assists in FDD scaling to larger teams, as we have one person available for complete ownership of feature.
- Chief Programmers



## FDD Primary Roles

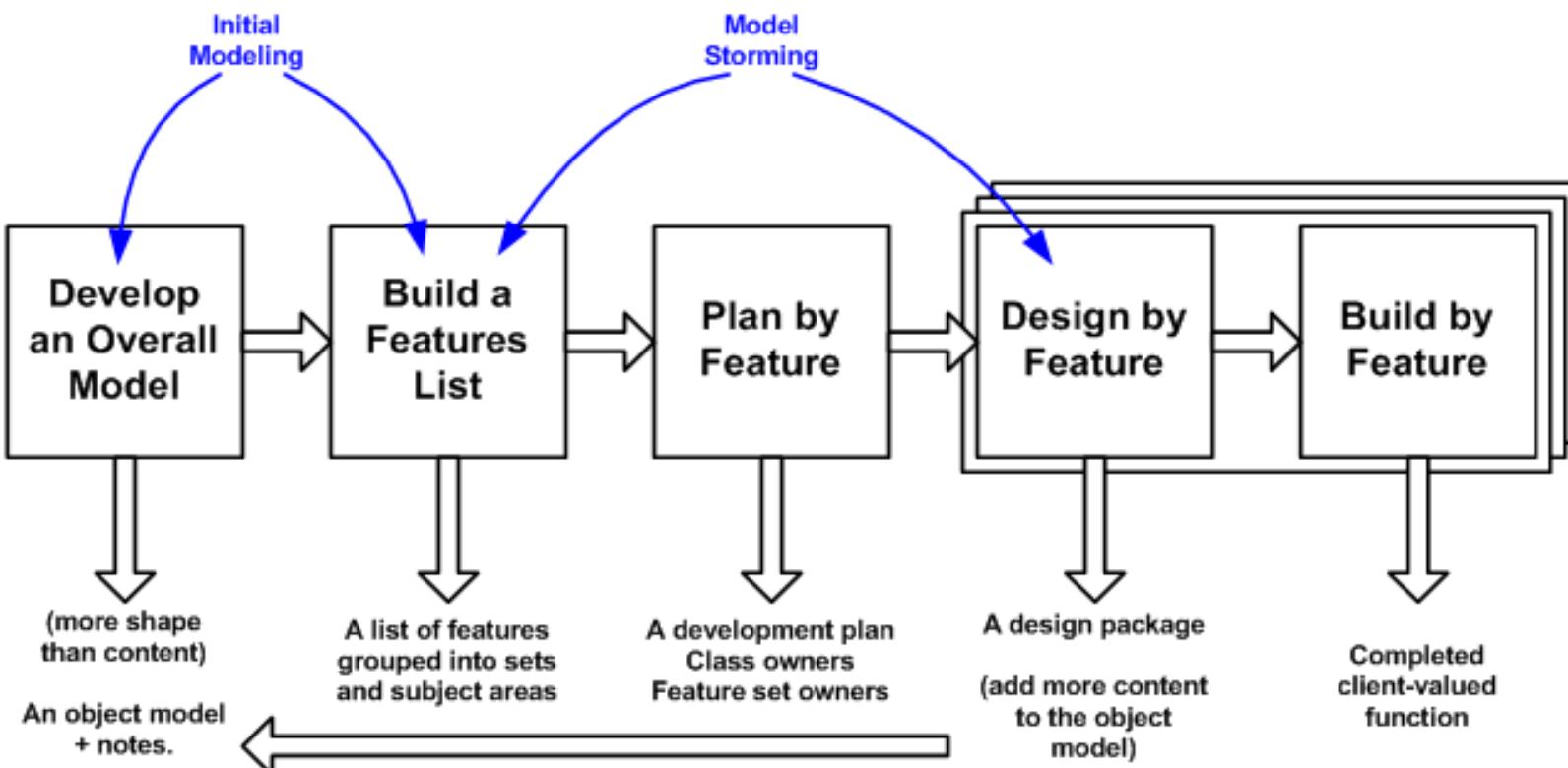
- Release Manager
- Language Guru
- Build Engineer
- Toolsmith
- System Administrator
- Tester
- Deployers
- Technical Writer



## Feature Driven Development Process

- Process #1: Develop an Overall Model
- Process #2: Build a Features List
- Process #3: Plan By Feature
  - Constructing the initial schedule, Forming level of individual features, Prioritizing by business value , As we work on above factors we do consider dependencies, difficulty, and risks.
  - These factors will help us on Assigning responsibilities to team members, Determining Class Owners , Assigning feature sets to chief programmers
- Process #4: Design By Feature
  - Goal: not to design the system in its entirety but instead is to do just enough initial design that you are able to build on
  - This is more about Form Feature Teams: Where team members collaborate on the full low level analysis and design.
- Process #5: Build By Feature
  - Goal: Deliver real, completed, client-valued function as often as possible

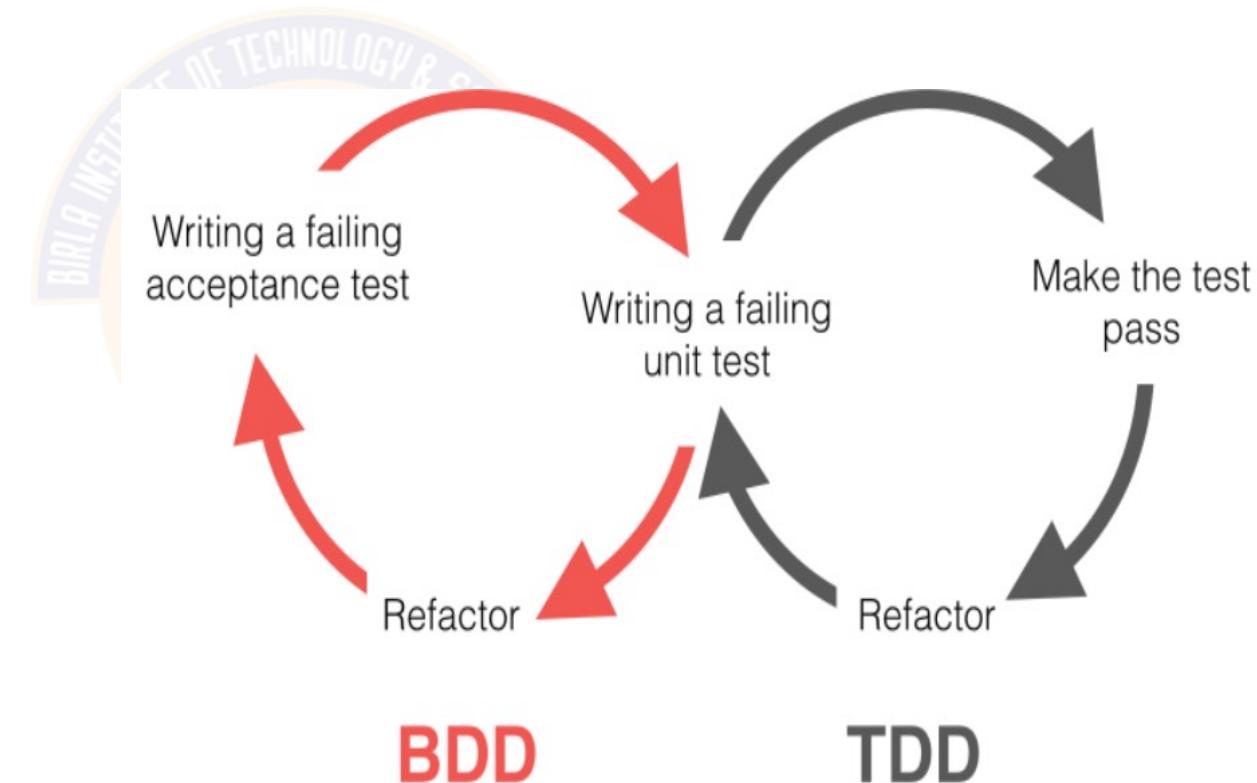
## Feature Driven Development Process



# DevOps: Process

## BDD

- What is BDD?:
  - General Technique of TDD
  - Follows same principle as TDD
  - Shared tools
  - Shared Process



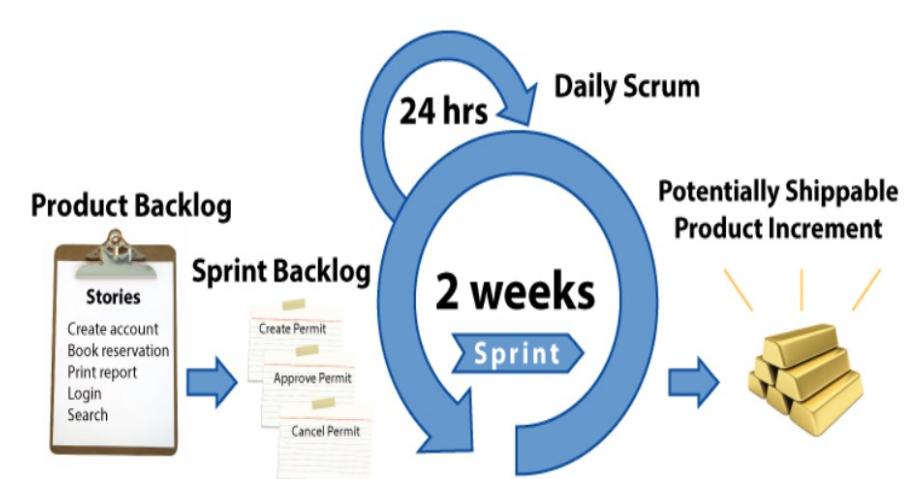
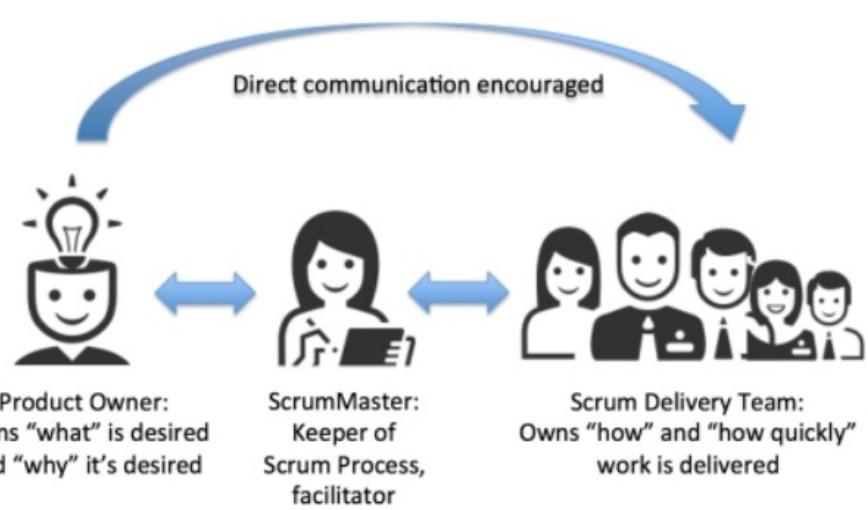
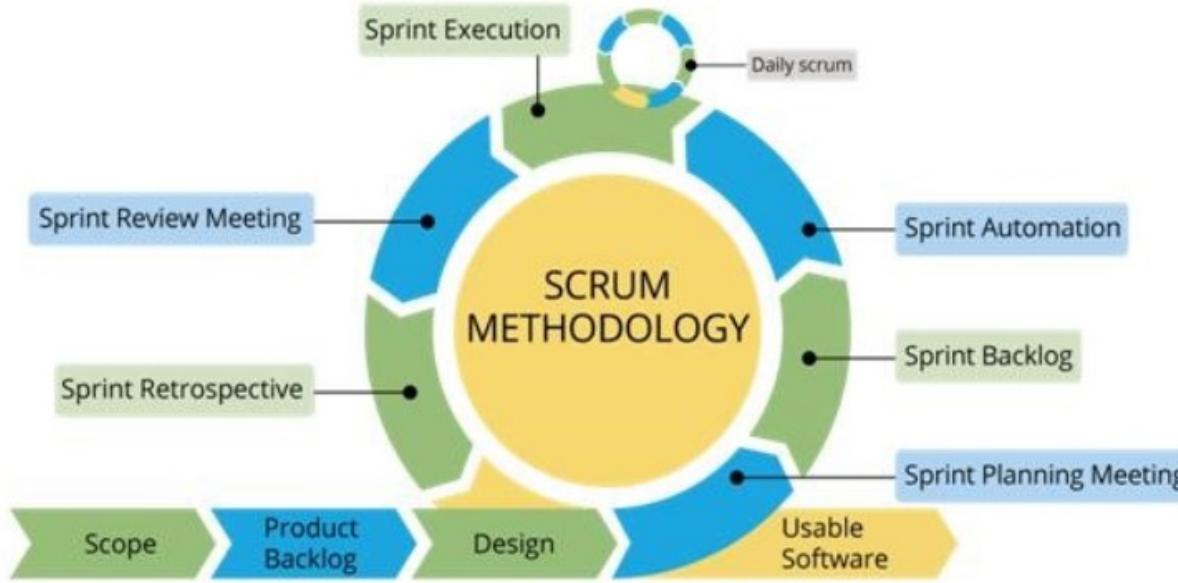
## BDD Basic structure : Example

- User story:
  - As someone interested in using the Mobile app, I want to sign up in from the app so that I can enjoy my membership.
- Mobile App Signup:
  - Scenario 1:
    - Given that I am on the app's "Create new account" screen
    - Then I should see a "Sign up using Facebook" button
    - And I should see a "Sign up using Twitter" button
    - And I should see a "Sign up with email" form field
    - Then I should see a new screen that asks permission to use my Facebook account data to create my new Mobile app account



# DevOps: Process

## SCRUM



# SCRUM

## SCRUM Overview

- You can refer below YouTube video for understanding the structure of organization to be agile.
- This Video is of First Bank in world to adopt Agile Scrum
- <https://www.youtube.com/watch?v=uXg6hG6FrG0>



# References

## CS 2 & 3

- Chapter 5 from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- 
- For BDD and FDD: <https://www.agilealliance.org>





# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## DevOps : People & Tools Dimension

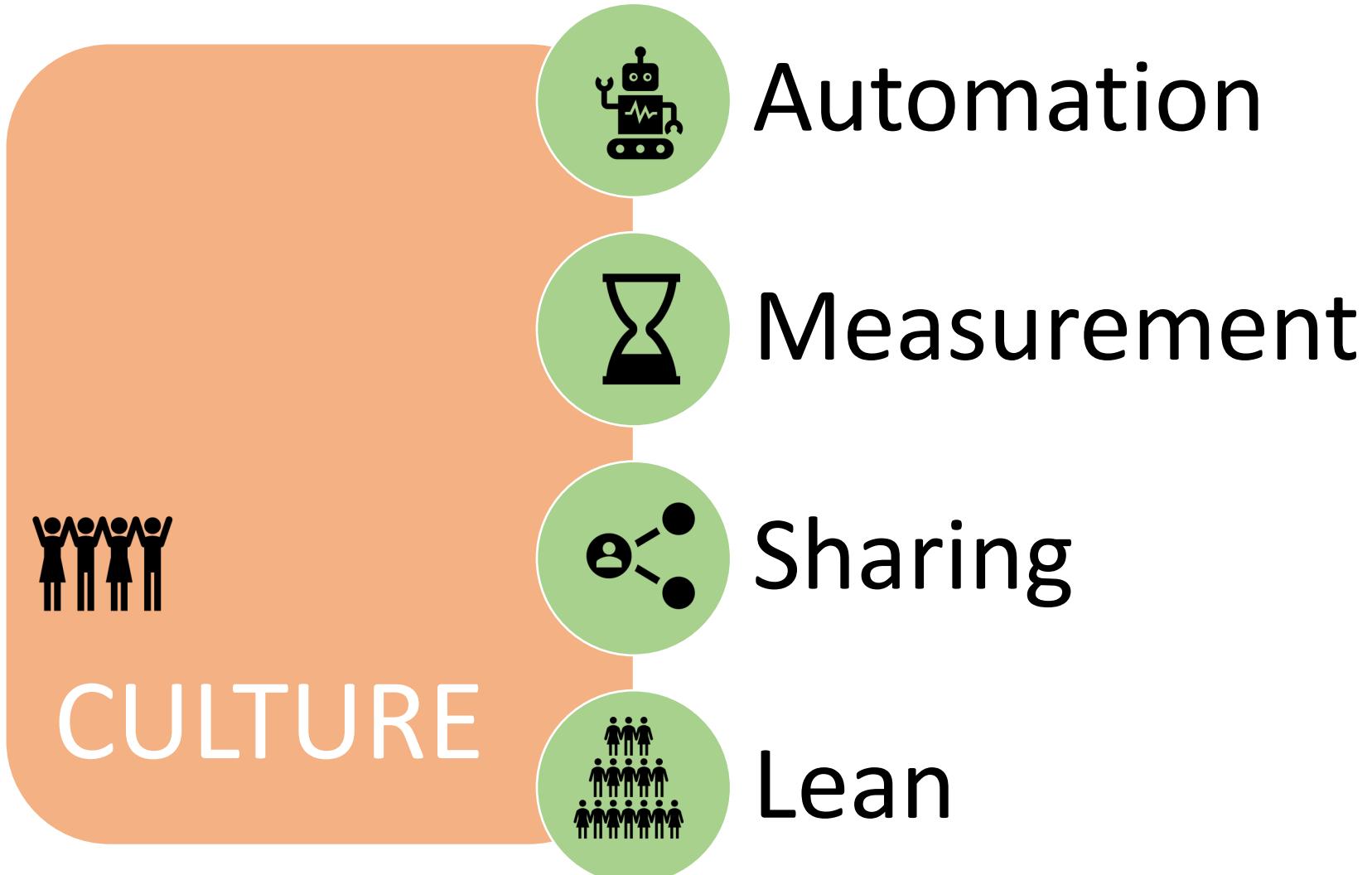
- Transformation to Enterprise DevOps culture
- DevOps - People
- DevOps – Tools



# Transformation to Enterprise DevOps culture

## DevOps Values

More than anything else, DevOps is a culture movement based on human and technical interaction to improve relationships and results



# Transformation to Enterprise DevOps culture

## Initial Planning for Enterprise Readiness

- Participants should absolutely include all the towers that make up the solution delivery
- Design Thinking : It is a great method; it leverages the expertise of all stakeholders, enables them to come to a common understanding
- High Level Output



Participant  
& Inputs

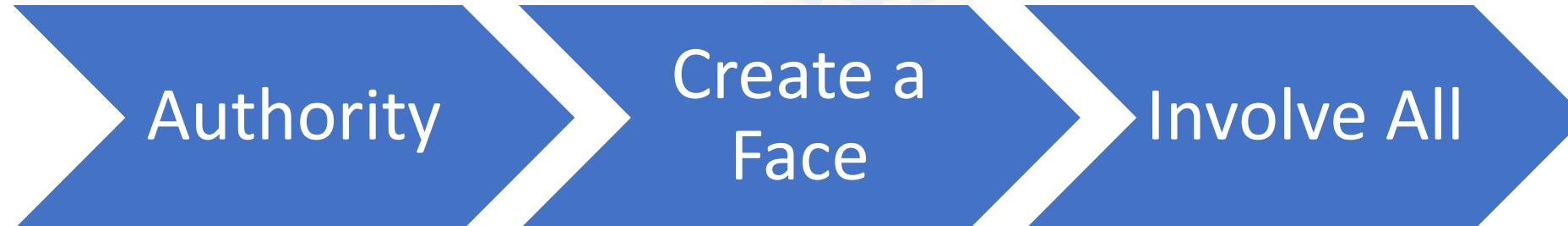
Design  
Thinking

High Level  
Outputs

# Transformation to Enterprise DevOps culture

## Establish a DevOps Centre of Excellence

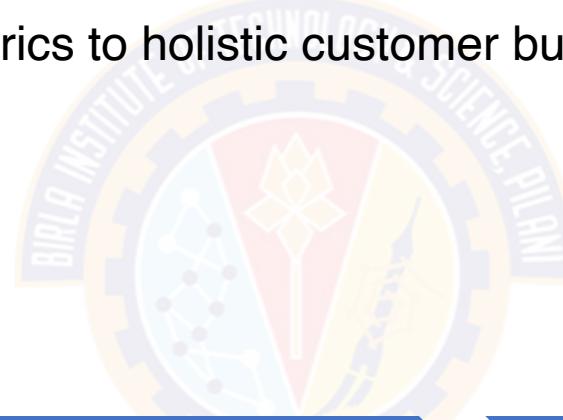
- At the right organizational level and enterprise authority
- Create a Face: It has to be led by an enterprise leader who has the support and buy-in from all the towers
- Active Participant from All Towers of Delivery & Participants must be chosen wisely
- These phases help concrete the vision and strategies of organization and help in setup the best practices to support cultural movement



# Transformation to Enterprise DevOps culture

## Establish Program Governance

- Agile and DevOps, practitioners' roles and responsibilities will change
- Need awareness, enablement and empowerment to succeed
- KPIs must shift from individual metrics to holistic customer business outcomes



Create  
Communication  
Plan

Enablement  
Program

Establish  
Program KPI's

# Transformation to Enterprise DevOps culture

## Establish Project In-take Process

- DevOps SME's conduct in-take workshops for Scrum Teams
- Automation scripts, Infrastructure assets
- Test Automation, Branching and Merging & Lessons Learned
- Fit for purpose tool selection and Application Criticality



Reusable  
Assets

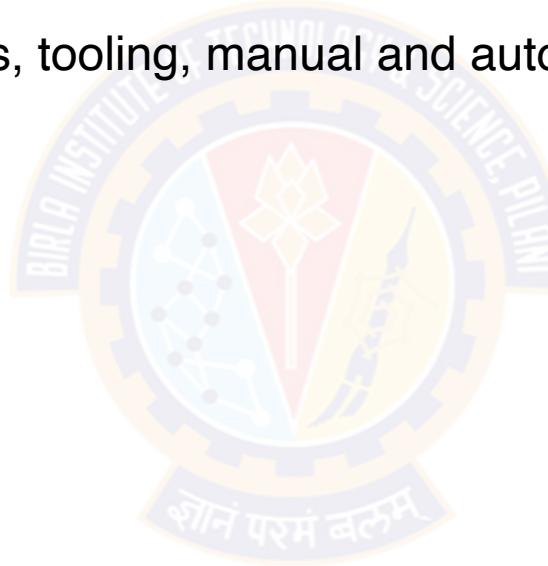
Best  
Practices

Right  
Sizing

# Transformation to Enterprise DevOps culture

## Identify and Initiate Pilots

- Value stream-mapping exercise
- Level of detail necessary:
- To identify end to end as-is process, tooling, manual and automated processes
- And skills and people



# Transformation to Enterprise DevOps culture

## Scale Out DevOps Program

- Onboard Parallel Release Trains
- Apply Intake Process
- Support, Monitor and Manage



# DevOps - People

## People

- DevOps is a cultural movement; it's all about people
- Building a DevOps culture, therefore, is at the core of DevOps adoption

**An organization may adopt the most efficient processes or automated tools possible, but they're useless without the people who eventually must execute those processes and use those tools**

- DevOps culture is characterized by a high degree of collaboration across roles, focus on business instead of departmental objectives, trust, and high value placed on learning through experimentation
- Building a culture isn't like adopting a process or a tool
- It requires social engineering of teams of people, each with unique predispositions, experiences, and biases
- This diversity can make culture-building challenging and difficult

# DevOps – People

## Managing People, Not Resources

**“Managing teams would be easy if it wasn’t for the people you have to deal with”**

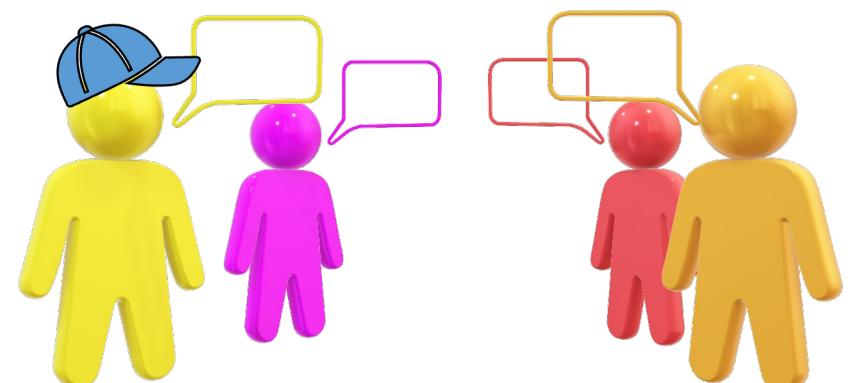
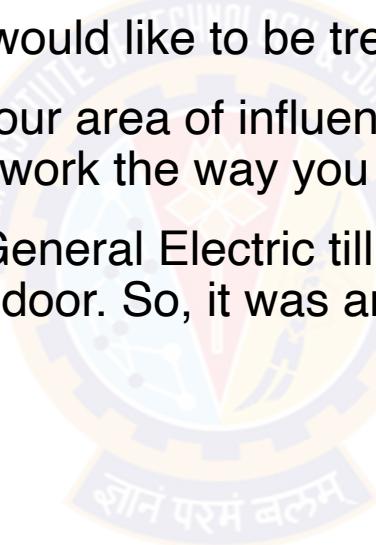
- Calling people as resources 
- A resource is something you can manage without much tailoring
- With people, that is never true
- For an Example: We have probably all been planning for projects and creating a plan that includes a certain amount of “anonymous full-time equivalents (FTEs).” Then a bit later, we start putting names to it and realize that we need more or less effort based on the actual people we have available for the project
- One Java developer is just not the same as another; and honestly, we would never consider ourselves to be just a resource whose job someone else could do with the same efficiency and effectiveness



# DevOps – People

## Don't change Organization; Change your self

- You will not be able to get the organization to change
- But what you can do is change your part of the organization
- Manage your teams the way that you would like to be treated
- As you climb higher in the hierarchy, your area of influence will increase and, with that, a larger and larger part of the organization will work the way you would like it to
- Ex. “Jack Welch” he was the CEO of General Electric till 2001, his cabin was of all transparent glass; And this cabin did not have any door. So, it was an open-door cabin. And the label on the door was “PLEASE DISTRUB ME”.



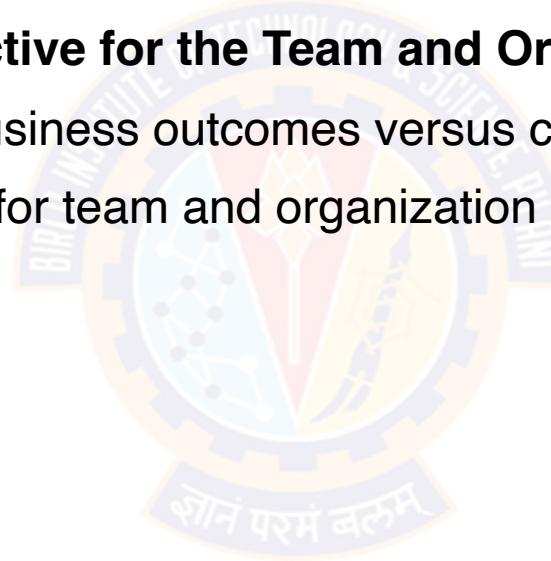
# DevOps – People

## Identifying Business Objective

- Getting everyone headed in the same direction
- And working toward the same goal

### **“Identify Common Business Objective for the Team and Organization”**

- Incent the entire team based on business outcomes versus conflicting team incentives
- Easy to measure progress of goal for team and organization

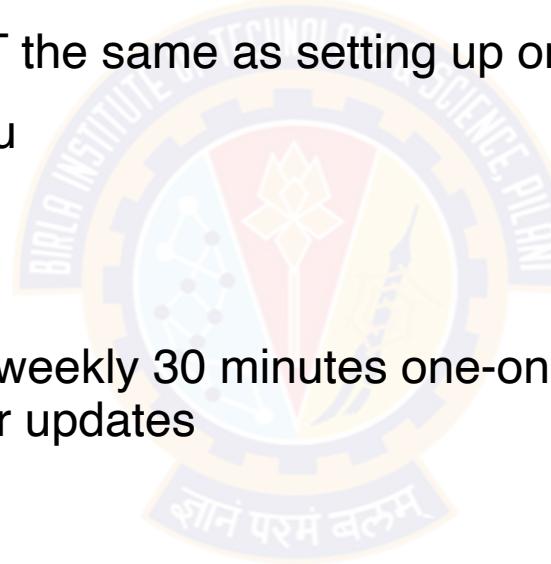


**DevOps isn't the goal. It helps you reach your goals**

# Effective Management of People

## One-On-Ones

- Regular one-on-one meetings with the people who report directly to you
  - Don't be Anonymous
  - Having an open-door policy is NOT the same as setting up one-on-ones
  - Let feel people are important to you
  - You are making time for them
- 
- Best Practices : have weekly or bi weekly 30 minutes one-on –one, learn more about person, let them raise first and provide your updates
  - Benefit to Manager?
  - Benefit to Employee?



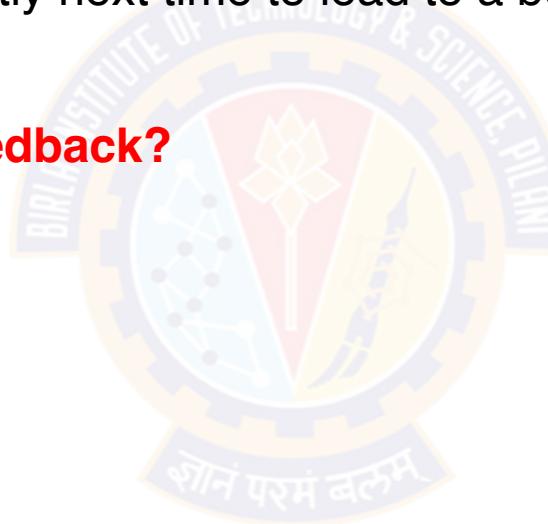
# Effective Management of People

## Feedback

- Everyone would like to get more feedback from his or her boss
- Dan Pink's mastery, autonomy, and purpose : "When you do x, y happens, which is not optimal. Can you find a way to do it differently next time to lead to a better outcome? "

### What You hear in your Feedback?

- Focus on feedback?



# Effective Management of People

## Delegation

- Feel on delegation of JOB to others?

**Managerial Economics 101**



# Effective Management of People

## Creating a Blameless Culture

- Who is being blamed?
- You will have to cover it without delegating the blame to the person in your team who is responsible
- The team will appreciate this
- The more you share with the rest of the organization the positive impact a member of your team has made, the better you will look too
- These two practices empower the people on your team to do the best job they can for you
- Whenever you have failures or problems, your root-cause analysis should not focus on who did what but on how we need to change the system so that the next person is able to avoid making the mistake again

# Effective Management of People

## Measuring Your Organizational Culture

- There are a few different ways to measure culture
- The Westrum survey measures
  - On my team, information is actively sought
  - On my team, failures are learning opportunities, and messengers of them are not punished
  - On my team, responsibilities are shared
  - On my team, cross-functional collaboration is encouraged and rewarded
  - On my team, failure causes inquiry
  - On my team, new ideas are welcomed
- There are few more suggestions:
  - I would recommend the team to my friends as a good place to work
  - I have the tools and resources to do my role well
  - I rarely think about leaving this team or my company
  - My role makes good use of my skills and abilities

**Remember that advice stated before: you can only control your part of the organization**

# DevOps – Tools

## What is DevOps Tools:

- Tools can be used to improve and maintain various aspects of culture
- Software development tools help with the process of programming, documenting, testing, and fixing bugs in applications and services
- Not restricted to specific roles, these tools are important to anyone who works on software in some capacity
  - Local development environment,
  - Version control,
  - Artifact Management,
  - Automation and Monitoring



# DevOps Tools

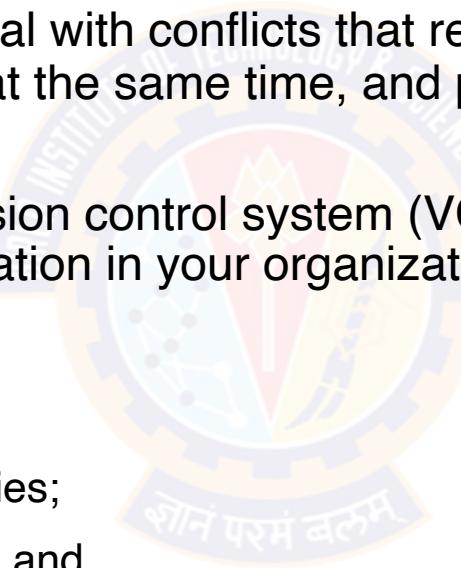
## Local Development Environment

- A consistent local development environment is critical to quickly get employees started contributing to your product
- Don't Consider this as limitation:
  - This is not to say that individuals should be locked into a single standard editor with no flexibility or customization, but rather it means ensuring that they have the tools needed to get their jobs done effectively
- Minimal requirements may vary in your environment depending on individual preferences:
  - Multiple Displays
  - High-Resolution Displays
  - Specific Keyboards, Mice, and other Input Devices

# DevOps Tools

## Version Control

- Having the ability to commit, compare, merge, and restore past revisions of objects to the repository allows for richer cooperation and collaboration within and between teams
- Version control enables teams to deal with conflicts that result from having multiple people working on the same file or project at the same time, and provides a safe way to make changes and roll them back if necessary
- When choosing the appropriate version control system (VCS) for your environment, look for one that encourages the sort of collaboration in your organization that you want to see
  - Opening and forking repositories;
  - Contributing back to repositories;
  - Contributions to your own repositories;
  - Defining processes for contributing; and
  - Sharing commit rights



# DevOps Tools

## Artifact Management

- An artifact is the output of any step in the software development process
- When choosing between a simple repository and more complex feature-full repository, understand the cost of supporting additional services as well as inherent security concerns
- An artifact repository should be:
  - Secure;
  - Trusted;
  - Stable;
  - Accessible; and
  - Versioned
- You can store a versioned common library as an artifact separate from your software version control, allowing all teams to use the exact same shared library



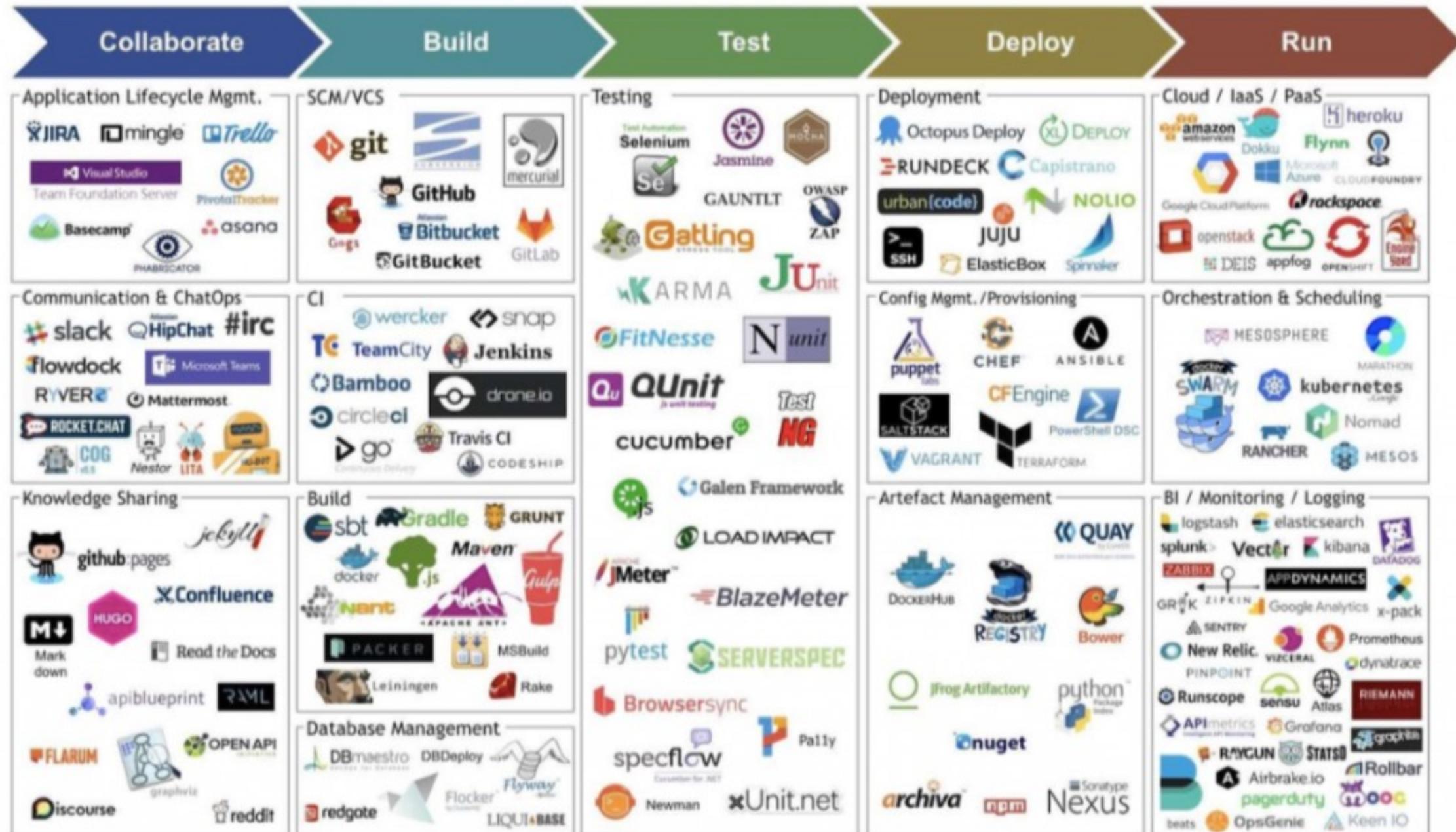
# DevOps Tools

## Other Automation Tools

- Automation tools reduce labor, energy, and/or materials used with a goal of improving quality, precision, and accuracy of outcomes
- Server Installation
  - Server installation is the automation of configuring and setting up individual servers
- Infrastructure Automation
  - Configuration Management
  - Capacity Management
- System Provisioning
- Test and Build Automation
  - On-demand automation
  - Scheduled automation
  - Triggered automation
  - Smoke testing
  - Regression testing
  - Usability testing



# DevOps Tools Ecosystem



| Periodic Table of DevOps Tools (V2) (V1) |    |                  |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|--|----|------------------|----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
|  |    |                  |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |    |                  |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1  | Fm |                  |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gh                                       |    | Github           |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3  | Os | 4                | Pd |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gt                                       |    | Dm               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Git                                      |    | Dbmaestro        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11                                       | Fm | 12               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bb                                       |    | Lb               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bitbucket                                |    | Liquibase        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 19                                       | Os | 20               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gl                                       |    | Rg               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| GitLab                                   |    | Redgate          |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 21                                       | Os | 22               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Mv                                       |    | Gr               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Maven                                    |    | Gradle           |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 23                                       | Os | 24               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fn                                       |    | Se               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FitNesse                                 |    | Selenium         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 25                                       | Fr | 26               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Se                                       |    | Ga               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Selenium                                 |    | Gating           |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 27                                       | Fr | 28               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Dh                                       |    | Jn               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Docker Hub                               |    | Jenkins          |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 29                                       | Os | 30               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ba                                       |    | Tr               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bamboo                                   |    | Travis CI        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 31                                       | Pd | 32               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gd                                       |    | Sf               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Deployment Manager                       |    | SmartFrog        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 33                                       | Os | 34               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cn                                       |    | Bc               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Consul                                   |    | bcfg2            |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 35                                       | Os | 36               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Lxc                                      |    | Rs               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Linux Containers                         |    | Rackspace        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 37                                       | Os | 38               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sv                                       |    | Dt               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Subversion                               |    | Dabtical         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 39                                       | Os | 40               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Gt                                       |    | Gp               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Grunt                                    |    | Gulp             |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 41                                       | Os | 42               | Fr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cu                                       |    | Cj               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 43                                       | Os | 44               | Fr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cucumber                                 |    | Cucumber.js      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 45                                       | Os | 46               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Npm                                      |    | Cs               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| npm                                      |    | Codeship         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 47                                       | Pd | 48               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Vs                                       |    | Cr               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Visual Studio                            |    | CircleCI         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 49                                       | F  | 50               | F  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cp                                       |    | Ju               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Capistrano                               |    | Juju             |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 51                                       | Os | 52               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Rd                                       |    | Cf               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Rundeck                                  |    | CFEngine         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 53                                       | Fr | 54               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ds                                       |    | Op               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Swarm                                    |    | OpenStack        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 55                                       | En | 56               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Hx                                       |    | Dp               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Helix                                    |    | Delphix          |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 57                                       | Fr | 58               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Sb                                       |    | Mk               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sbt                                      |    | Make             |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 59                                       | Os | 60               | Fr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Jt                                       |    | Jm               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| JUnit                                    |    | JMeter           |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 61                                       | Fr | 62               | Fr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Tn                                       |    | Ay               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TestNG                                   |    | Artifactory      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 63                                       | Os | 64               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Tc                                       |    | Sh               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| TeamCity                                 |    | Shipable         |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 65                                       | Fm | 66               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cc                                       |    | Ry               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CruiseControl                            |    | RapidDeploy      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 67                                       | En | 68               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Oc                                       |    | Cy               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Octopus Deploy                           |    | CodeDeploy       |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 69                                       | En | 70               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| No                                       |    | Ca               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 71                                       | Os | 72               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CA Nolio                                 |    | Continua CI      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 73                                       | En | 74               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cw                                       |    | Id               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 75                                       | Os | 76               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Msb                                      |    | Pk               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MSBuild                                  |    | Packer           |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 77                                       | Fr | 78               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Mc                                       |    | Xltv             |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Mocha                                    |    | XL TestView      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 79                                       | En | 80               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Jm                                       |    | Jasmine          |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 81                                       | Os | 82               | Os |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Nx                                       |    | Co               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Nexus                                    |    | Continuum        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 83                                       | Fm | 84               | Pd |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ca                                       |    | So               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Continua CI                              |    | Solano CI        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 85                                       | En | 86               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Xld                                      |    | EB               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Xl Deploy                                |    | ElectricBox      |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 86                                       | Fm | 87               | Fm |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Dp                                       |    | Deploybot        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Deploybot                                |    | UrbanCode Deploy |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 88                                       | En | 89               | Fr |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ud                                       |    | Fl               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| UrbanCode Deploy                         |    | Fleet            |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 89                                       | Fr | 90               | En |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fl                                       |    | Os               |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fleet                                    |    | OpenShift        |    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Follow @xebialabs

XebiaLabs  
Deliver Faster

|     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |    |     |    |     |    |     |    |     |    |     |    |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|-----|----|-----|----|-----|----|-----|----|-----|----|
| 91  | En | 92 | En | 93 | En | 94 | En | 95 | En | 96 | En | 97 | En | 98 | Pd | 99 | Fm | 100 | Pd | 101 | Fm | 102 | Fm | 103 | Fm | 104 | Pd | 105 | En |
| Xlr |    | Ur |    | Bm |    | Hp |    | Au |    |    |    |    |    |    |    |    |    |     |    |     |    |     |    |     |    |     |    |     |    |



# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Could for DevOps & Version Control System

- Cloud as a catalyst for DevOps
- Evolution of Version Control
- Version Control System Types
  - Centralized Version Control Systems
  - Distributed Version Control Systems
- Introduction to GIT
- GIT Basics commands
- Creating Repositories, Clone, Push, Commit, Review
- Git Branching
- Git Managing Conflicts
- Git Tagging
- Git workflow
  - Centralized Workflow
  - Feature Branch Workflow
- Best Practices- clean code



# DevOps Tools

## Cloud as a Catalyst for DevOps

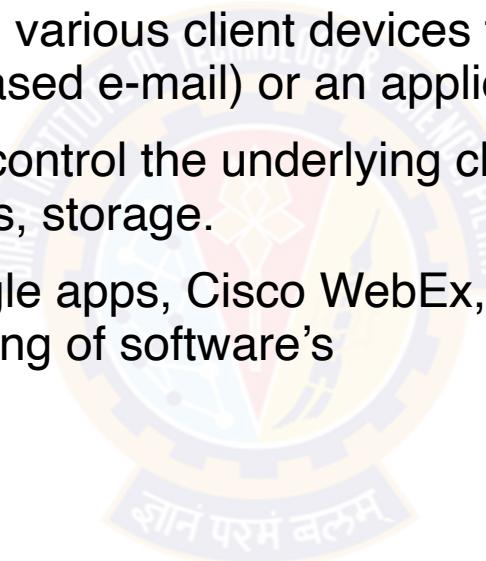
- Characterization of the cloud by National Institute of Standards and Technology (NIST)
  - On-demand self-service
  - Broad network access
  - Resource pooling
  - Rapid elasticity
    - It is the Capabilities can be elastically provisioned and released
- Measured service
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)



# Cloud as a Catalyst for DevOps

## Software as a Service (SaaS)

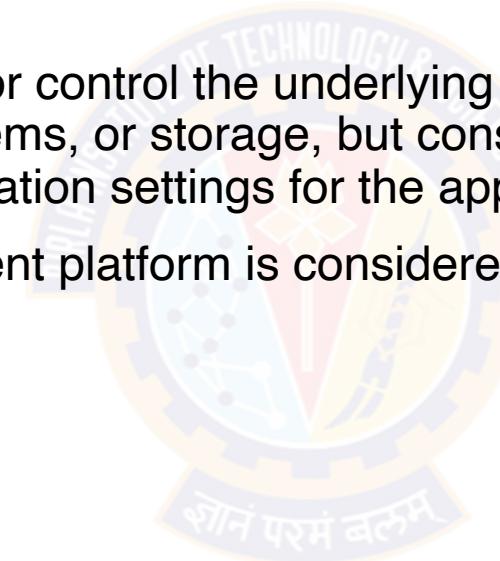
- In this the consumer is provided the capability to use the provider's applications running on a cloud infrastructure
- The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based e-mail) or an application interface
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, storage.
- For an example, you can relate google apps, Cisco WebEx, as a Service, Office 365 service, where Provider deals with the licensing of software's



# Cloud as a Catalyst for DevOps

## Platform as a Service (PaaS)

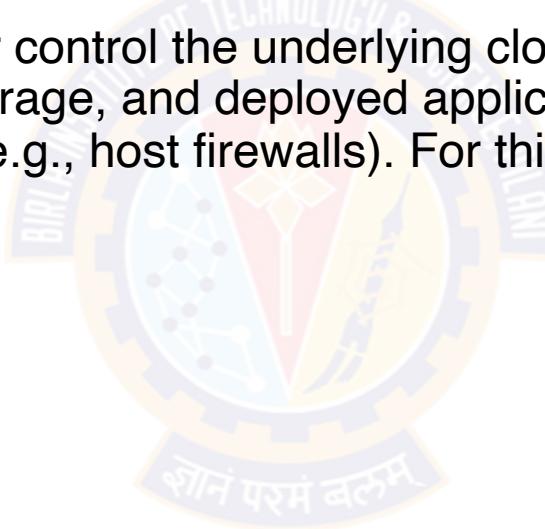
- The consumer is provided the capability to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider
- The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but consumer has control over the deployed applications and possibly configuration settings for the application-hosting environment
- For an Example: .NET Development platform is considered as a platform



# Cloud as a Catalyst for DevOps

## Infrastructure as a Service (IaaS)

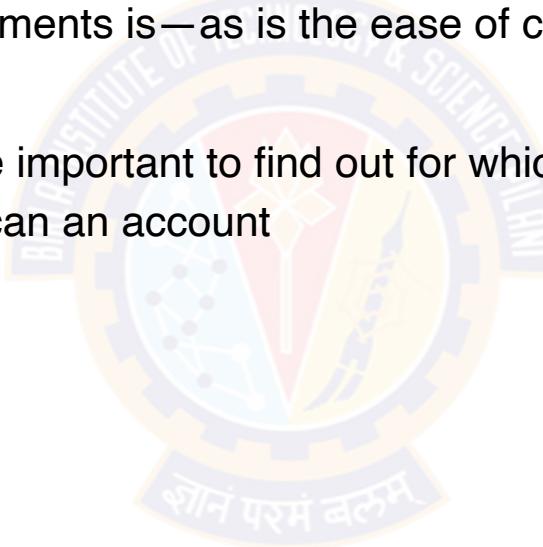
- The consumer is provided the capability to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications
- The consumer does not manage or control the underlying cloud infrastructure but consumer has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls). For this you can consider any Server Provisioning is IaaS,



# Cloud as a Catalyst for DevOps

## Three of the unique aspects of the cloud that impact DevOps

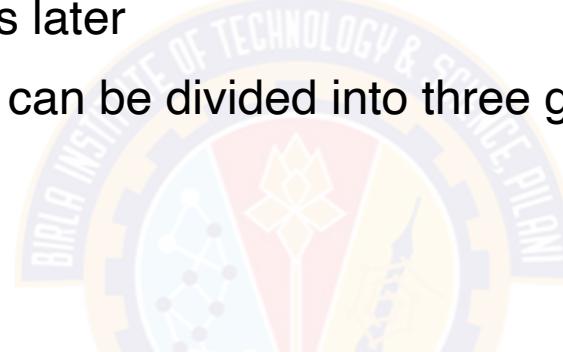
- Three of the unique aspects of the cloud that impact DevOps:
- The ability to create and switch environments simply
  - Simply create and migrate environments is—as is the ease of cloning new instances
- The ability to create VMs easily
  - Administering the running VMs are important to find out for which VM we are paying but not using it
  - Tool such as, Janitor Monkey to scan an account
- The management of databases



# Evolution of Version Control

## Generations of VCS

- What is “version control”, and why should you care?
- Definition: Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later
- The history of version control tools can be divided into three generations



| Generation        | Networking  | Operations         | Concurrency         | Example Tool    |
|-------------------|-------------|--------------------|---------------------|-----------------|
| First Generation  | None        | One file at a time | Locks               | RCS, SCCS       |
| Second Generation | Centralized | Multi-file         | Merge before commit | CVS, Subversion |
| Third Generation  | Distributed | Changesets         | Commit before merge | Bazaar, Git     |

# Version Control System

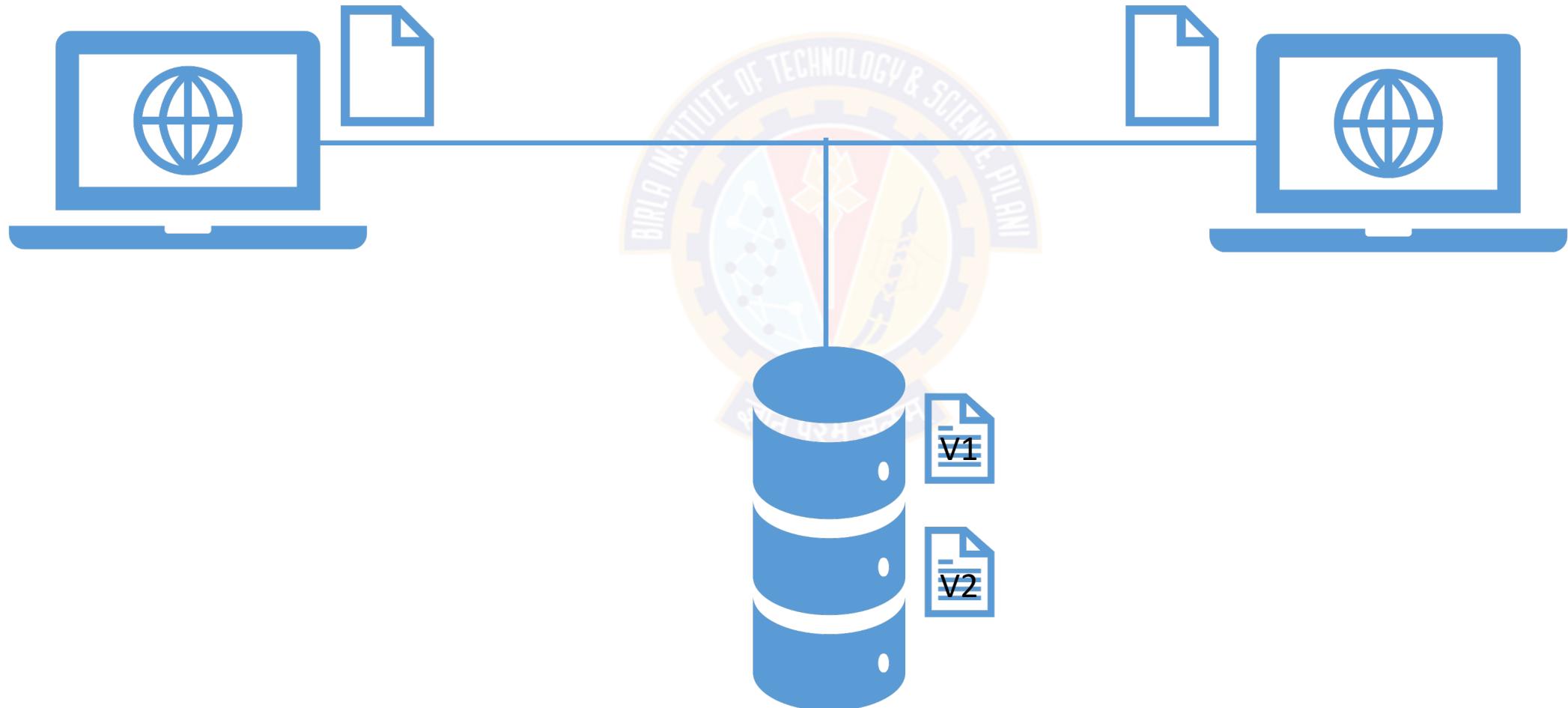
## Benefits

- Change history
- Concurrent working (Collaboration)
- Traceability
- Backup & Restoration



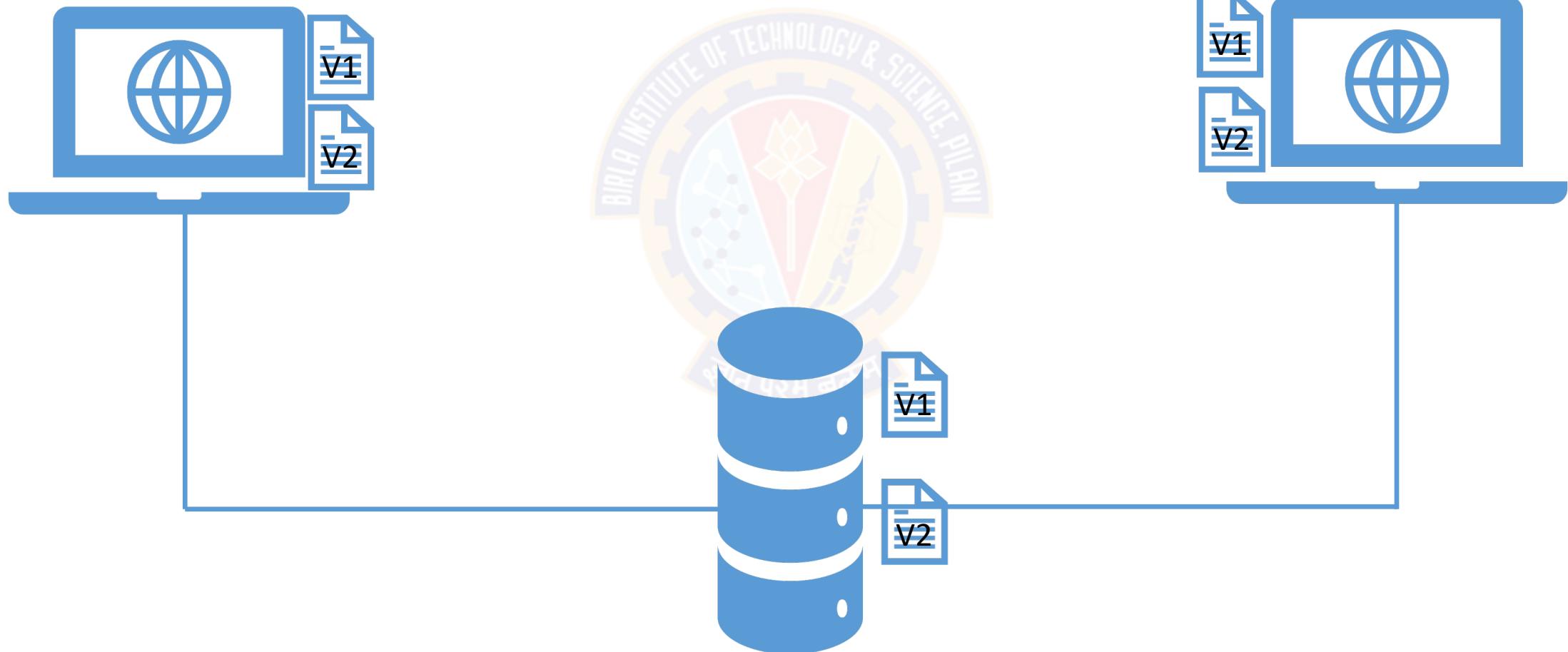
# Version Control System Types

## Centralized source code management System



# Version Control System Types

## Distributed source code management System



# CVCS Vs. DVCS

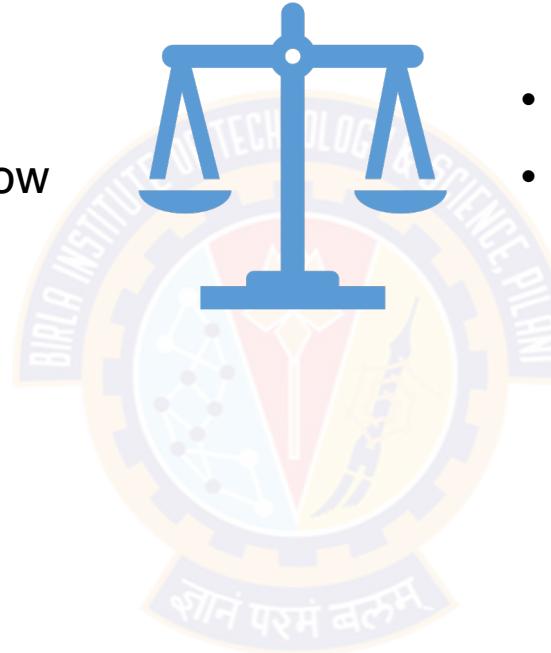
Lets discuss con's

## CVCS

- Single point of failure
- Remote commits are slow
- Continuous connection

## DVCS

- Need more space
- Bandwidth for large project



# Available Tools

## CVCS

- Open source:
  - Subversion (SVN)
  - Concurrent Versions System (CVS)
  - Vesta
  - OpenCVS
- Commercial:
  - AccuRev
  - Helix Core
  - IBM Rational ClearCase
  - Team Foundation Server (TFS)

## DVCS

- Open source:
  - Git
  - Bazaar
  - Mercurial
- Commercial:
  - Visual Studio: Team Services
  - Sun WorkShop: TeamWare
  - Plastic SCM – by Codice Software, Inc
  - Code Co-op



# Git & GitHub

## What we will learn in this session

- Git & GitHub relationship
- Prerequisite
- Foundation of Git



Concept of Git



Understanding GitHub



Beyond the basics

# Git

## What is Git



Popular Source Control system



Distributed system



Free (Open Source tool)



# Git

## Why use Git

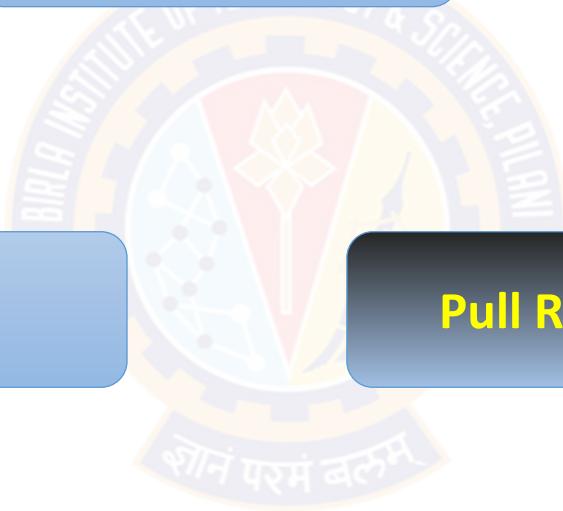
Fast

Disconnected

Powerful yet easy

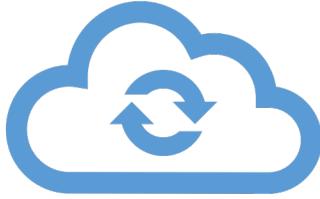
Branching

Pull Requests



# GitHub

## What is GitHub?



Hosting service on Git



More than just source control for your code

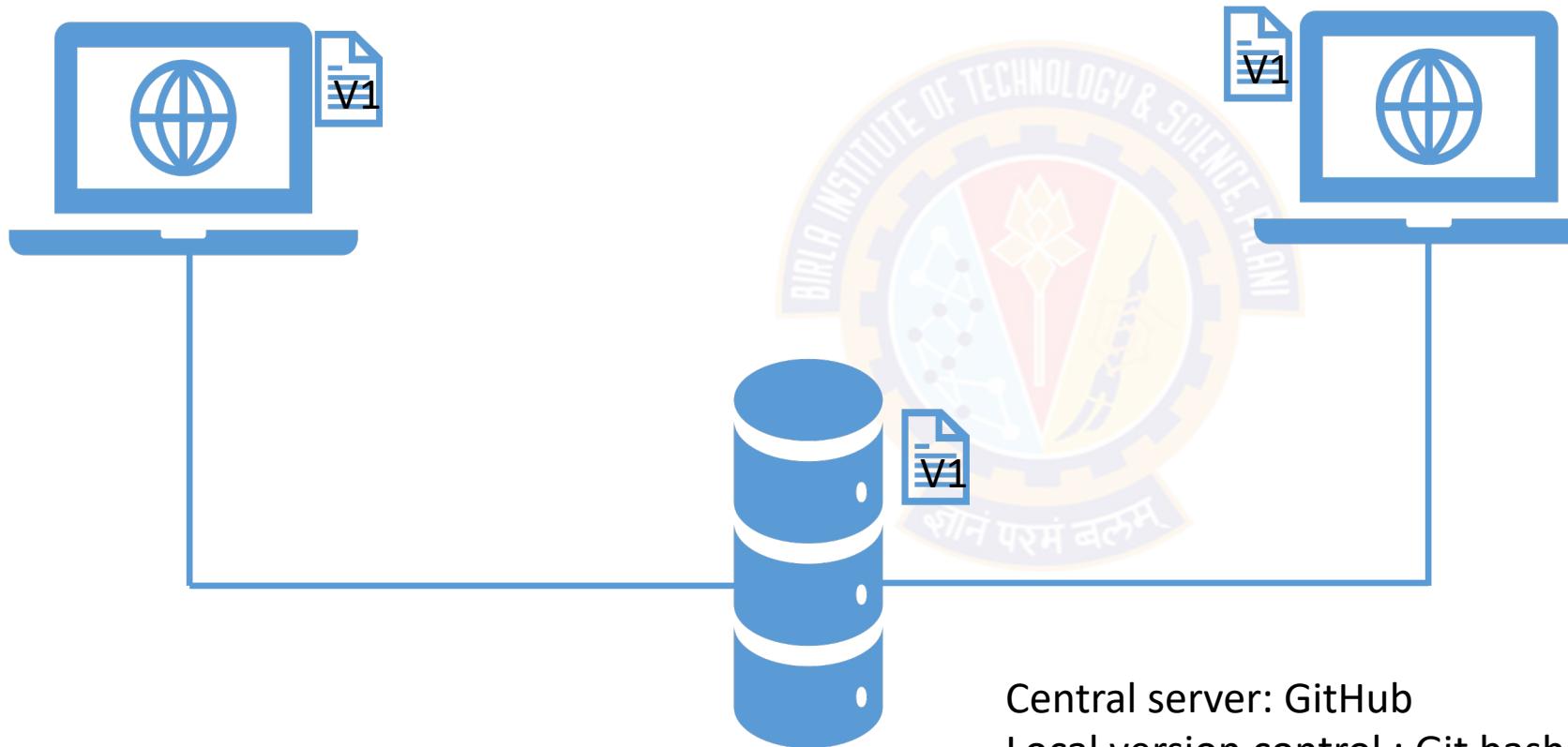
- Issue management, Working with Teams, etc.,



Free & Paid options

# Git & GitHub

## Relationship



# Git

## Working with Git



**Console**  
We will use this



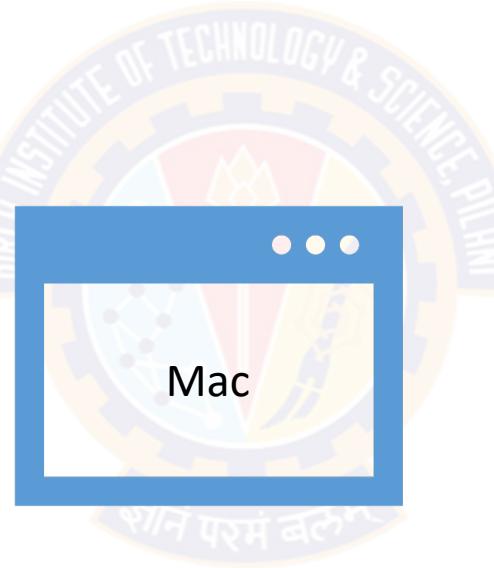
**GUI**  
GitHub Desktop  
Source tree



# Git & GitHub

## Getting your system ready

- Install appropriate Git bash form official site
- Command line (Git bash)
- Server account (GitHub account)



Support for all platform

# Git Foundation

## The 3 State of Git



# Git Foundation

## The 3 areas of Git

Working  
directory

Staging  
area

.git  
repo

Remote repo  
GitHub

# Git Foundation

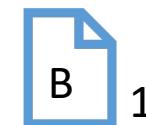
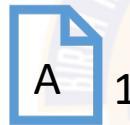
## Concepts of Snapshots in Git & GitHub

First  
version

Version 2

Version 3

Version 4



# Git Foundation

## Commits in Git



# Git

## Basic Commands

```
$ git          $ git push
$ git config   $ git fetch
$ git init     $ git merge
$ git clone    $ git pull
$ git status   $ git log
$ git add      $ git reset
$ git commit   $ git revert
$ git branch
$ git checkout
```

# GitHub

## GitHub's main Features



Code management



Pull requests



Issues



CICD



Global search

## Connecting with local machine

### HTTPS

Requires user name & password

### SSH

Easier to work with



## Working with repository



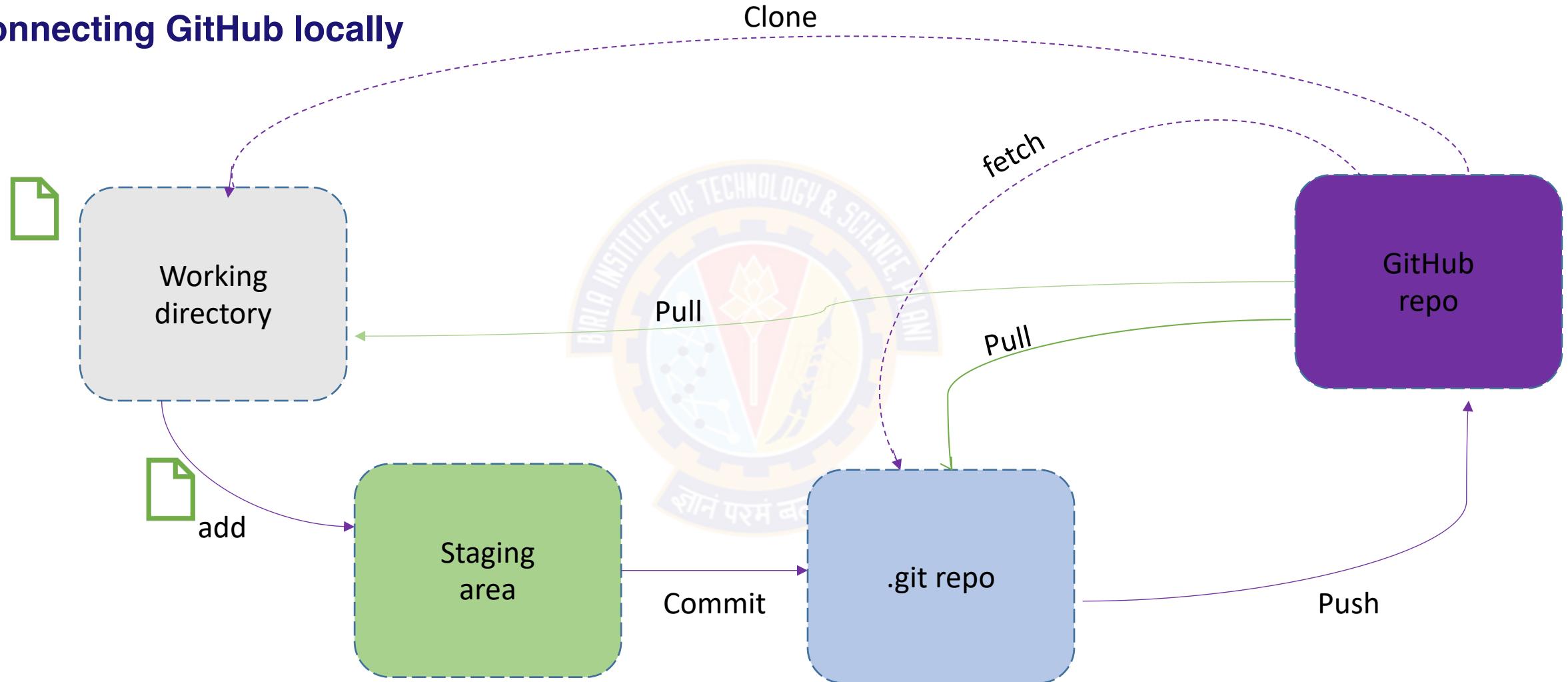
Repositories are building block of GitHub

“Folder” for your project

Public or Private

# GitHub

## Connecting GitHub locally



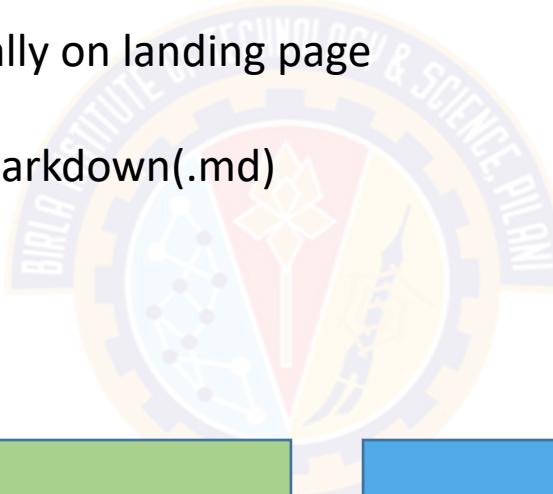
## Working with special files



README is special file known by GitHub

Rendered automatically on landing page

Typically written in markdown(.md)



Other files:

LICENSE

CHANGELOG

CODE\_OF\_CONDUCT

CONTRIBUTORS

SUPPORT

CODEOWNERS

## Repository Feature

TOPICS

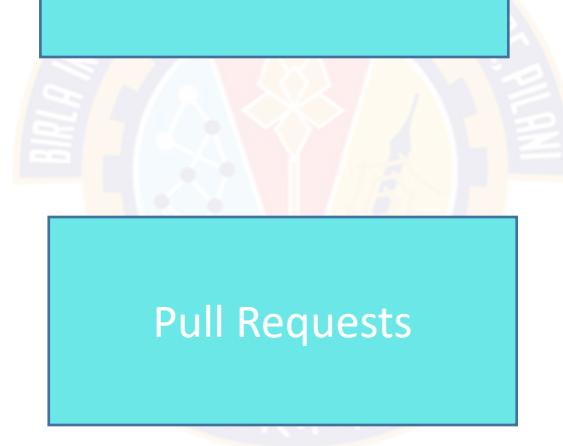
ISSUES

Insight

PROJECTS

Pull Requests

Settings

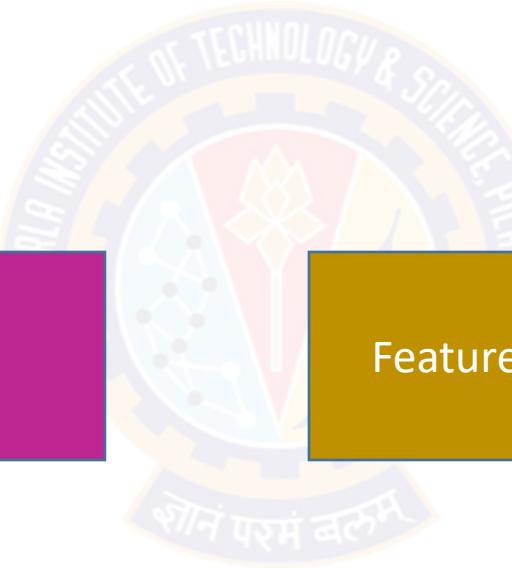


# Git

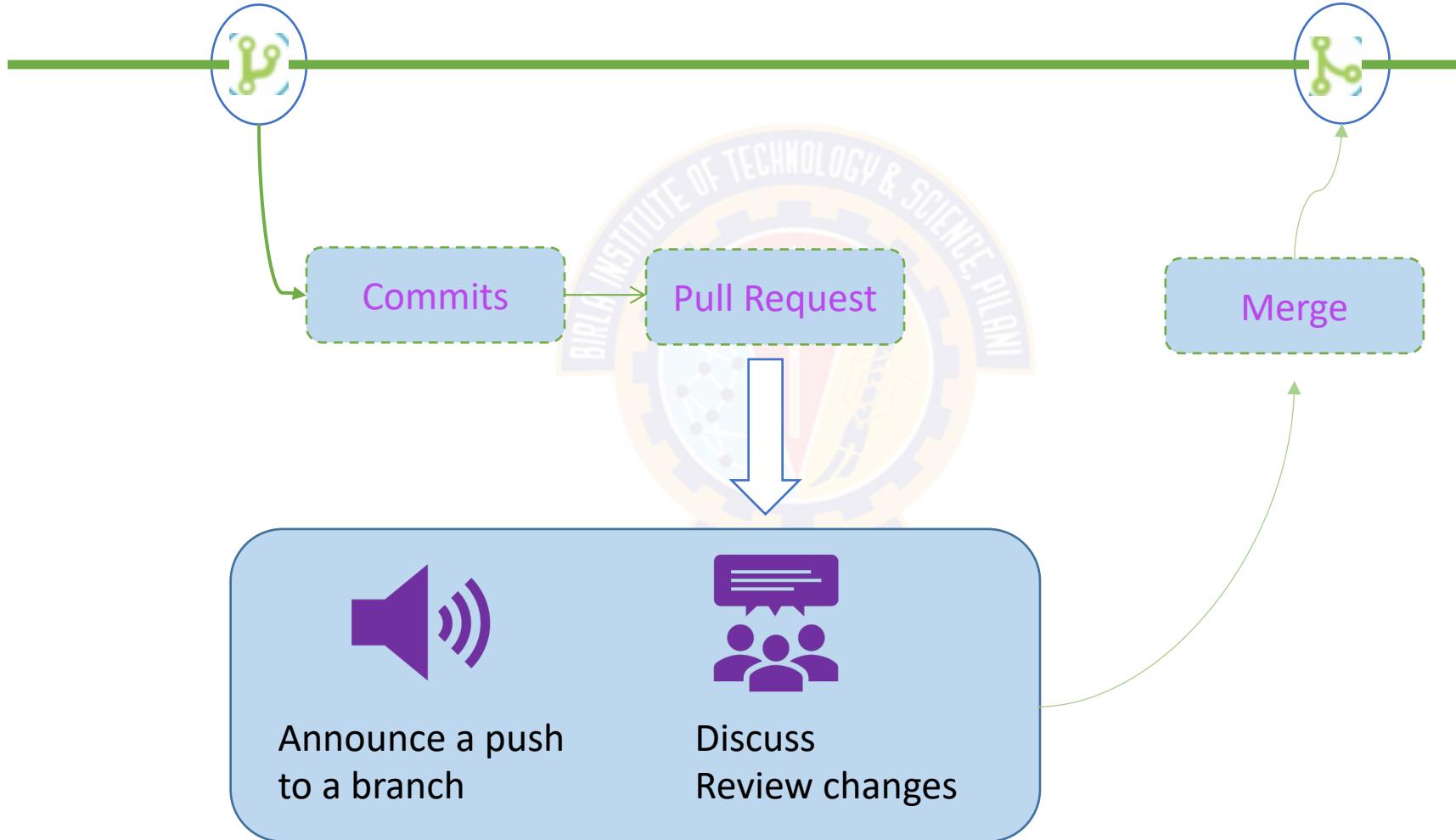
## Workflow

Centralized

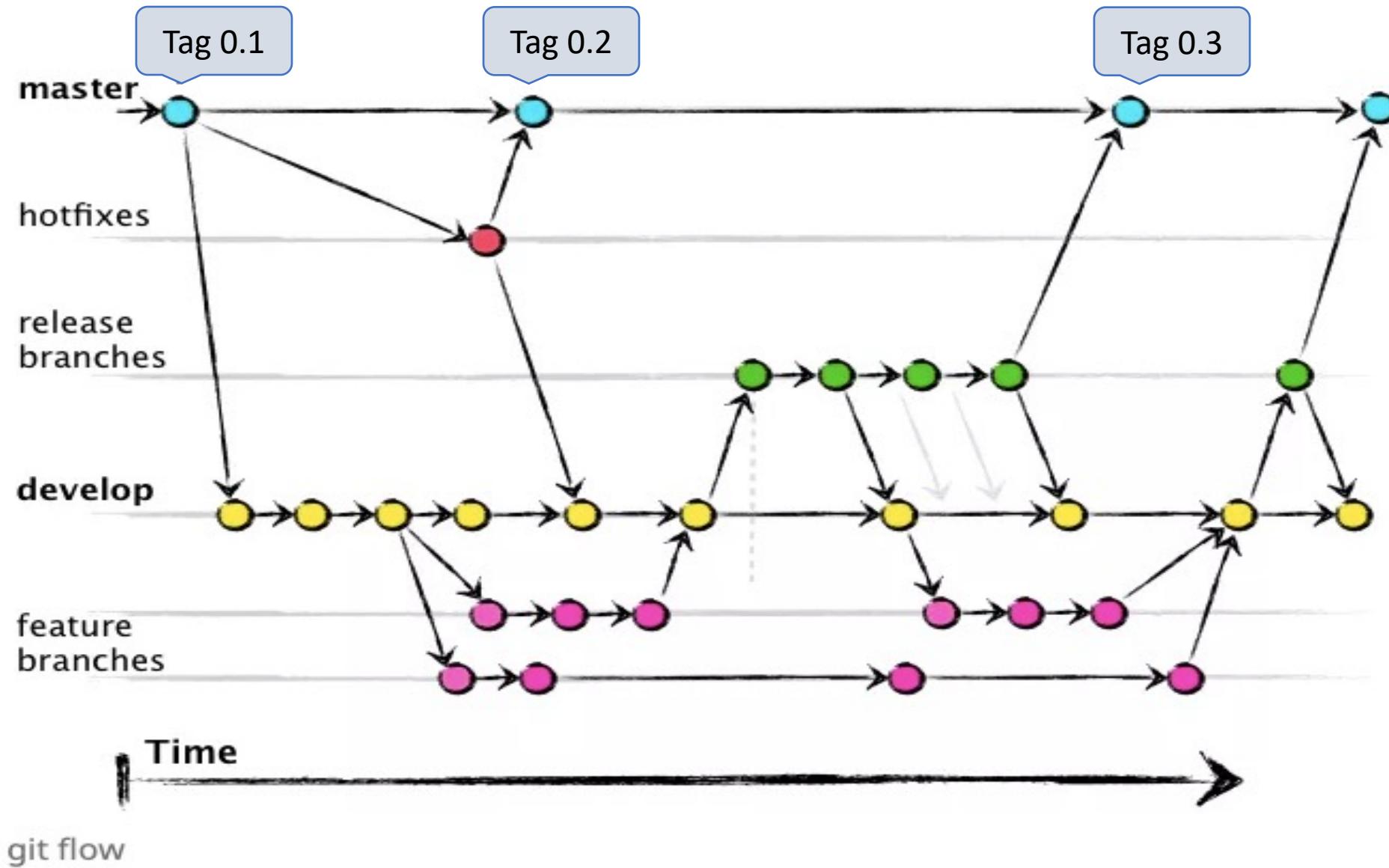
Feature workflow



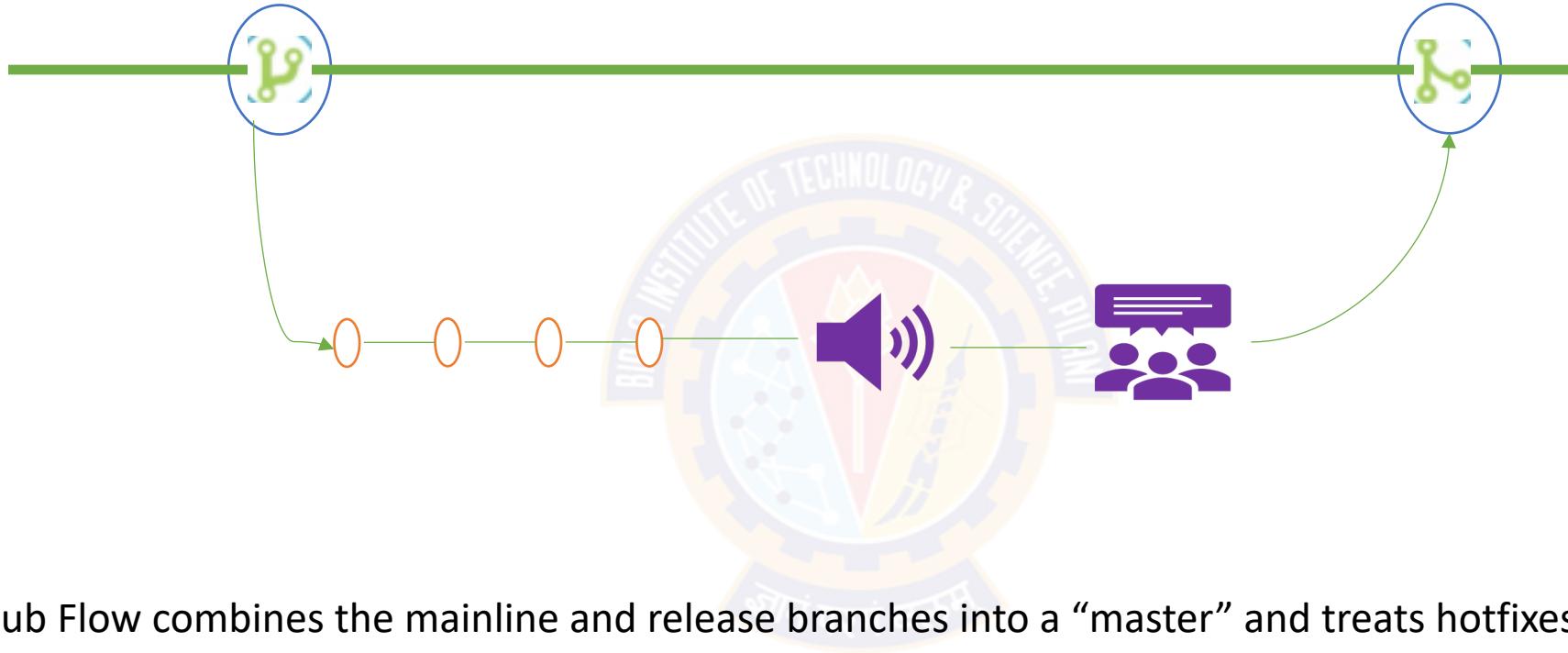
## Branching



## Git Flow



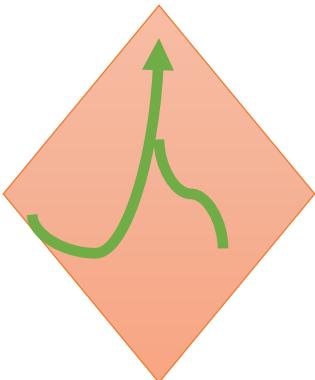
## GitHub Flow



GitHub Flow combines the mainline and release branches into a “master” and treats hotfixes just like feature branches.

# Git & GitHub

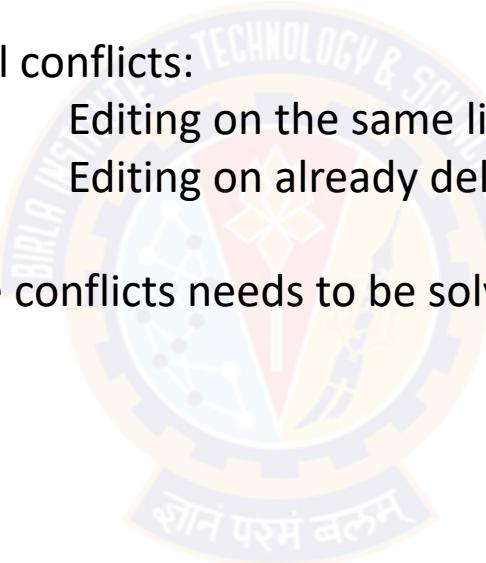
## Merging with conflicts



Typical conflicts:

- Editing on the same line
- Editing on already deleted file

Merge conflicts needs to be solved before merge happen (Manual intervention)



# Git Command

## Revert

- Git revert will create a new commit
- Git revert undoes a single commit
- It is a safe way if undo

## Reset

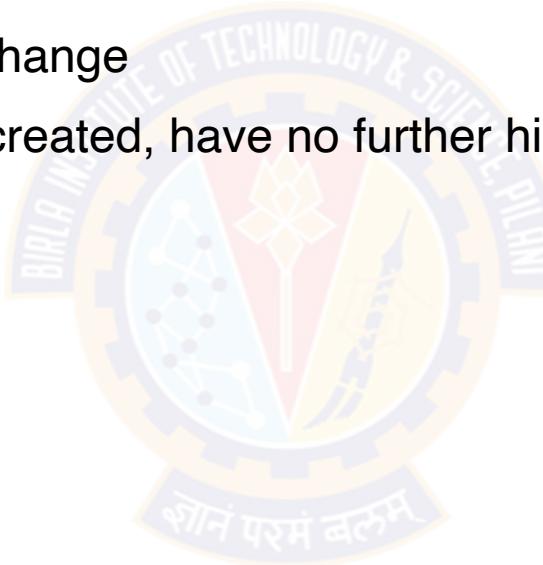
- Git reset command is a complex
- Dangerous
- Git reset has three primary form of invocation
  - `git reset --hard HEAD`
  - `git reset --mixed HEAD`
  - `git reset --soft HEAD`



# Git command

## Git Tag

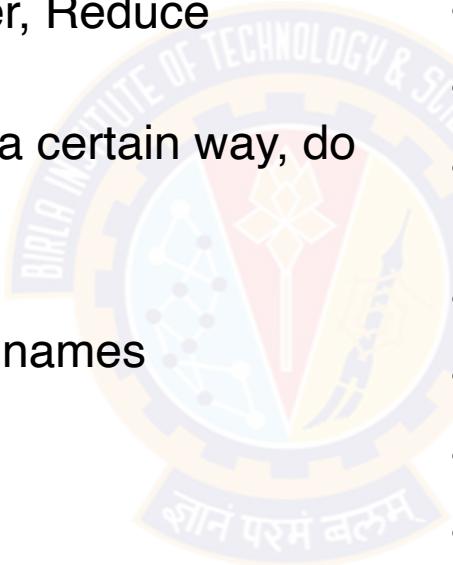
- Tags are ref's that point to specific points in Git history
- Marked version release (i.e. v1.1.1)
- A tag is like a branch that doesn't change
- Unlike branches, tags, after being created, have no further history of commits
- Common Tag operations:
  - Create tag
  - List tags
  - Delete tag
  - Sharing tag



# Git

## Clean Code

- Follow standard conventions
- Keep it simple, Simpler is always better, Reduce complexity as much as possible
- Be consistent i.e. If you do something a certain way, do all similar things in the same way
- Use self explanatory variables
- Choose descriptive and unambiguous names
- Keep functions small
- Each Function should do one thing
- Use function names descriptive
- Prefer to have less arguments
- Always try to explain yourself in code; put proper comments
- Declare variables close to their usage
- Keep lines short
- Code should be readable
- Code should be fast
- Code should be generic
- Code should be reusable





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Version Control System

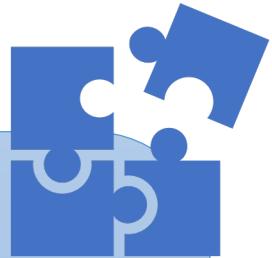
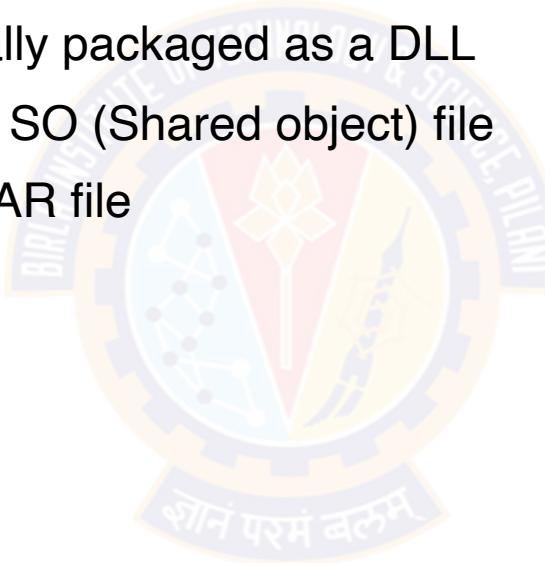
- Manage Dependencies
- Automate the process of assembling software components with build tools
- Use of Build Tools
  - Maven
  - Gradle



# Component

## Why Component based design

- large-scale code structure within an application
- also refer as “modules”
- In Windows, a component is normally packaged as a DLL
- In UNIX, it may be packaged as an SO (Shared object) file
- In the Java world, it is probably a JAR file



### Benefits

- encouraging reuse and good architectural (loose coupling)
- efficient ways for large teams of developers to collaborate

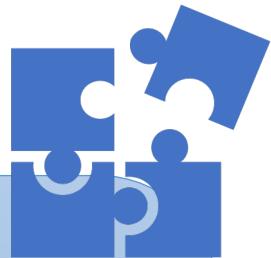
# Component

## Challenges of component based design

- Components form a series of dependencies, which in turn depend on external libraries
- Each component may have several release branches



To overcome this we need to follow the best practices

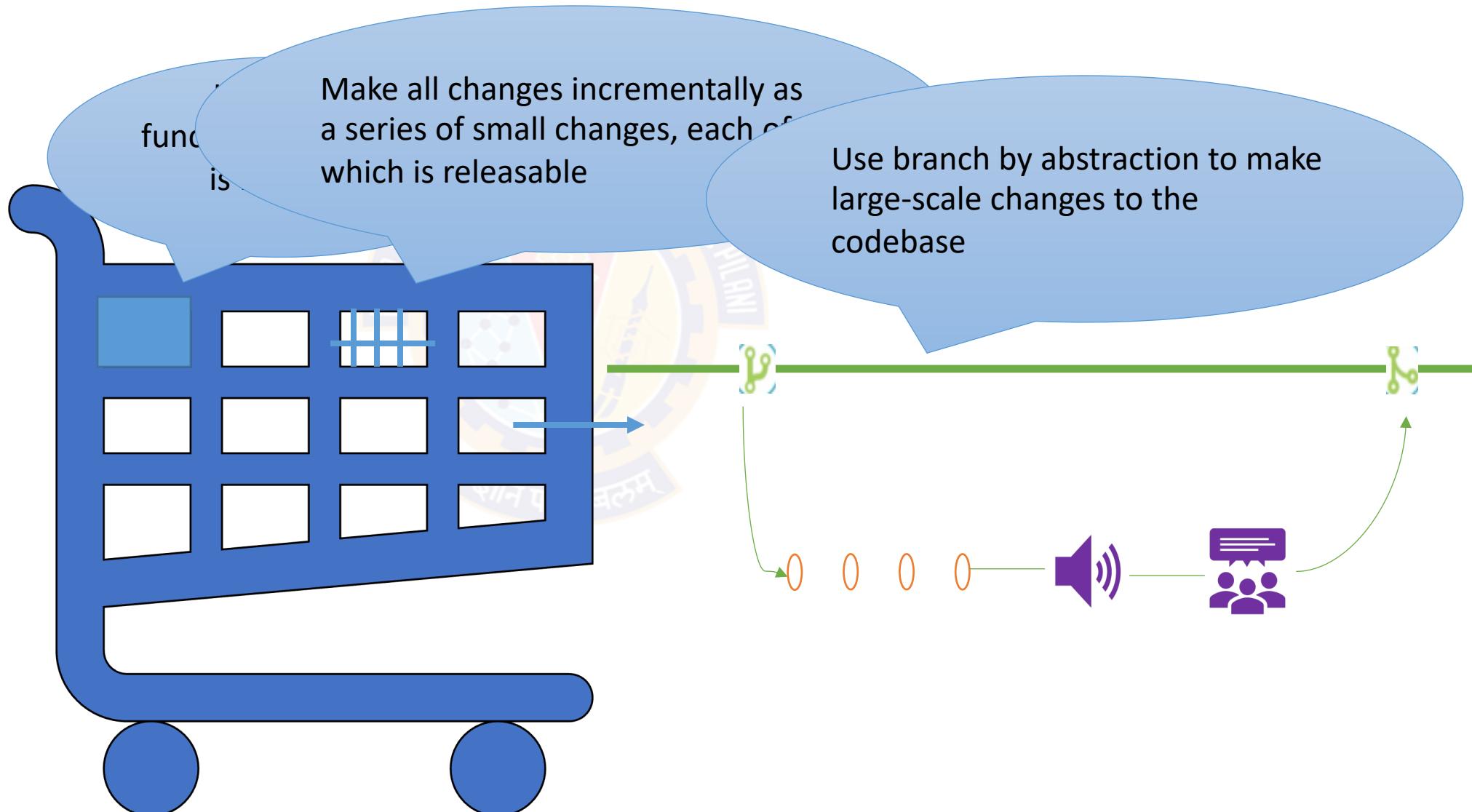


### Delay in release

- finding good versions of each of these components
- Then assembled them into a system which even compiles is an extremely difficult process

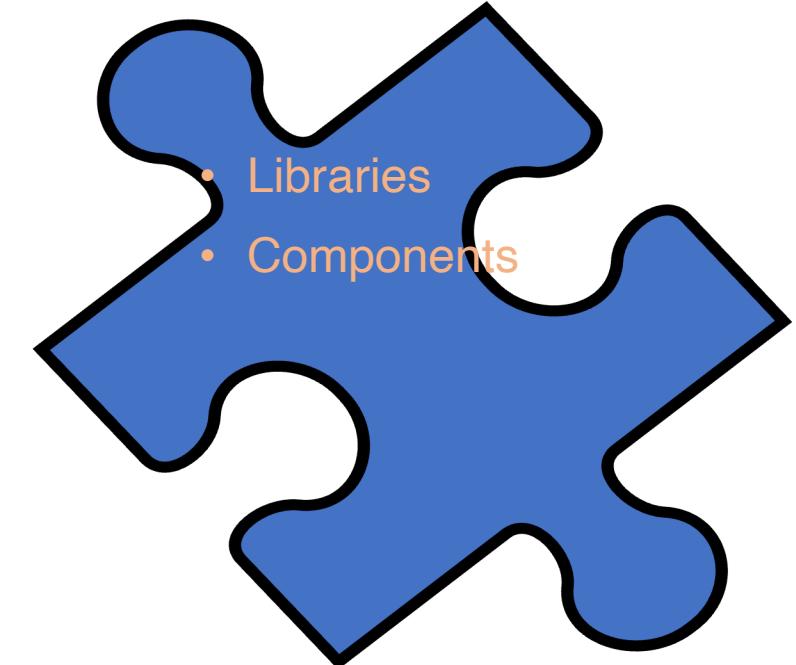
# Best Practices for component based design

## Keeping Your Application Releasable



# Best Practices for component based design

## Mange your applications dependencies



# Dependencies

## What is dependencies

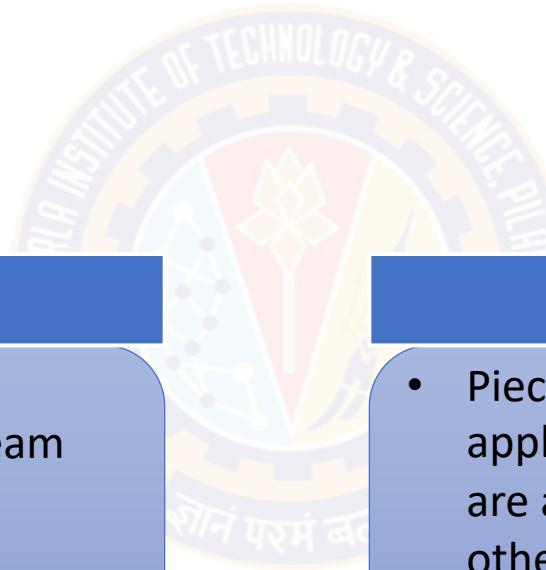
- Dependency occurs when one piece of software depends upon another in order to build or run
- Software applications → host operating environment
- Like:
- Java applications. → JVM
- .NET applications → CLR,
- Rails applications → Ruby and the Rails framework,
- C applications → C standard library etc.



# Dependencies

## Dependencies can be

- Build time dependencies and Run time dependencies
- Libraries and components



### Libraries

- Software packages that your team does not control, other than choosing which to use
- Libraries are Usually updated rarely

### Components

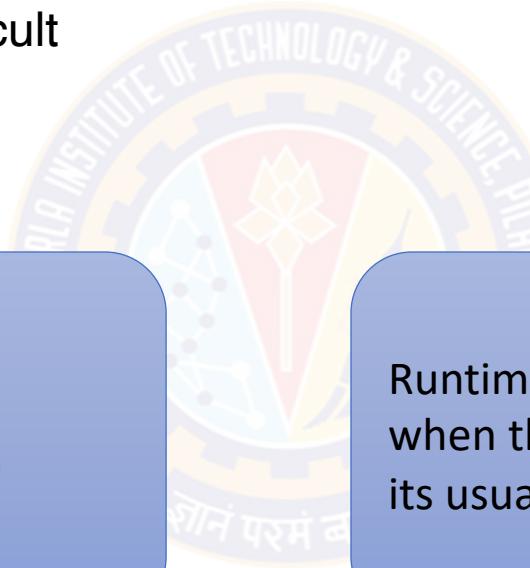
- Pieces of software that your application depends upon, but which are also developed by your team, or other teams in your organization
- Components are usually updated frequently

This distinction is important because when designing a build process, there are more things to consider when dealing with components than libraries

# Dependencies

## Build time dependencies and Run time dependencies

- Ex. In C and C++, your build-time dependencies are simply header files, while at run time you require a binary to be present in the form of a dynamic-link library (DLL) or shared library (SO)
- Managing dependency can be difficult



Build-time dependencies must be present when your application is compiled and linked (if necessary)

Runtime dependencies must be present when the application runs, performing its usual function

# Most common dependency problem

## With libraries at run time

- Dependency Hell also refer as DLL Hell
- Occurs when an application depends upon one particular version of something, but is deployed with a different version, or with nothing at all



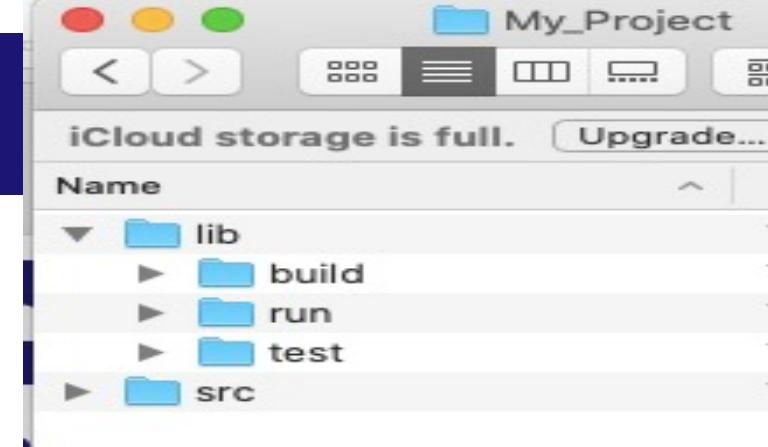
# Managing Libraries

## Implementing Version Control

- Simplest solution, and will work fine for small projects
- Create lib directory
- Use a naming convention for libraries that includes their version number; you know exactly which versions you're using

### Benefit:

- Everything you need to build your application is in version control
- Once you have a local check-out of the project repository, you know you can repeatably build the same packages that everybody else has



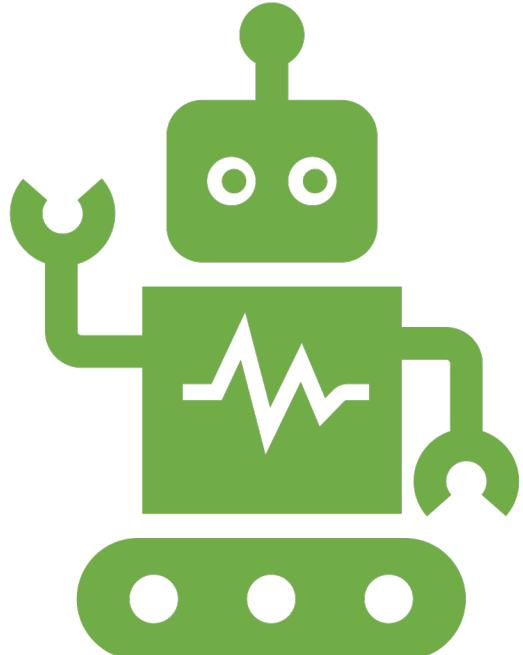
### Problems:

- your checked-in library repository may become large and it may become hard to know which of these libraries are still being used by your application
- Another problem crops up if your project must run with other projects on the same platform
- Manually managing transitive dependencies across projects rapidly becomes painful

# Managing Libraries

## Automated

- Declare libraries and use a tool like Maven or Ivy or gradle
- To download libraries from Internet repositories or (preferably) your organization's own artifact repository



# Managing Component

## Components to be separated from Codebase

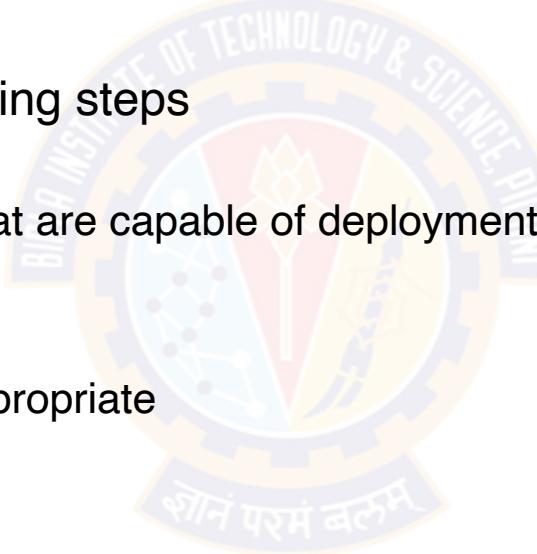
- Part of your codebase needs to be deployed independently (for example, a server or a rich client)
- It takes too long to compile and link the code



# Managing Component

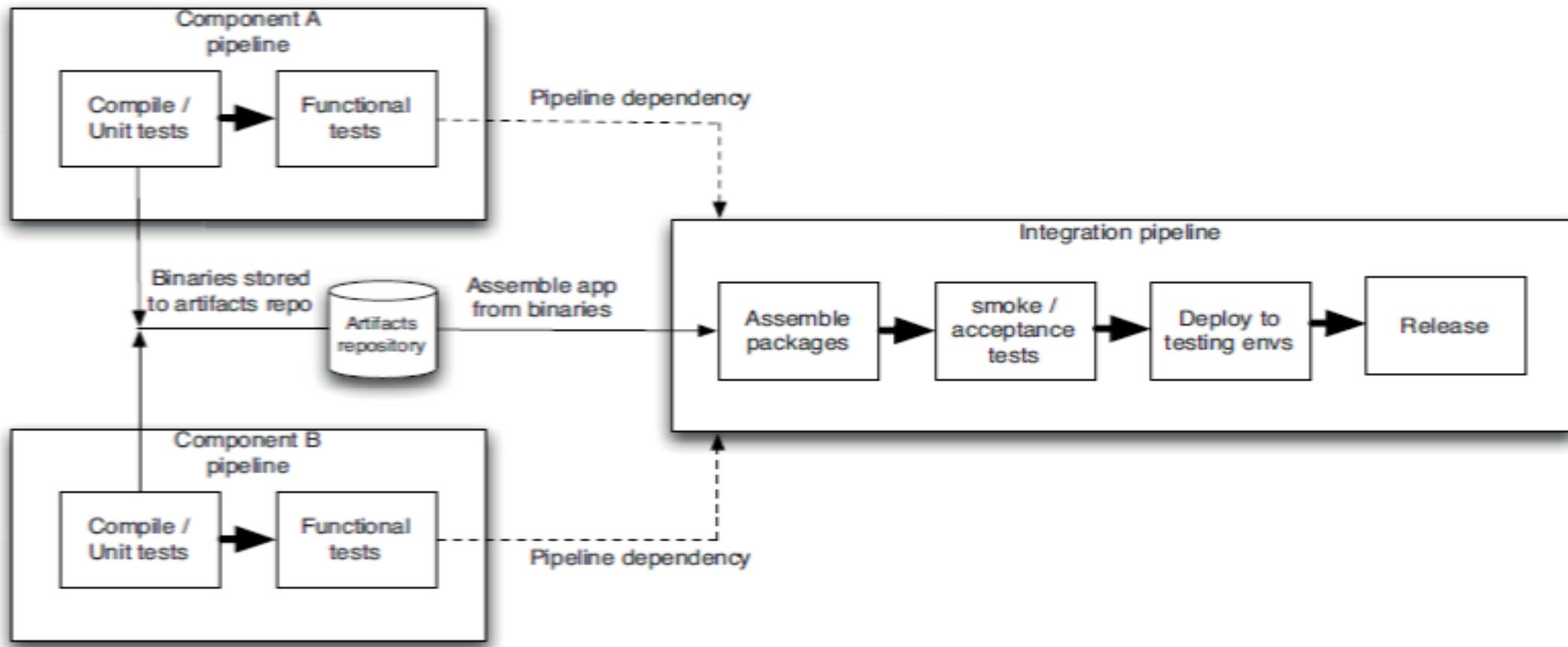
## Pipelining Components

- Split your system into several different pipelines
- The build for each component or set of components should have its own pipeline to prove that it is fit for release
- This pipeline will perform the following steps
  - Compile the code, if necessary
  - Assemble one or more binaries that are capable of deployment to any environment
  - Run unit tests
  - Run acceptance tests
  - Support manual testing, where appropriate



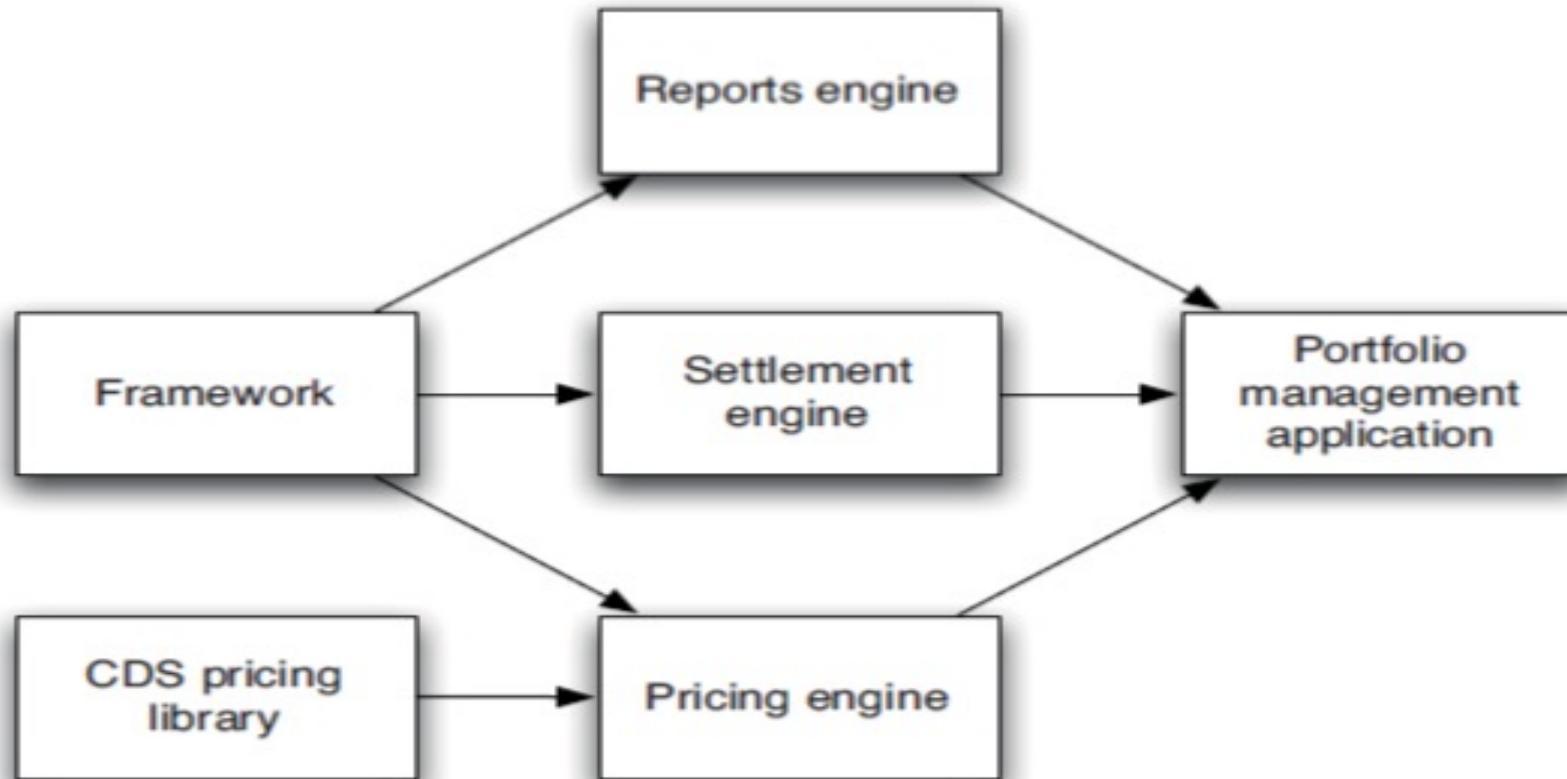
# Managing Component

## Integration Pipeline



# Managing Component

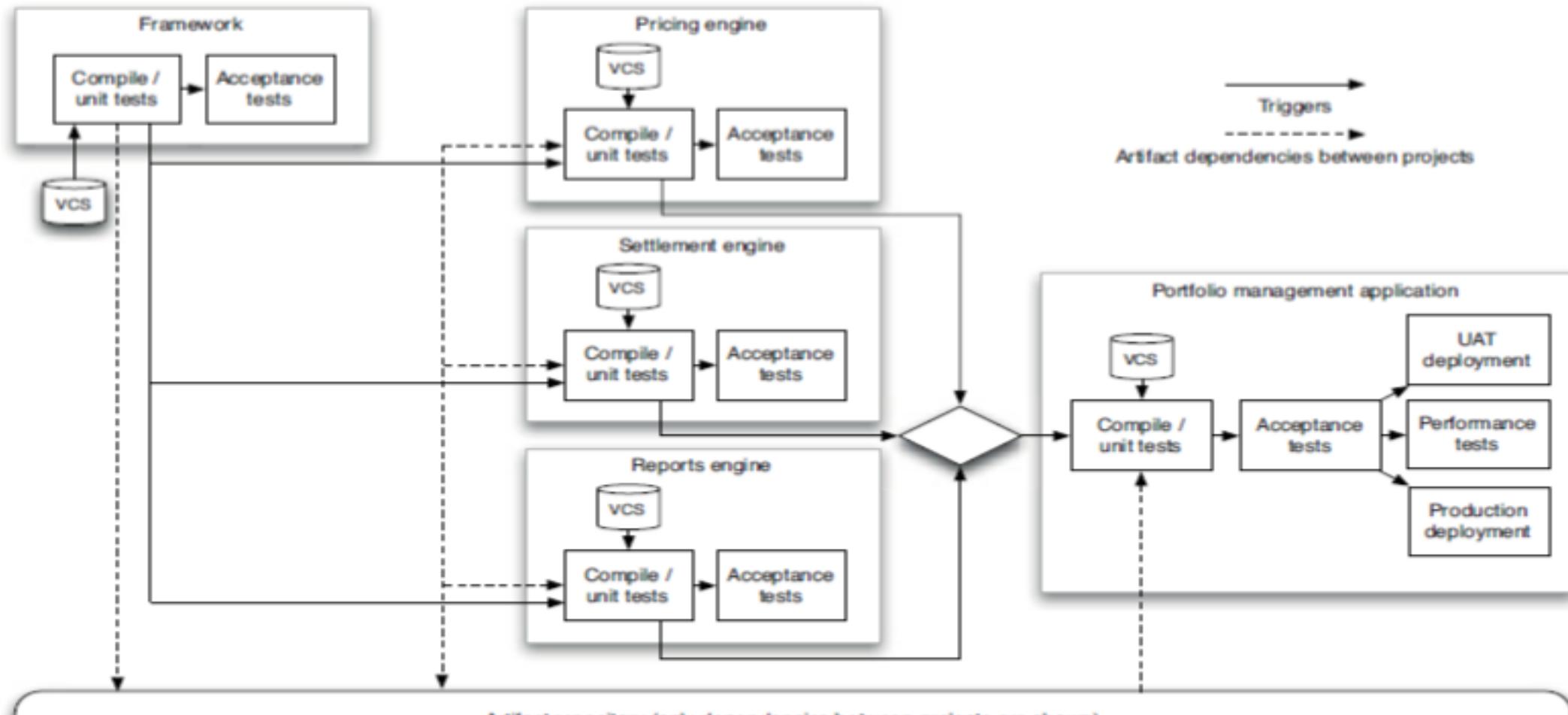
## Managing Dependency Graphs



credit default swap (CDS) library that is provided by third party

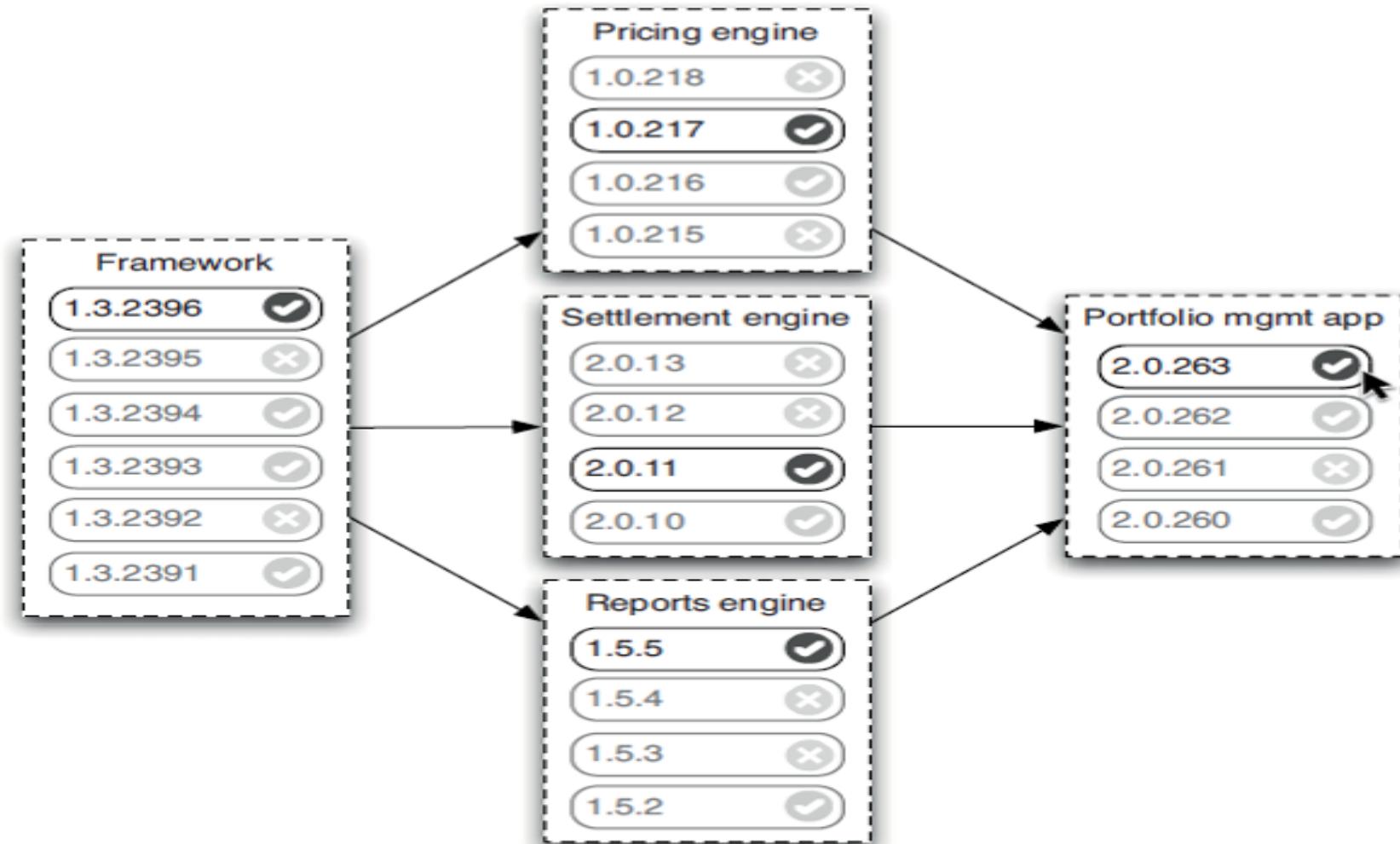
# Managing Component

## Pipelining Dependency Graphs



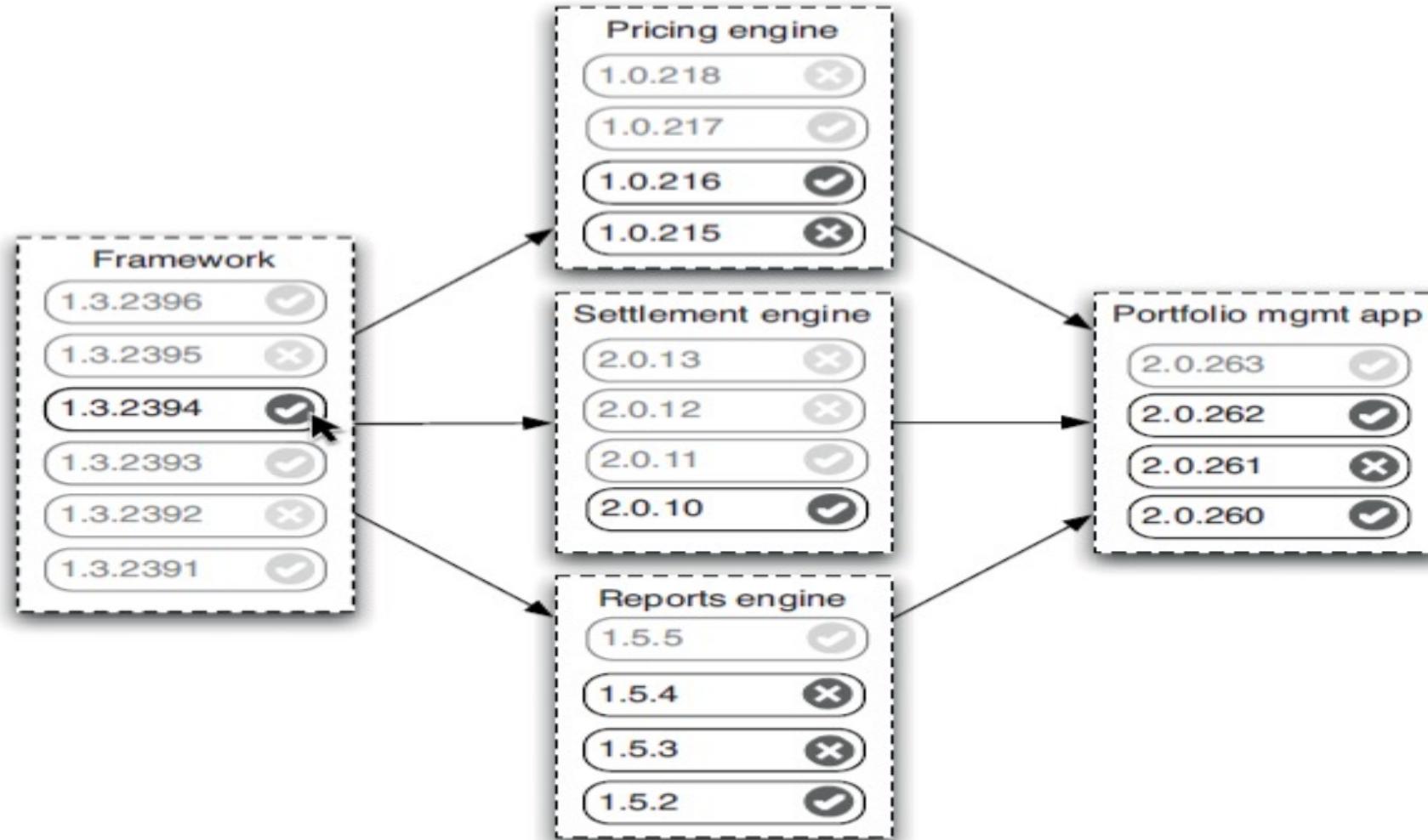
# Managing Component

## Visualizing upstream dependencies



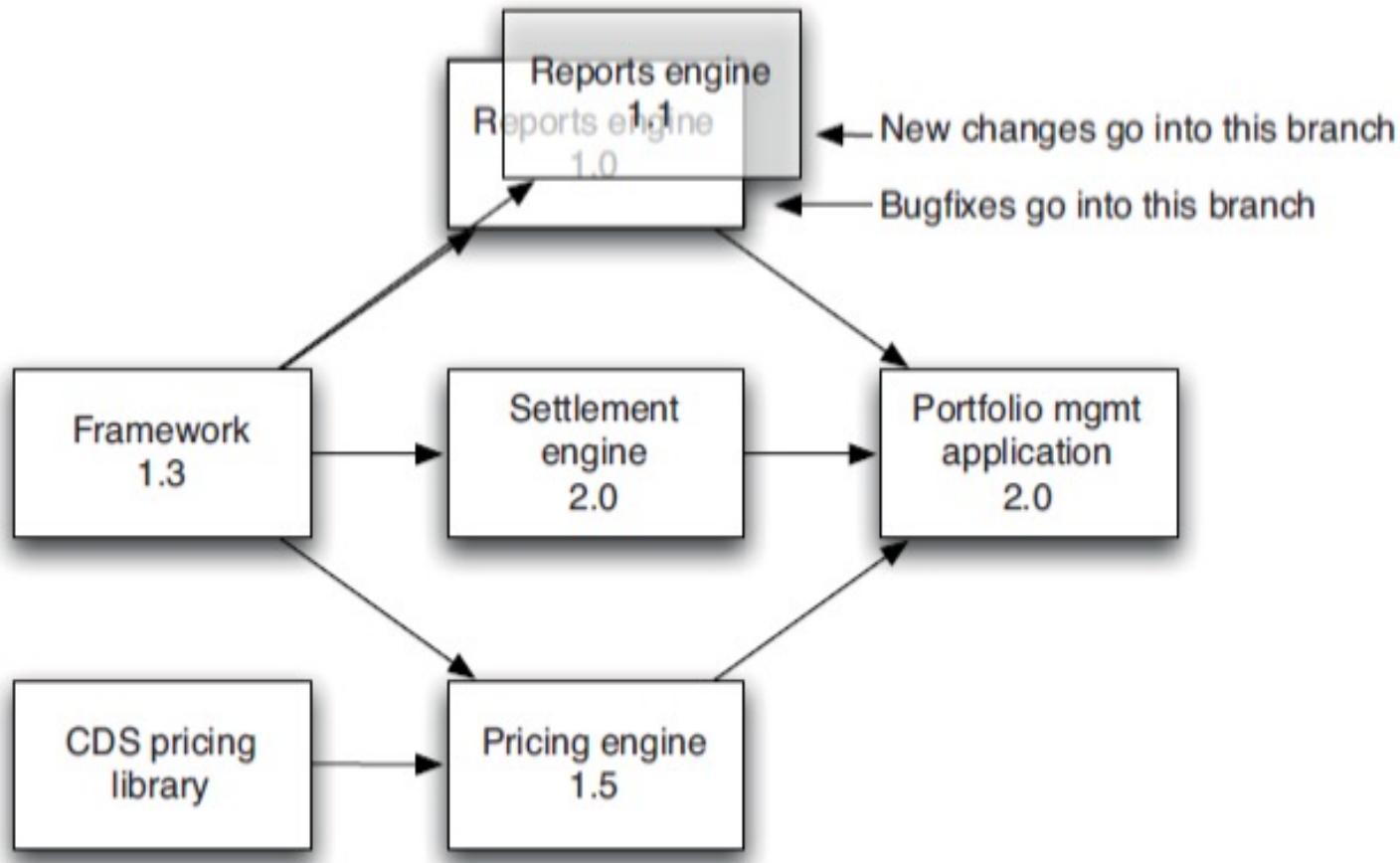
# Managing Component

## Visualizing downstream dependencies



# Managing Component

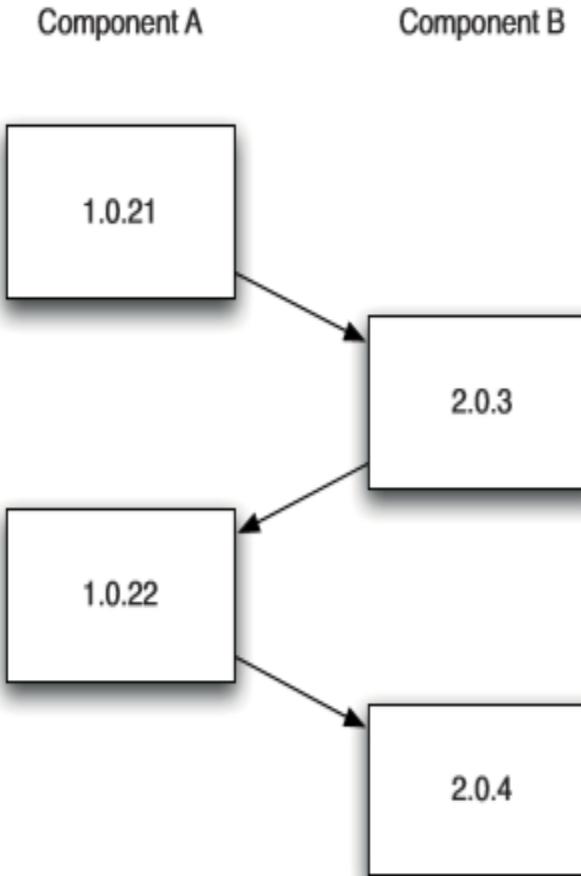
## Branching Components



# Managing Component

## Circular Dependencies

- This occurs when the dependency graph contains cycles
- No build system supports such a configuration out of the box, so you have to hack your toolchain to support it



**Circular dependency build ladder**

# Build automation

## Build tools

- Steps of a build process

Compile the source code

Running & evaluating the unit test

Processing existing resource  
files (configurations)

Generating artifacts (WAR, JAR,...)

# Build automation

## Build tools

- Additional steps that are often executed in the a build process:

Administering dependencies

Analyzing code quality (static  
code analysis)

Running additional tests

Archiving generated artifacts and  
packages in a central repository

# Automating Build Process

## Build Tools

- Maven
- Gradle

| Technology | Tool                       |
|------------|----------------------------|
| Rails      | Rake                       |
| .Net       | MsBuild                    |
| Java       | Ant, Maven, Buildr, Gradle |
| C,C++      | SCons                      |

# Maven

## What is Maven



### Build Tool

- One artifact (Component, JAR, even a ZIP)
- Manage Dependencies

### Management Tool

- Handles Versioning / Releases
- Describe Project
- Produce Javadocs / Site information



Apache Software Foundation  
Maven official site is built with Maven  
Open Source

# Maven

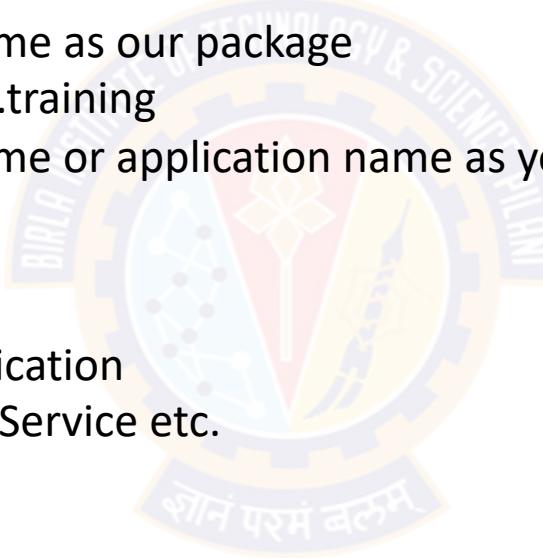
## Maven Structure

- src/main/java : by default maven looks for a src/main/java directory underneath of project)
- target folder : compiles all our source code to target directory
- pom.xml : maven compile source code in way provided by pom.xml
- Different language : src/main/groovy or src/main/resources
- Unit testing: src/test/java
- Target directory : Everything get compile  
Even your test gets run and validated  
Packaged Contents (like JAR, WAR or ZIP depending on what we have provided in pom.xml)

# Maven

## Pom.xml

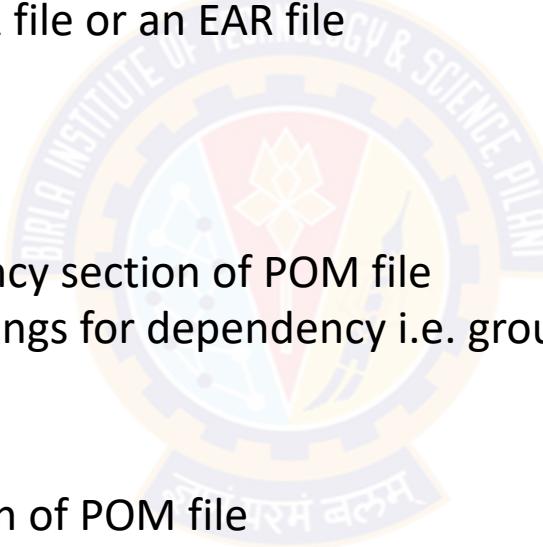
- Maven uniquely identifies a project using
- groupId :
  - The groupId is often the same as our package
  - Ex: com.bits or com.maven.training
  - groupId is like, business name or application name as you would reference it as a web address
- artifactId :
  - Same as name of your application
  - Ex. HelloWorld, OnDemandService etc.
- version :
  - Version of project
  - Format {Major}.{Minor}.{Maintenance} if it is RELEASE
  - '-SNAPSHOT' to identify in development
  - Ex: 1.0-SNAPSHOT



# Maven

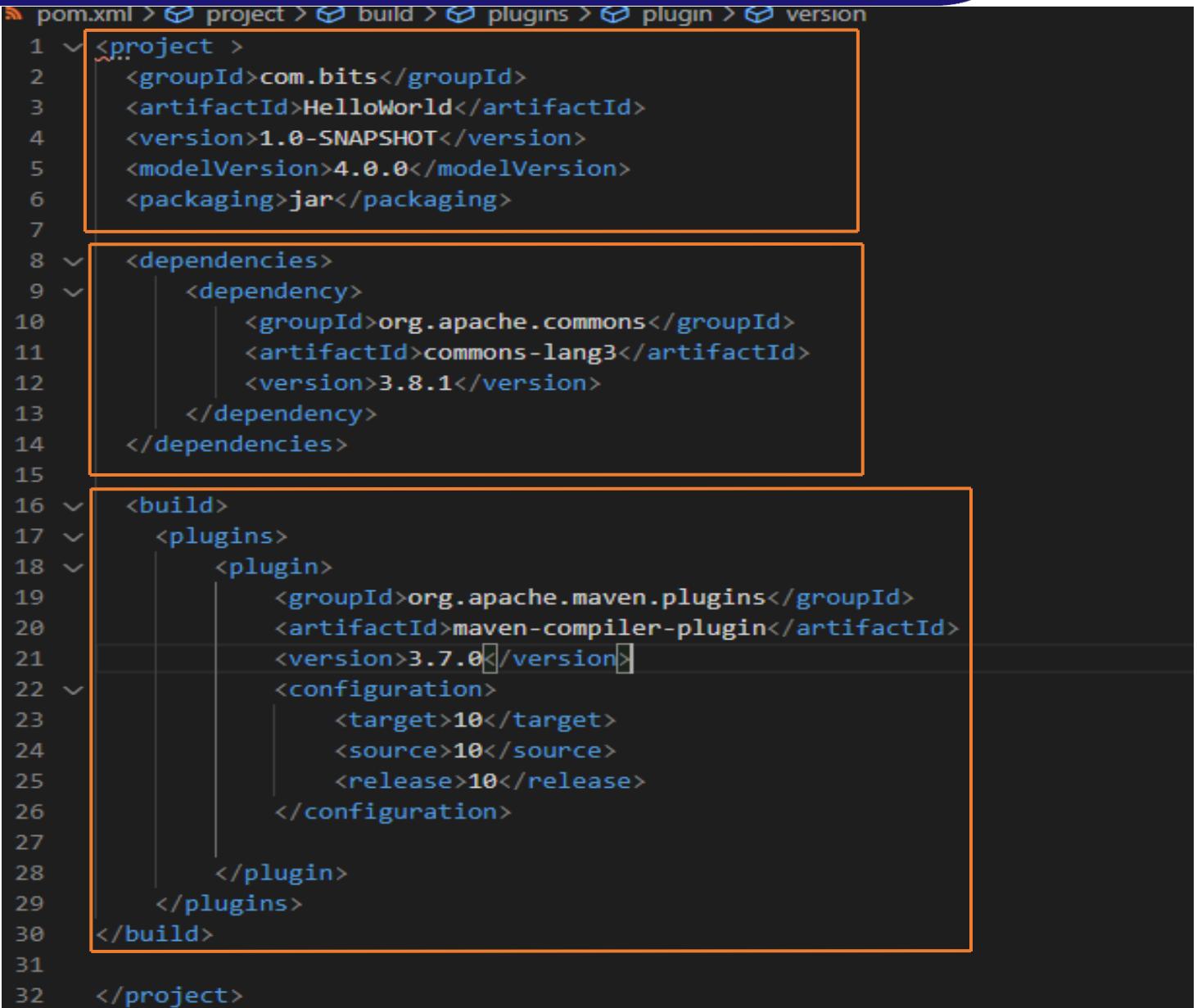
## Pom.xml

- packaging :
  - Packaging is how we want to distribute our application
  - Ex. JAR file, a WAR file, RAR file or an EAR file
  - The default packing is JAR
- dependencies :
  - Just add it to our dependency section of POM file
  - Need to know our three things for dependency i.e. groupId, artifactId, and version
- plugins
  - Just add it to plugins section of POM file



# Maven

## Pom.xml example



```
pom.xml > project > build > plugins > plugin > version
1 <project>
2   <groupId>com.bits</groupId>
3   <artifactId>HelloWorld</artifactId>
4   <version>1.0-SNAPSHOT</version>
5   <modelVersion>4.0.0</modelVersion>
6   <packaging>jar</packaging>
7
8   <dependencies>
9     <dependency>
10       <groupId>org.apache.commons</groupId>
11       <artifactId>commons-lang3</artifactId>
12       <version>3.8.1</version>
13     </dependency>
14   </dependencies>
15
16   <build>
17     <plugins>
18       <plugin>
19         <groupId>org.apache.maven.plugins</groupId>
20         <artifactId>maven-compiler-plugin</artifactId>
21         <version>3.7.0</version>
22         <configuration>
23           <target>10</target>
24           <source>10</source>
25           <release>10</release>
26         </configuration>
27
28       </plugin>
29     </plugins>
30   </build>
31
32 </project>
```

# Maven

## Maven Goals



clean



compile



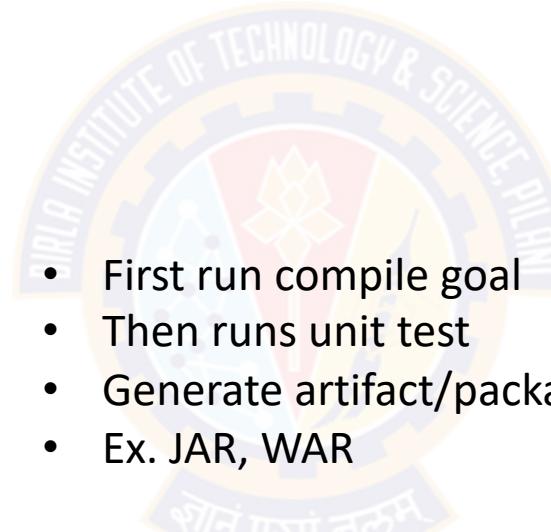
package



install



deploy



- First run compile goal
- Then runs unit test
- Generate artifact/package as per provided in pom.xml
- Ex. JAR, WAR
  
- First run package goal
- Then install the package in local repository
- Default it is .m2 folder
  
- Runs install goal first
- then deploy it to a corporate or remote repository
- Like file sharing

# Maven

## Project Inheritance

- Pom files can inherit configuration
  - groupId, version
  - Project Config
  - Dependencies
  - Plugin configuration
  - Etc.



```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <parent>
    <artifactId>maven-training-parent</artifactId>
    <groupId>org.lds.training</groupId>
    <version>1.0</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>maven-training</artifactId>
  <packaging>jar</packaging>
</project>
```

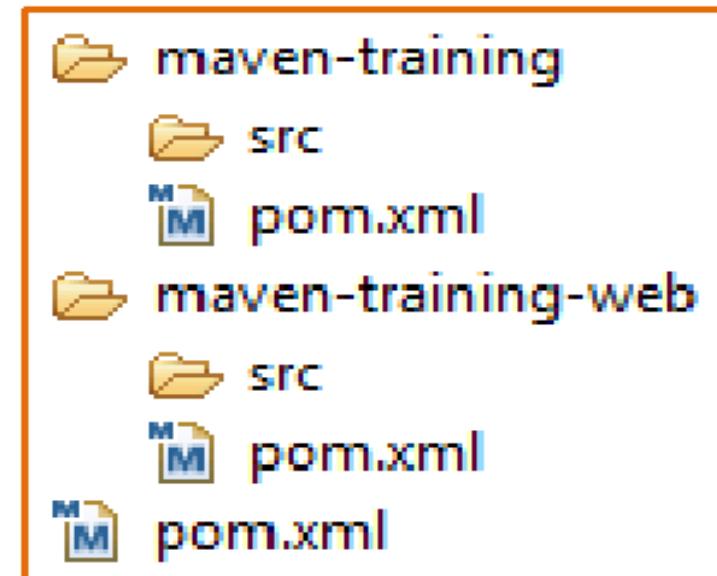
# Maven

## Multi Module Projects

- Maven has 1st class multi-module support
- Each maven project creates 1 primary artifact
- A parent pom is used to group modules



```
<project>
  ...
  <packaging>pom</packaging>
  <modules>
    <module>maven-training</module>
    <module>maven-training-web</module>
  </modules>
</project>
```



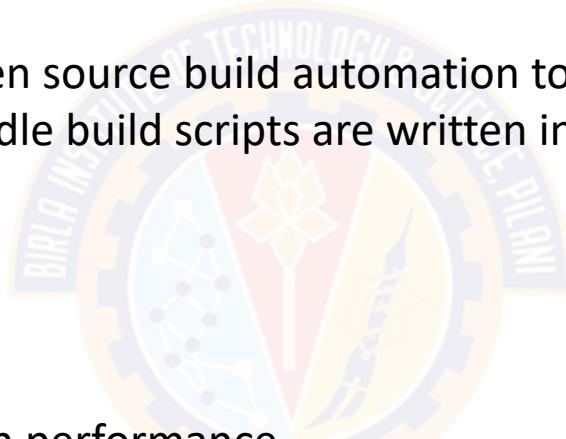
# Gradle

## What is Gradle



### What is Gradle

Open source build automation tool  
Gradle build scripts are written in Domain Specific Language [DSL]



### Why Gradle

#### High performance

- runs only required task; which have been changed
- Build cache helps to reuse tasks outputs from previous run
- ability to have shared build cache within different machine

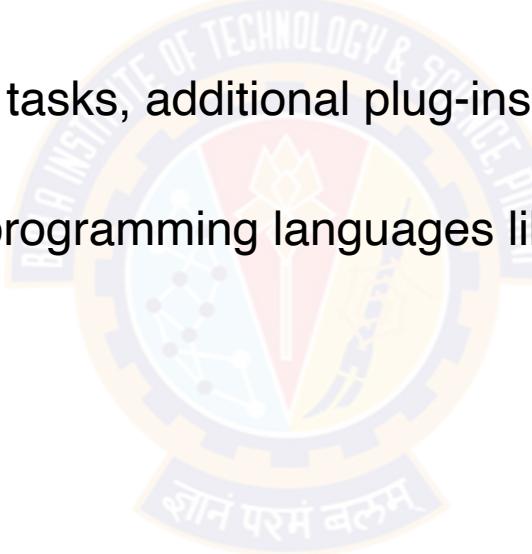
#### JVM foundation

- Java Development Kit is prerequisite
- It is not limited to Java

# Gradle

## Core concepts

- Tasks and the dependencies between them
- Gradle calculates a directed, acyclic graph to determine which tasks have to be executed in which order
- Graph can change through custom tasks, additional plug-ins, or the modification of existing dependencies
- Plug-ins allows to work with other programming languages like Groovy, kotlin C++ etc.,



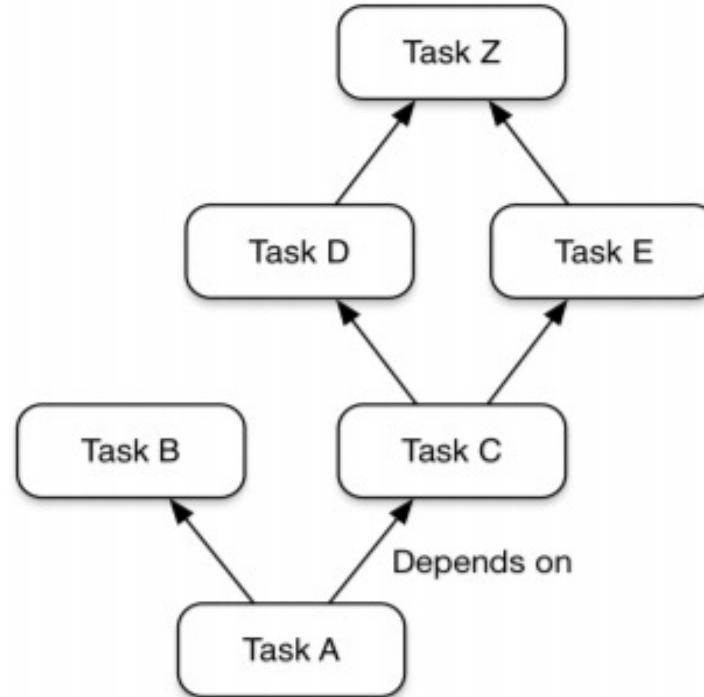
# Gradle

## Core model

- The core model is based on tasks
  - Directed Acyclic Graphs (DAGs) of tasks
- Example of a Graph
- Tasks Consists of:
  - Action : To perform something
  - Input : values to action
  - Output : generated by



Generic task graph



# Build Tool

## Summary



### Gradle



#### Flexibility:

- Flexibility on Conventions
- User Friendly & Customized

#### Flexibility:

- No flexibility on Conventions
- It is rigid

#### Performance:

- It process only the files has been changed
- Reusability by working with Build Cache
- Shipping is faster

#### Performance:

- It process the complete build
- No Build Cache concept
- Shipping is slow as compared to Gradle

#### User Experience:

- IDE Support : Is in evolving Stage
- CLI : Modern CLI

#### User Experience:

- IDE Support : Is mature
- CLI solution is classic in comparison with Gradle

# References

## CS 4,5,6 & 7

- Chapter 5 from Effective DevOps Building a Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis and Katherine Daniels
- Transformation to Enterprise DevOps culture: <https://devops.com/six-step-approach-enterprise-devops-transformation/>
- **Cloud as a Catalyst:**
- TextBook 2: Contineous Delivery : Chapter 11 Managing Infrastructure and Environments
- TextBook 1: DevOps A S/W Architect Pres: Chapter 2 the Cloud as a Platform
- Chapter 3, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
- Chapter 5, Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis
- <https://git-scm.com/>



# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Automating Build Process

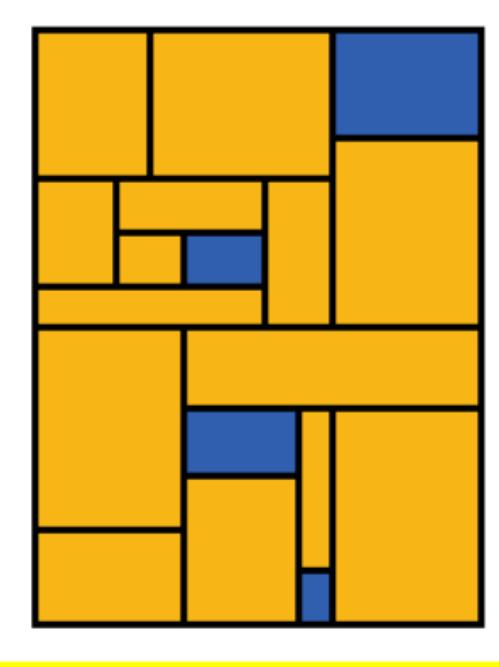
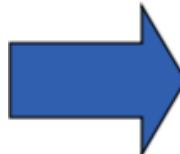
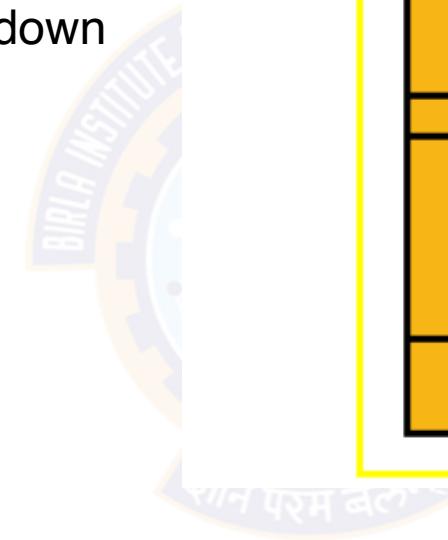
- Unit testing
- Automates Test Suite - Selenium
- Continuous Code Inspection
- Code Inspection Tools
  - Sonarqube



# Unit Testing

## Traditional Testing

- Test the system as a whole
- Errors go undetected
- Isolation of errors difficult to track down



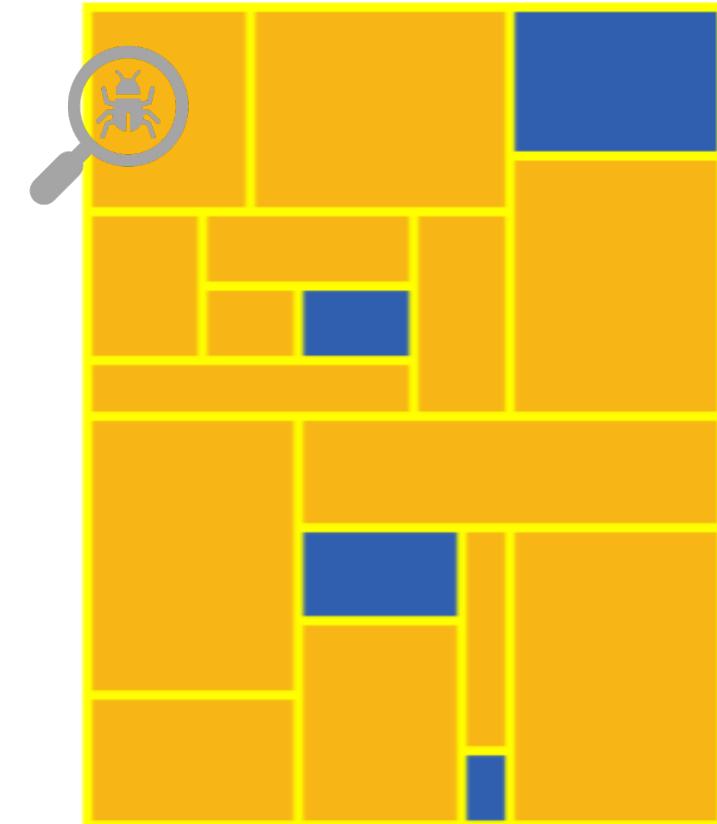
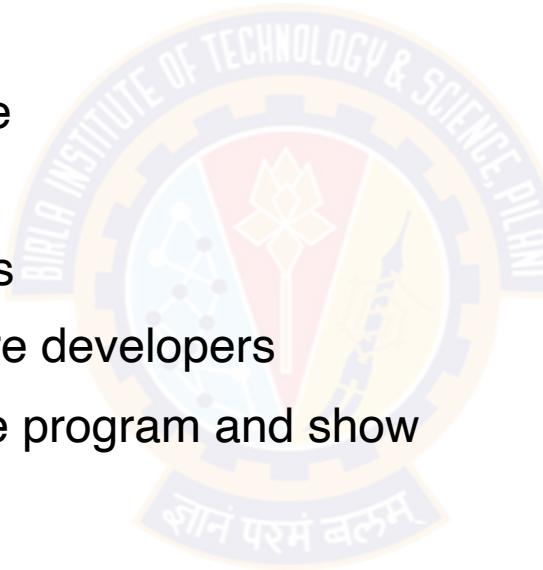
### Traditional Testing Strategies

- Print Statements
- Use of Debugger
- Debugger Expressions
- Test Scripts

# Unit Testing

## What is Unit Testing

- Is a level of the software testing process where individual units/components of a software/system are tested
- Each part tested individually
- All components tested at least once
- Errors picked up earlier
- Scope is smaller, easier to fix errors
- Typically written and run by software developers
- Its goal is to isolate each part of the program and show that the individual parts are correct



# Unit Testing

## Why Unit Testing

### Concerned with

- Functional correctness and completeness
- Error handling
- Checking input values (parameter)
- Correctness of output data (return values)
- Optimizing algorithm and performance

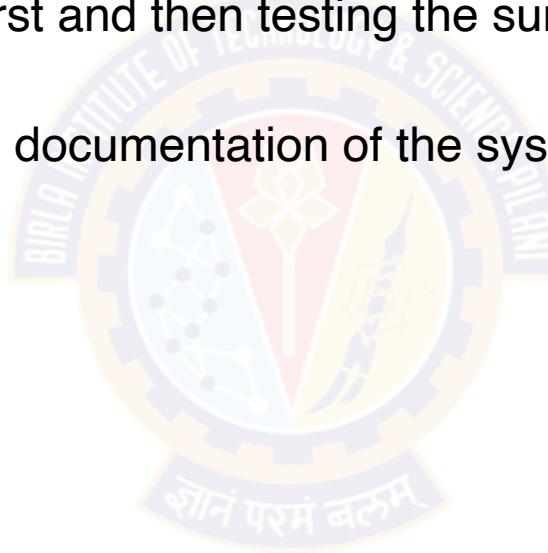


- Faster Debugging
- Faster Development
- Better Design
- Excellent Regression Tool
- Reduce Future Cost

# Unit Testing

## Benefits

- Unit testing allows the programmer to refactor code earlier and make sure the module works correctly
- By testing the parts of a program first and then testing the sum of its parts, i.e. integration testing becomes much easier
- Unit testing provides a sort of living documentation of the system



# Unit Testing

## Guidelines

- Keep unit tests small and fast
- Unit tests should be fully automated and non-interactive
- Make unit tests simple to run
- Measure the tests
- Fix failing tests immediately
- Keep testing at unit level
- Keep tests independent
- Name tests properly
- Prioritize testing



# Test Automation

## Selenium

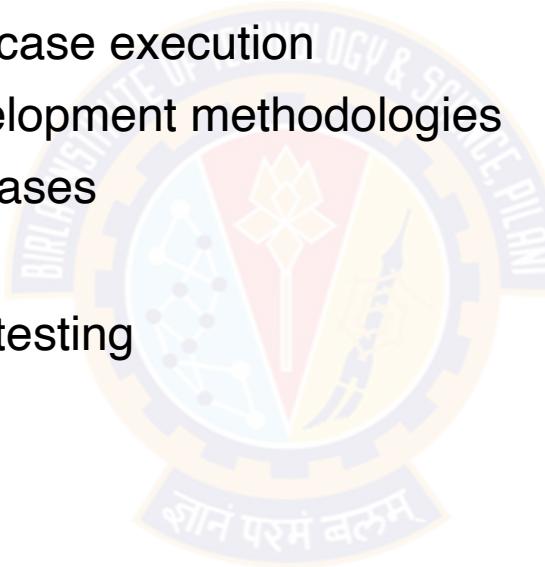
- Perform an sort of interaction
- Selenium helps to automate web browser interaction
- Scripts perform the interactions



# Selenium

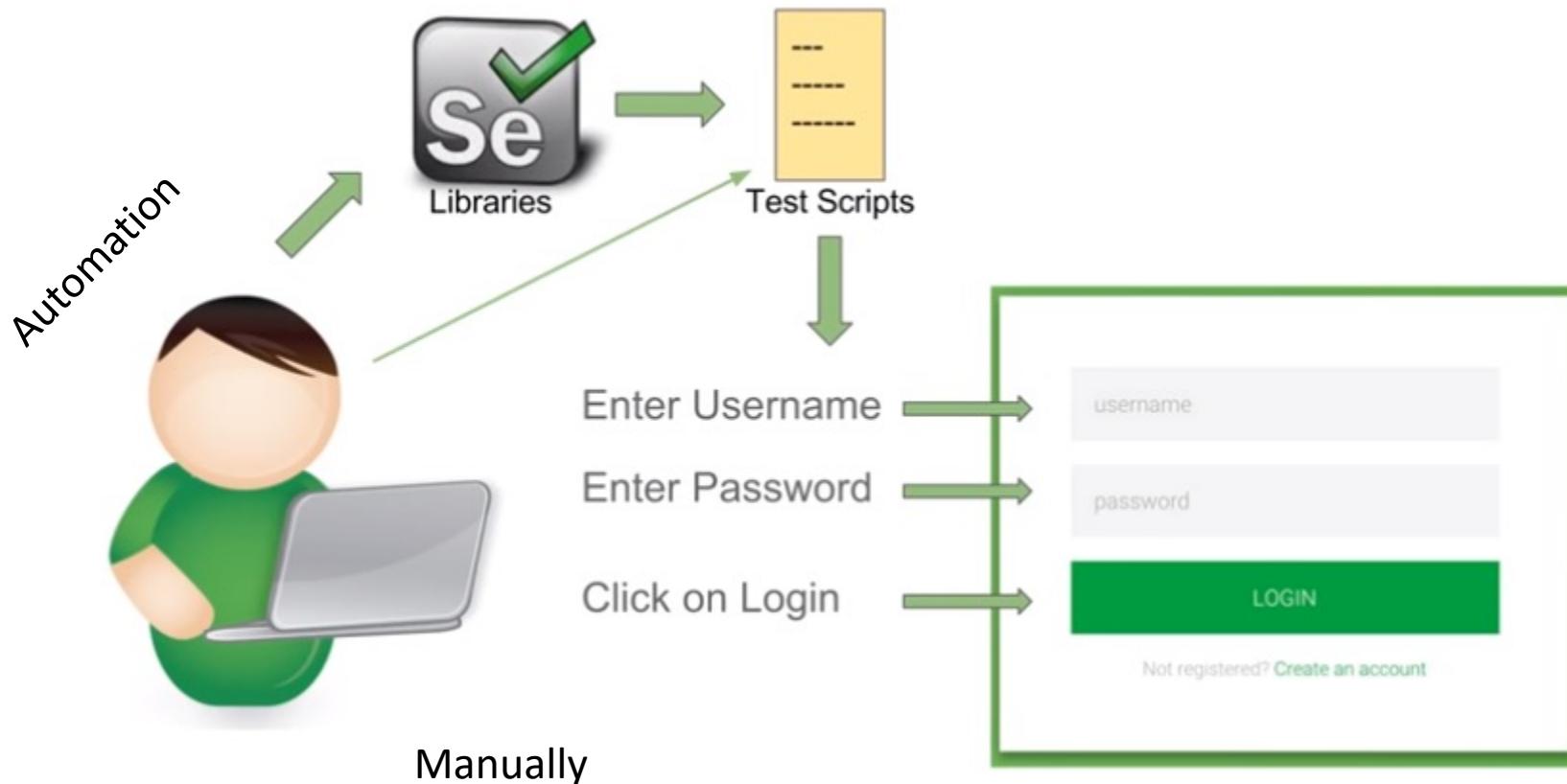
## Benefits

- Frequent regression testing
- Rapid feedback to developers
- Virtually unlimited iterations of test case execution
- Support for Agile and extreme development methodologies
- Disciplined documentation of test cases
- Customized defect reporting
- Finding defects missed by manual testing
- Reduced Business Expenses
- Reusability of Automated Tests
- Faster Time-to-Market



# Selenium

Lets say you want to test one login page



# Selenium

## Example

- At a high level you will be doing three things with Selenium

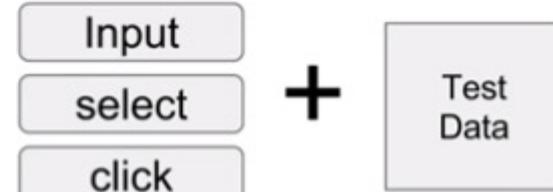
1

Identify web elements  
(using identifiers like id,xpath)

A screenshot of a 'Customer' data entry form. It contains fields for Name, Email, Investment, Date Joined, and Active status. There are 'OK' and 'Cancel' buttons at the bottom.

2

Add Actions  
(using your preferred programming language)



3

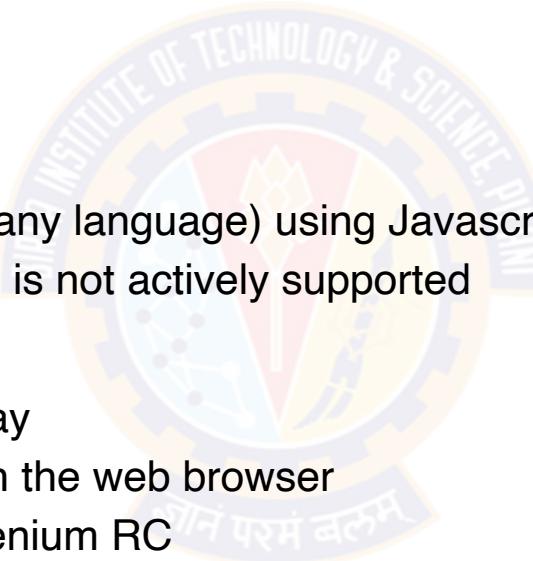
Run the test



# Selenium

## Components

- Selenium IDE
  - A Record and playback plugin for Firefox add-on
  - Prototype testing
- Selenium RC (Remote Control)
  - Also known as selenium 1
  - Used to execute scripts (written in any language) using Javascript
  - Now Selenium 1 is deprecated and is not actively supported
- WebDriver
  - Most actively used component today
  - An API used to interact directly with the web browser
  - Is a successor to Selenium 1 / Selenium RC
  - Selenium RC and WebDriver are merged to form Selenium 2
- Selenium Grid
  - A tool to run tests in parallel across different machines and different browser simultaneously
  - Used to minimize the execution time



# Selenium

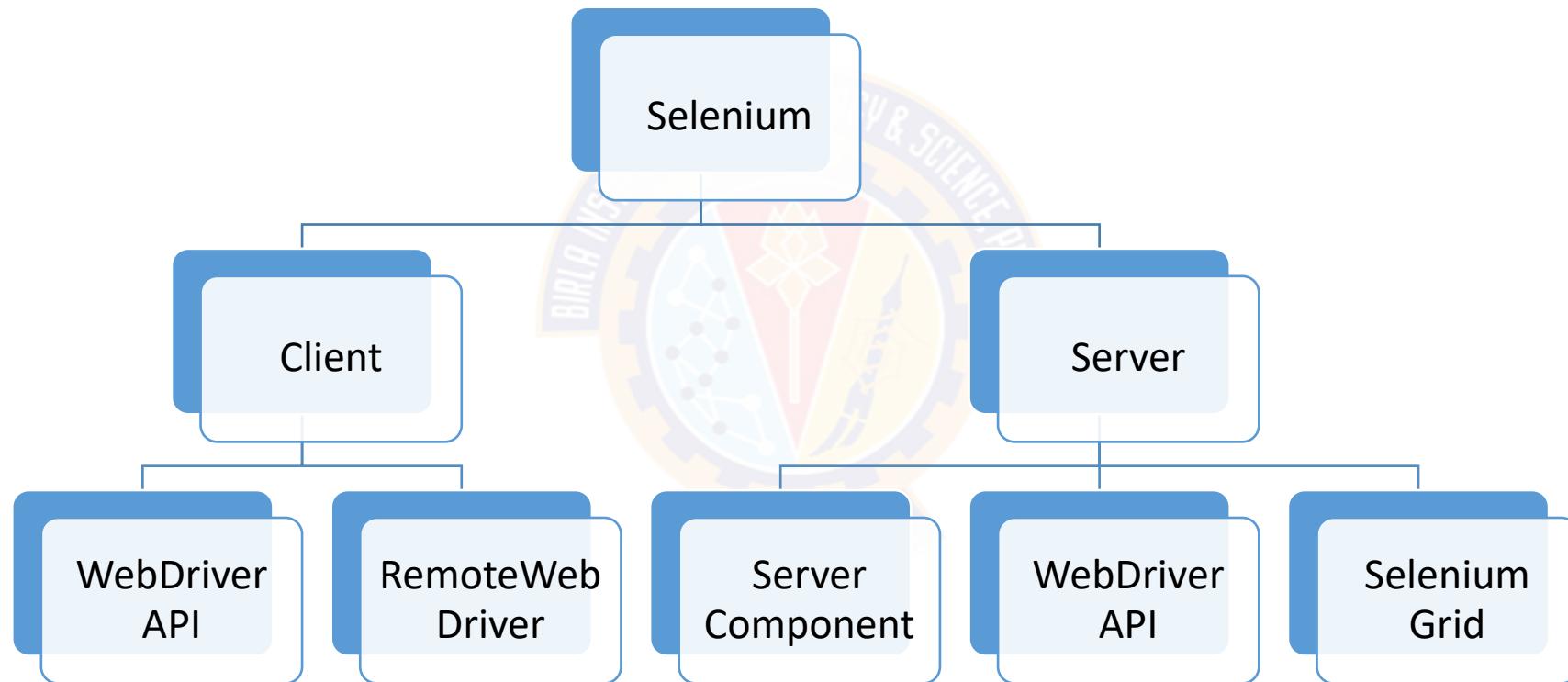
## Supports

- Browsers
- OS
- Language



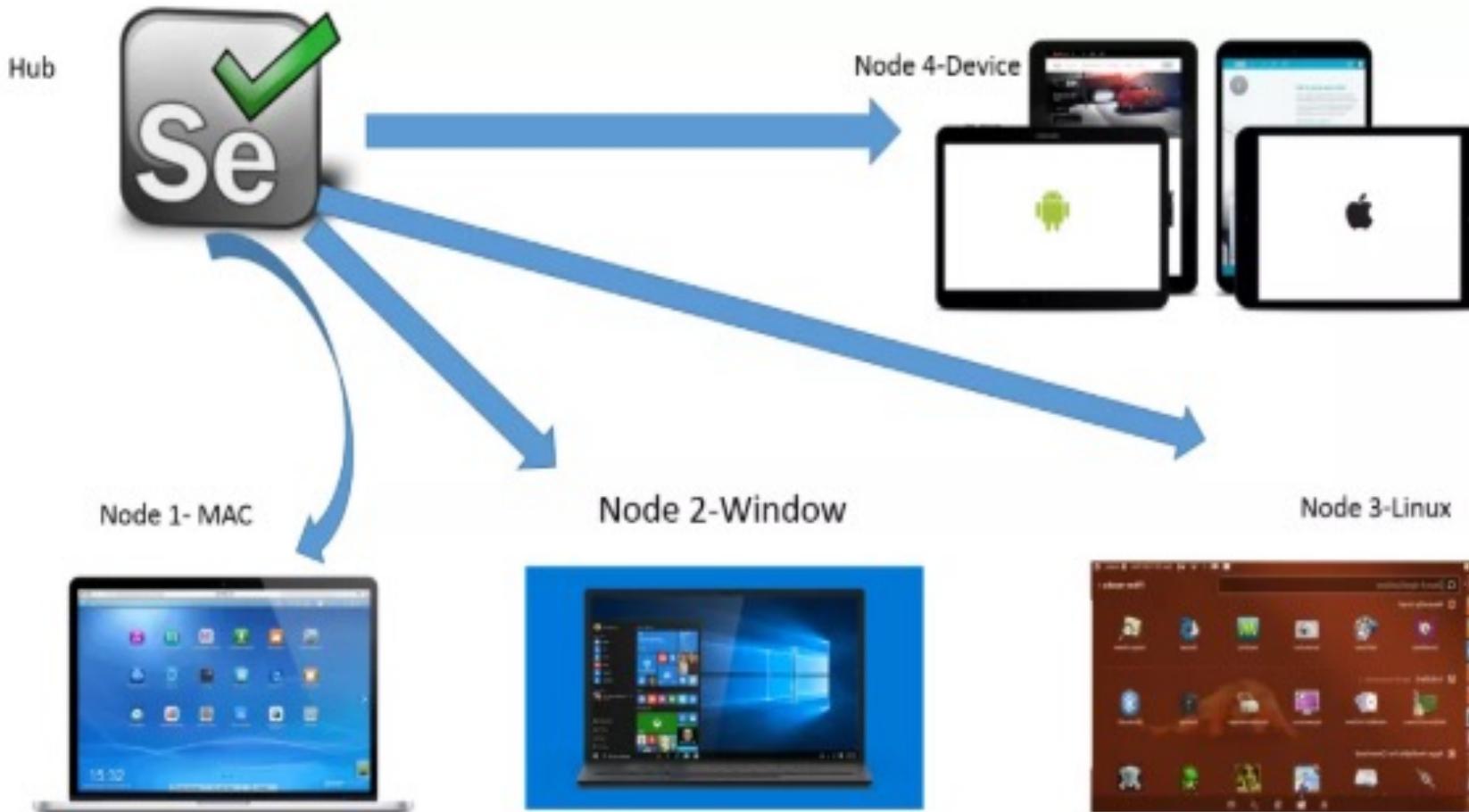
# Selenium

## Architecture



# Selenium

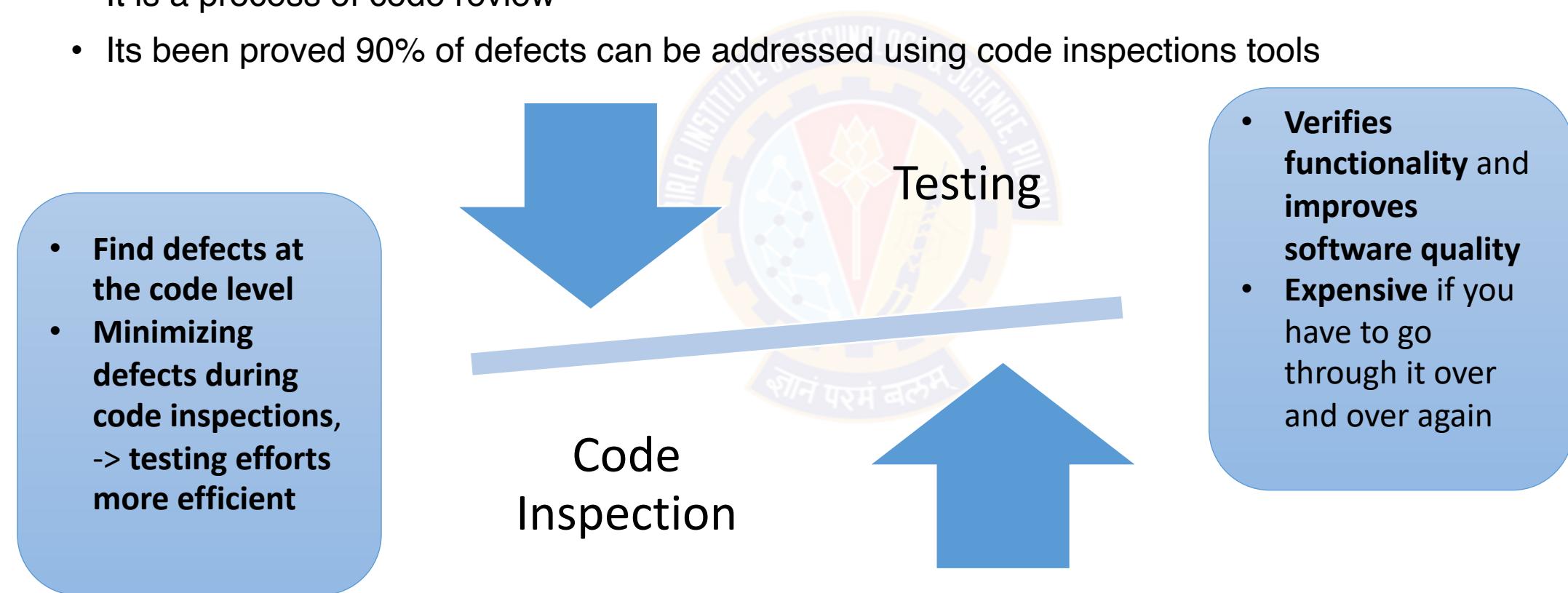
## Selenium Grid



# Continuous Code Inspection

## Continuous code inspection = Constantly scanning code

- Identify if any defects
- It is a process of code review
- It's been proved 90% of defects can be addressed using code inspection tools



*Note: Even with automated testing, it takes time to verify functionality; by resolving defects at the code level, you'll be able to test functionality faster*

# Continuous Code Inspection

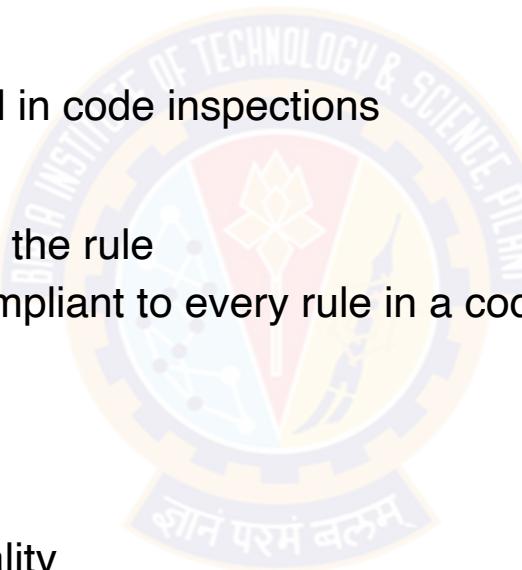
## Code Inspection Measures

- Code inspections must be well-defined as per requirements:
  - Functional requirements : User Needs : Cosmetic
  - Structural requirements : System Needs : Re-engineering
- Run Time Defects:
  - Identify run time errors before program run
  - Examples: Initialization (using the value of unset data), Arithmetic Operations (operations on signed data resulting in overflow) & Array and pointers (array out of bounds, dereferencing NULL pointers), etc.,
- Preventative Practices:
  - This help you avoid error-prone or confusing code
  - Example: Declarations (function default arguments, access protection), Code Structure (analysis of switch statements) & Safe Typing (warnings on type casting, assignments, operations), etc.,
- Style:
  - In-house coding standards are often just style, layout, or naming rules and guidelines
  - Instead using a proven coding standard is better for improving quality

# Continuous Code Inspection

## Improve Your Code Inspection Process:

- Involve Stakeholders
  - Developer, Management & Customer
- Collaborate
  - Collaboration — both in coding and in code inspections
- Recognize Exceptions
  - Sometimes there are exceptions to the rule
  - In an ideal world, code is 100% compliant to every rule in a coding standard
  - The reality is different
- Document Traceability
  - Traceability is important for audits
  - Capture the history of software quality
- **What to Look For in Code Inspection Tools**
  - Automated inspection
  - Collaboration system



# Continuous Code Inspection Tool

## SonarQube



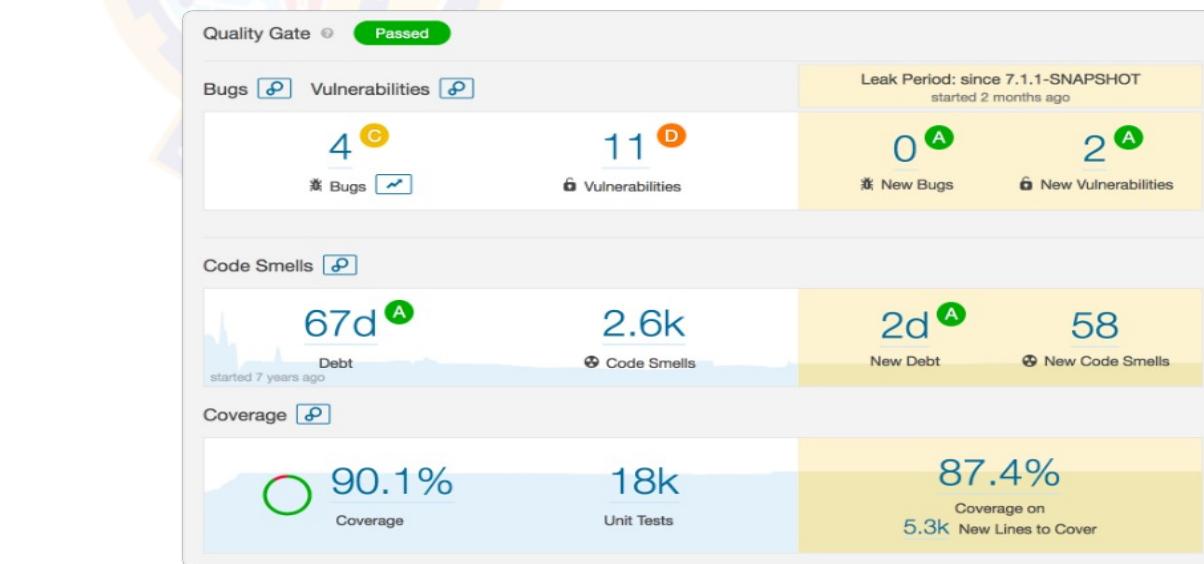
capability to show health of an application



Highlight issues newly introduced



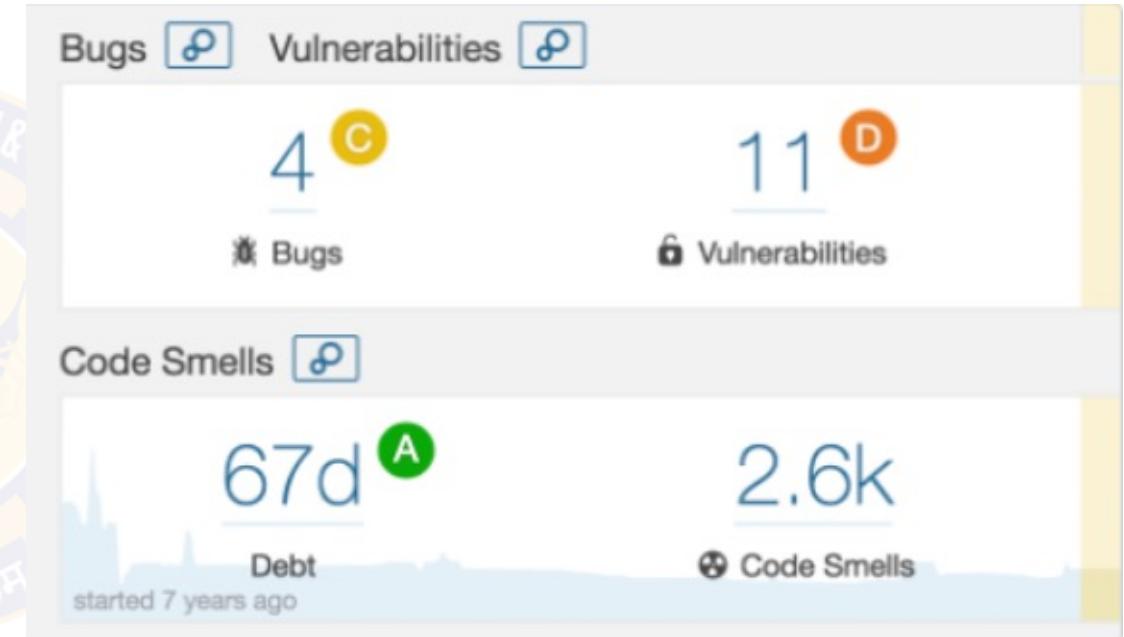
Quality Gate, you can fix the leak and therefore improve code quality systematically



# SonarQube

## Overall health

- Bug:
  - An issue that represents something wrong in the code
  - If this has not broken yet, it will, and probably at the worst possible moment
- Code Smell:
  - A maintainability-related issue in the code
  - Examples: Dead Code, Duplicate code, Comments, Long method, Long parameter list, Long class etc.,
- Vulnerability:
  - A security-related issue which represents a backdoor for attackers



# SonarQube

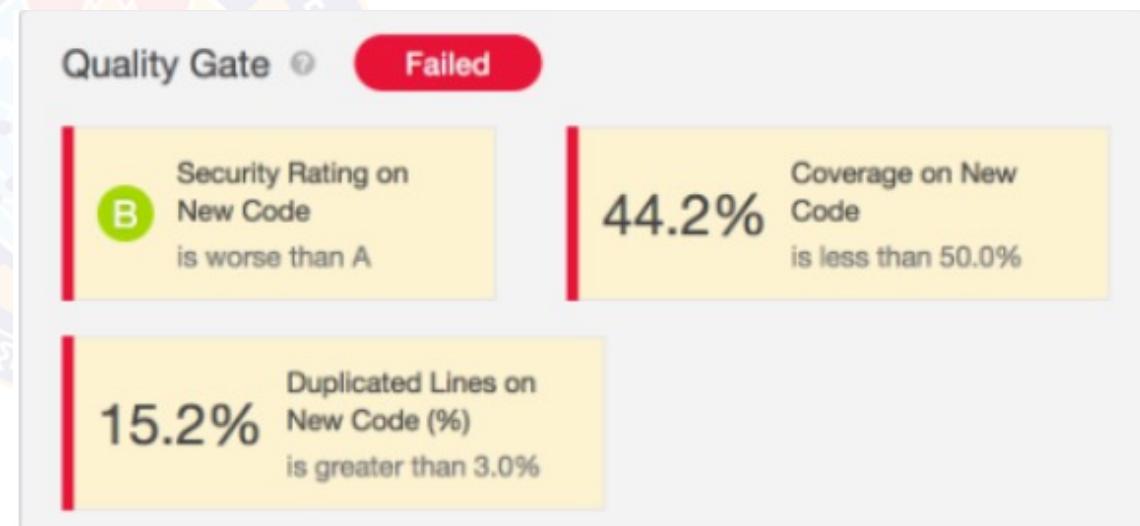
## Enforce Quality Gate

- To fully enforce a code quality practice across all teams, need a Quality Gate
- A set of requirements that tells whether or not a new version of a project can go into production
- SonarQube's default Quality Gate checks what happened on the Leak period and fails if your new code got worse in this period

A quality gate is the best way to enforce a quality policy in your organization

Define a set of Boolean conditions based on measure thresholds against which projects are measured

It supports multiple quality gate definitions

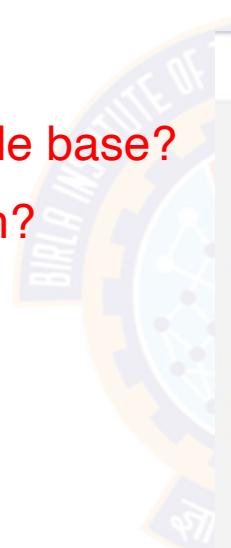


Example: Failed Project

# SonarQube

## Dig into issues

- The “Issues” page of your project gives you full power to analyze in detail
- What the main issues are?
- Where they are located?
- When they were added to your code base?
- And who originally introduced them?

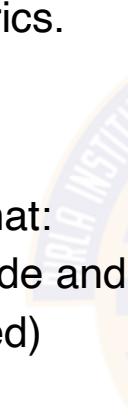


The screenshot shows the SonarQube 'Issues' page for a project. The page has a navigation bar with tabs: Overview, Issues (selected), Measures, Code, and Activity. Below the tabs, there are buttons for 'My Issues' and 'All'. A 'Bulk Change' button is also present. The main area is titled 'Filters' with a 'Clear All Filters' button. Under 'Display Mode', 'Issues' is selected. The 'Type' filter shows 4 Bugs, 5 Vulnerabilities, and 702 Code Smells. The 'Severity' filter shows 0 Blockers, 103 Critical issues, 737 Info issues, and 702 Major issues. On the right, a list of issues is shown with their details and status. The first issue is 'Make the "LOG" logger private static final.' It is a Code Smell, Major severity, Open status, and assigned to Sébastien Lasaïn with 5 votes. The second issue is 'Catch Exception instead of Throwble.' It is a Code Smell, Major severity, Open status, and assigned to Simon Brandhof with 20 votes. The third issue is '1 duplicated blocks of code must be removed.' It is a Code Smell, Major severity, Open status, and assigned to tomverin with 20min effort. The fourth issue is 'Annotate the parameter with @javax.annotation.Nullable in mx'. It is a Code Smell, Major severity, Open status, and assigned to tomverin with 20min effort.

# SonarQube

## Analyzing Source Code

- Analyze pull requests
  - Focuses on new code – The Pull Request quality gate only uses your project's quality gate conditions that apply to "on New Code" metrics.
- Branch Analysis
  - Each branch has a quality gate that:
  - Applies on conditions on New Code and overall code
  - Assigns a status (Passed or Failed)



| Branch  | Status      | Issues | Details              |
|---|-------------|--------|----------------------|
| master  | Main Branch | Passed |                      |
| dm/refactor_system_info_ws                    | Passed      | 1      | <a href="#">View</a> |
| feature/MMF-1066/apply_feedback               | Passed      | 0      | <a href="#">View</a> |
| feature/MMF-1066/make_updatecenter_a_ma...    | Passed      | 0      | <a href="#">View</a> |
| feature/daniel/SONAR-10008/issue_search_da... | Passed      | 0      | <a href="#">View</a> |
| feature/daniel/SONAR-7992/version_names_1...  | Passed      | 0      | <a href="#">View</a> |
| feature/eh/SONAR-10018                        | Passed      | 0      | <a href="#">View</a> |

# SonarQube

## Integration for DevOps

**maven**



**Gradle**

Makefile

MSBuild



 **Bamboo**

 **Travis CI**

 **Jenkins**

 **AppVeyor**

 **Azure DevOps**

 **TeamCity**



# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Continuous Integration

- Continuous Integration
- Prerequisites for Continuous Integration Version Control
- Continuous Integration Practices

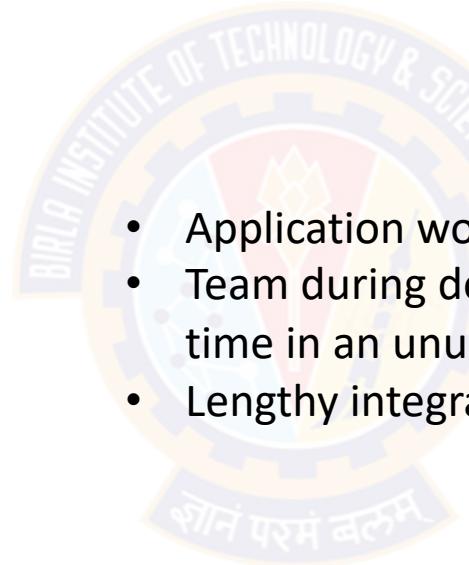


# Continuous Integration

## Continuous integration (CI)

- Process of integrating new code written by developers with a mainline or “master” branch frequently throughout the day

***“Nobody is interested in trying to run the whole application until it is finished”***



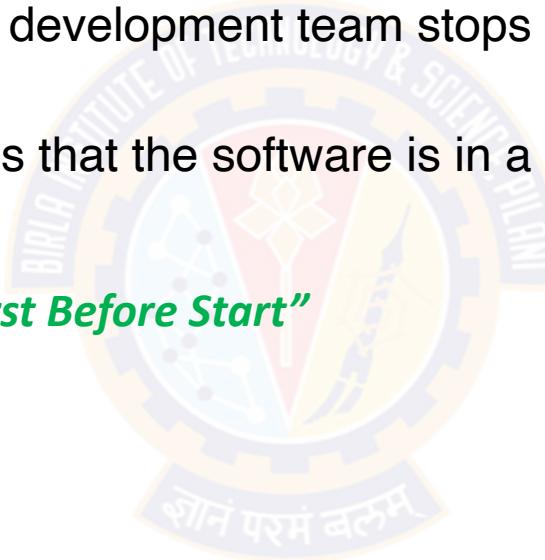
- Application wont be in a working state
- Team during development spends significant proportion time in an unusable state
- Lengthy integration phases at the end of development

# Continuous Integration

## Continuous integration requires

- Every time somebody commits any change, the entire application is built and a comprehensive set of automated tests is run against it
- If the build or test process fails, the development team stops whatever they are doing and fixes the problem immediately
- The goal of continuous integration is that the software is in a working state all the time

*In simple words we can say “Finish First Before Start”*



# Implementing Continuous Integration

## Pre-requisites

1. Version Control



2. An Automated Build

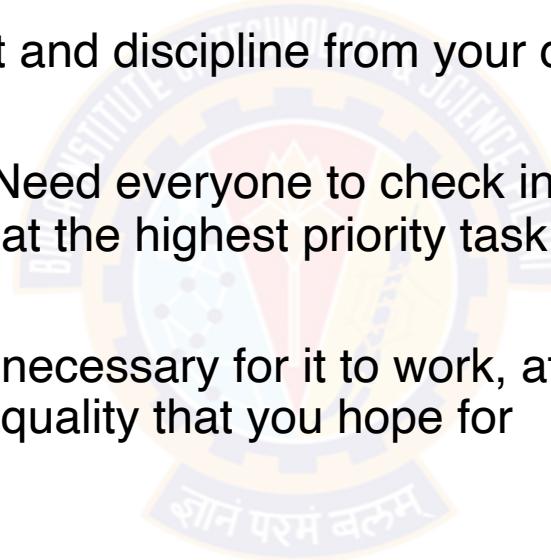


3. Agreement of the Team

# Continuous Integration Pre-requisite

## Agreement of the Team

- This is more about People & Culture
- Continuous integration is a practice, not a tool
- It requires a degree of commitment and discipline from your development team or people involved
- As said “Fix first before Proceed”: Need everyone to check in small incremental changes frequently to mainline and agree that the highest priority task on the project is to fix any change that breaks the application
- If people don’t adopt the discipline necessary for it to work, attempts at continuous integration will not lead to the improvement in quality that you hope for



# Continuous Integration

## CI Tools

- Open Source :
  - Jenkins
  - Cruise Control
  - GitLab CI
  - GitHub Actions
- Commercial:
  - ThoughtWorks Studios
  - TeamCity by JetBrains
  - Bamboo by Atlassians
  - BuildForge by IBM



# How it was before Continuous Integration

## Just a glance !!!

- Nightly Build 



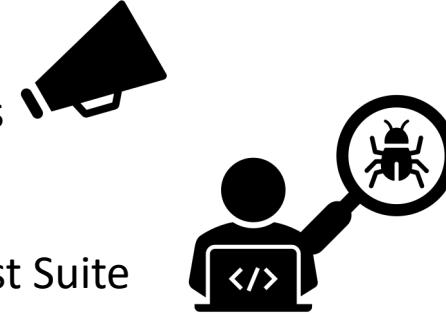
*Note: this strategy will not be a good Idea when you have a geographically dispersed team working on a common codebase from different time zones*

# Continuous Integration

## Pre-requisite & Best Practices

Prerequisite

Check In Regularly -> frequent check-ins



Create a Comprehensive Automated Test Suite

Unit Test

Component Test

Acceptance Test

*Will provide extremely high level of confidence that any introduced change has not broken existing functionality*

Keep the Build and Test Process Short

*Standard Recommendation:*

*10 min is Good*

*5 min is Better*

*90 Sec is IDEAL*

Managing Your Development Workspace

local Development Workspace must be replica of Production

# Continuous Integration

## Pre-requisite & Best Practices

### Best Practices

#### Don't Check In on a Broken Build

What if we do Check In on Broken Build:

If any new check-in or build trigger during broken state will take much longer time to fix

Frequent broken build will encourage team not to care much about working condition

#### Commit locally than direct to Production

#### Wait for Commit Tests to Pass before Moving On

At time of Check-in, you should monitor the build progress  
Never go home with broken build

#### Always Be Prepared to Revert to the Previous Revision

The previous revision was good because; you don't check in on a broken build

# Continuous Integration

## Scenario 1

Friday at 5.30 PM; your build is fail.

- You will be leaving late, and try to fix it
- You can revert changes
- You can leave the build broken

What Option you will opt here?



### Leaving Broken Build:

- On Monday your memory will no longer be fresh
- It will take you significantly longer to understand the problem and fix it
- Everyone at work will yell at you
- If you are late on Monday; be ready to answer n number of calls
- Not the least your name will be mud



Stay late to fix the build after working hours

Check in early enough so you have helping hands around  
If it is late to Check In then Save your Check in for Next Day

Make rule of not checking in less than an hour before the end of work

**Best Solution if you are alone then:**

Revert it from your Source Control

# Continuous Integration

## Scenario 2

If you try to revert every time then; how can you make progress?

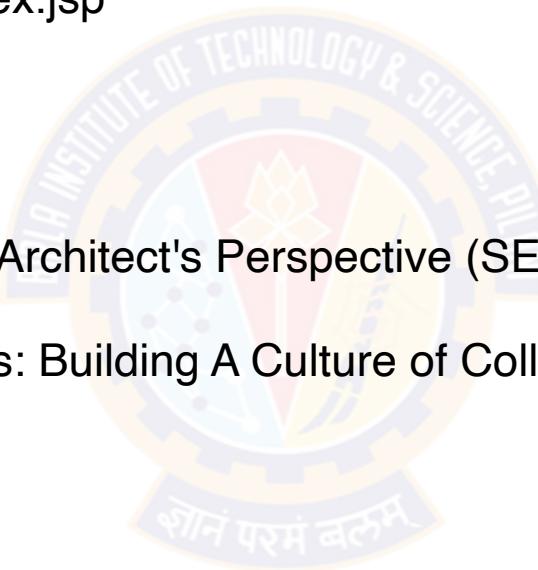


### Time Boxing

- Establish a team rule
- When the build breaks on check-in, try to fix it for ten minutes or any approximate time your environment can bare
- However it should not be so long
- If, after ten minutes, you aren't finished with the solution, revert to the previous version from your version control system

## CS 8 and 9

- Chapter 4, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
  - <https://www.seleniumhq.org/docs/index.jsp>
  - <https://www.sonarsource.com>
  - <https://docs.sonarqube.org>
- 
- Chapter 5, from DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu ,
  - Chapter 11,12, from Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Continuous Integration

- Using Continuous Integration Software:: Jenkins
- Artifact Management



# Continuous Integration System

## Jenkins

- The Jenkins project was started in 2004 (originally called Hudson) by Kohsuke Kawaguchi
- Open Source
- Offers more than 1400 plugins



# Jenkins

## Prepare your environment

- Need Version control system
- Java
- Install Jenkins
- Jenkins default port 8080



# Jenkins

## Post installation

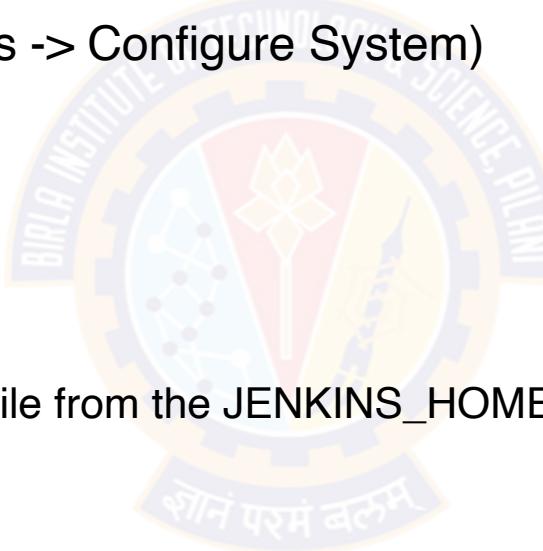
- **Unlocking Jenkins**
  - When you first access a new Jenkins instance, you are asked to unlock it using an automatically generated password
- Linux -> Jenkins console log output
- Windows -> `$JENKINS_HOME/secrets/initialAdminPassword`



# Jenkins

## Customization

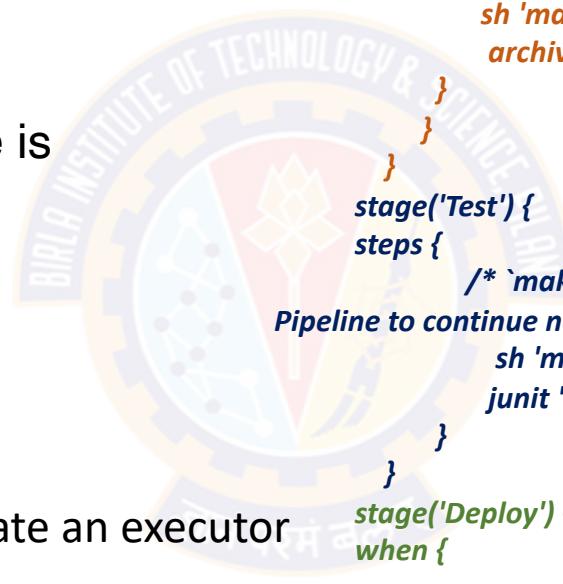
- Plugins
- Integration with Git (Manage Jenkins -> Manage Plugins)
- Integrating Maven (Manage Jenkins -> Configure System)
- <https://plugins.jenkins.io/>
- Removing plugin
  - Uninstall option from Jenkins UI
  - Removing the corresponding .hpi file from the JENKINS\_HOME/plugins directory on the master



# Jenkins

## Pipeline

- “Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins”
- The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile)



```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'make'
                archiveArtifacts artifacts: '**/target/*.jar'
            }
        }
        stage('Test') {
            steps {
                /* `make check` returns non-zero on test failures, * using `true` to allow the
                   Pipeline to continue nonetheless */
                sh 'make check || true'
                junit '**/target/*.xml'
            }
        }
        stage('Deploy') {
            when {
                expression {
                    currentBuild.result == null || currentBuild.result == 'SUCCESS'
                }
            }
            steps {
                sh 'make publish'
            }
        }
    }
}
```

In Figure:

1. Agent: It indicates that Jenkins should allocate an executor and workspace for this part of the Pipeline
2. Stage: It describes a stage of this Pipeline
3. Steps: It describes the steps to be run in this stage
4. SH: sh executes the given shell command (linux)
5. junit: It is a Pipeline step provided by the plugin

# Jenkins

## Why Jenkins Pipeline

### **Code:**

Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review

### **Durable:**

Pipelines can survive both planned and unplanned restarts of the Jenkins master

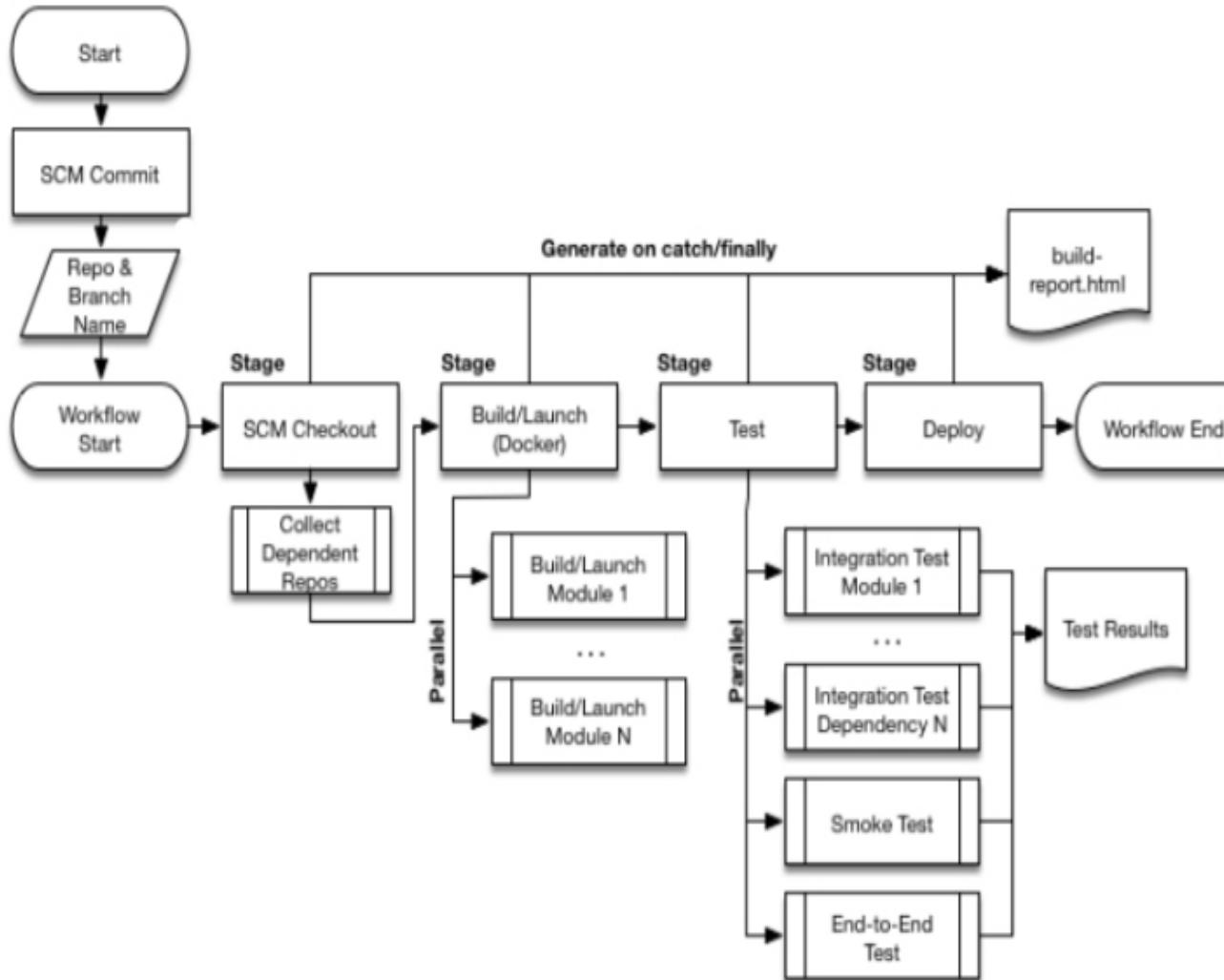
### **Pausable:**

Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run

### **Versatile:**

perform work in parallel

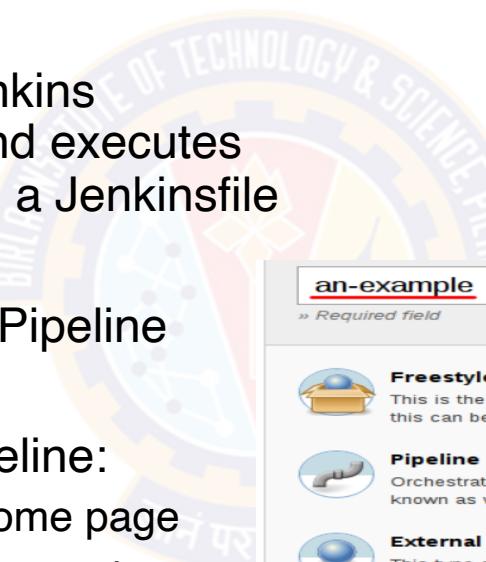
# Jenkins Pipeline



# Jenkins

## Multibranch

- The Multibranch Pipeline project type enables you to implement different Jenkinsfiles for different branches of the same project
- In a Multibranch Pipeline project, Jenkins automatically discovers, manages and executes Pipelines for branches which contain a Jenkinsfile in source control
- This eliminates the need for manual Pipeline creation and management
- Steps to To create a Multibranch Pipeline:
  - Step1: Click New Item on Jenkins home page
  - Step2 : Enter a name for your Pipeline, select Multibranch Pipeline and click OK



an-example  
» Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Pipeline**  
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**External Job**  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

**Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

**GitHub Organization**  
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

**Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

# Jenkins

## Multibranch

- Step3 : Add a Branch Source (for example, Git) and enter the location of the repository
- Step4: Save the Multibranch Pipeline project

The screenshot shows the Jenkins Multibranch Pipeline configuration page. The project name is 'an-example'. The 'Branch Sources' section is expanded, showing a dropdown menu with 'Git' selected. Other options in the menu include 'GitHub', 'Single repository & branch', 'Subversion', and 'Luna trigger'. A checkbox for 'Trigger builds remotely (e.g., from scripts)' is checked. The 'Git' configuration section shows a 'Project Repository' set to 'git://example.com/amazing-project.git', a 'Credentials' dropdown set to 'none', and a 'Repository browser' dropdown set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. The 'Property strategy' is set to 'All branches get the same properties'. A red box highlights the 'Git' option in the 'Branch Sources' dropdown menu.

Name: an-example

Display Name:

Description:

[Plain text] [Preview](#)

Branch Sources

Add source

Trigger builds remotely (e.g., from scripts)

**Git**

Project Repository: git://example.com/amazing-project.git

Credentials: - none -

Ignore on push notifications:

Repository browser: (Auto)

Additional Behaviours:

Advanced...

Property strategy: All branches get the same properties

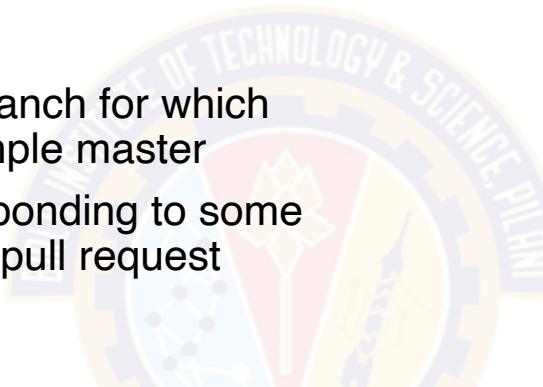
Add property:

Delete source:

# Jenkins

## Multibranch

- Step5: Build Trigger Interval can be set
- Additional Environment Variables
  - BRANCH\_NAME : Name of the branch for which this Pipeline is executing, for example master
  - CHANGE\_ID : An identifier corresponding to some kind of change request, such as a pull request number



**Build Triggers**

Trigger builds remotely (e.g., from scripts) ?

Build periodically ?

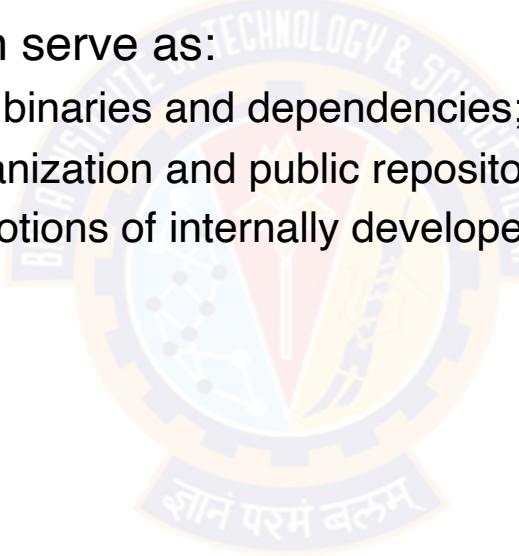
Periodically if not otherwise run ?

Interval  ▼ ?

# Artifact Management

## What is Artifact

- An artifact is the output of any step in the software development process
- Artifacts can be a number of things like JARs , WARs, Libraries, Assets and application
- Generally, an artifact repository can serve as:
  - A central point for management of binaries and dependencies;
  - A configurable proxy between organization and public repositories; and
  - An integrated depot for build promotions of internally developed software



# Artifact Management

## Artifact Management Tools

- An artifact repository should be:
  - secure;
  - trusted;
  - stable;
  - accessible; and
  - Versioned
- Artifact Management Tools:
  - JFrog
  - Nexus
  - Maven
  - HelixCore



# Artifact Management

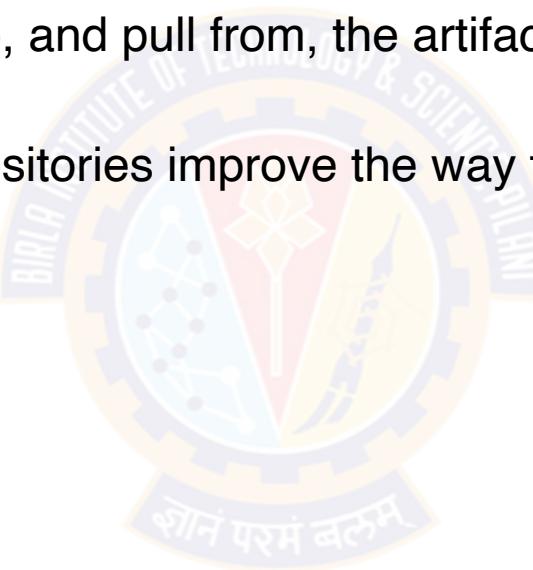
## Why

- Having an artifact repository allows you to treat your dependencies statically
- Artifact repositories allow you to store artifacts the way you use them
- Some repository systems only store a single version of a package at a time
- Ideally, your local development environment has the same access to your internal artifact repository as the other build and deploy mechanisms in your environment
- This helps minimize the “it works on my laptop” syndrome because the same packages and dependencies used in production are now used in the local development environment
- If you don’t have access to the internet within your environment; artifact management helps to have your own universe
- Relying on the internet for your dependencies means that somebody else ultimately owns the availability and consistency of your builds, something that many organizations hope to avoid
- An artifact repository organizes and manages binary files and their metadata in a central place

# Artifact Management

## Benefits

- The Binary Repository Manager: Artifact repositories allow teams to easily find the correct version of a given artifact
- This means developers can push to, and pull from, the artifact repository as part of the DevOps workflow
- Single Source of Truth: Artifact repositories improve the way teams work together by ensuring everyone is using the correct files
- It helps CI / CD to be more reliable





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

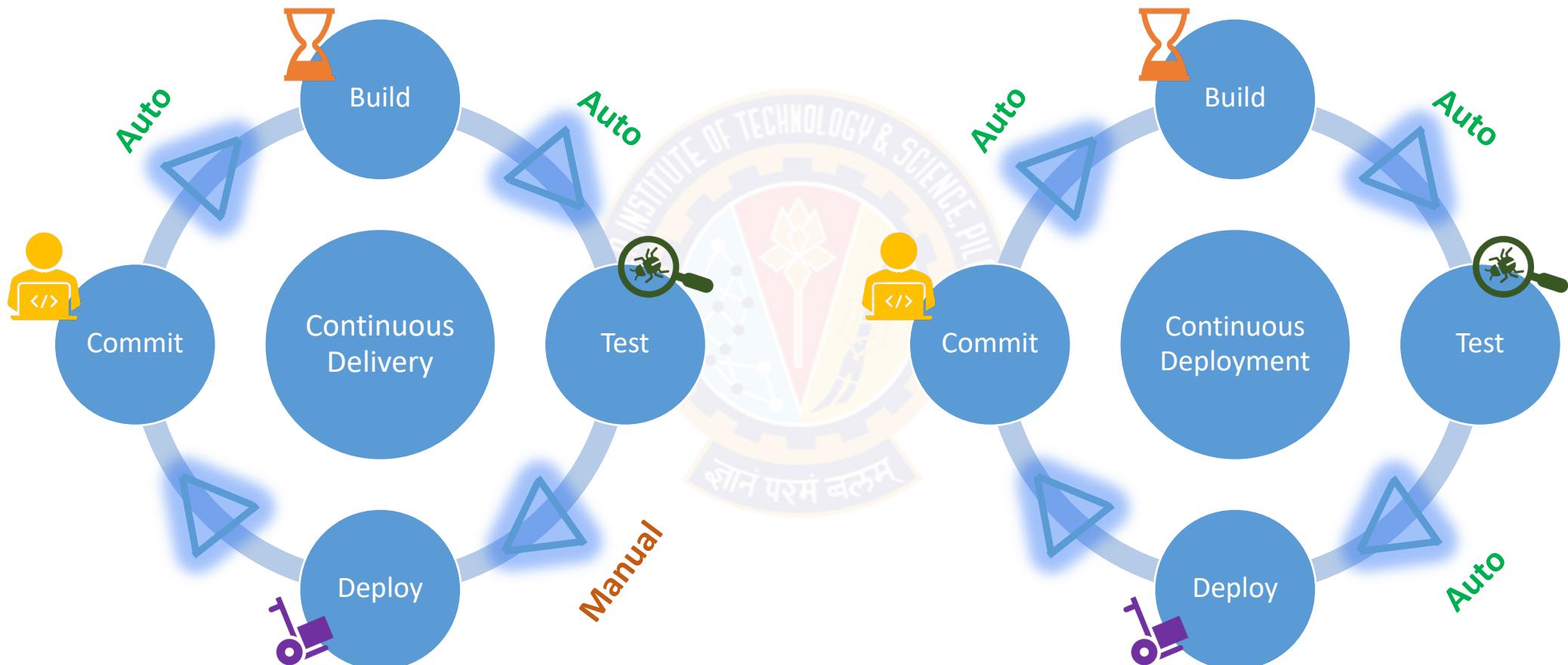
## Continuous Deployment

- Introduction to Deployment
- Deployment Consideration
- Challenges of Deployment
- Deployment pipeline
- Structure of Deployment Pipeline
- Basic Deployment Pipeline
- Stages of Deployment Pipeline



# Continuous Deployment

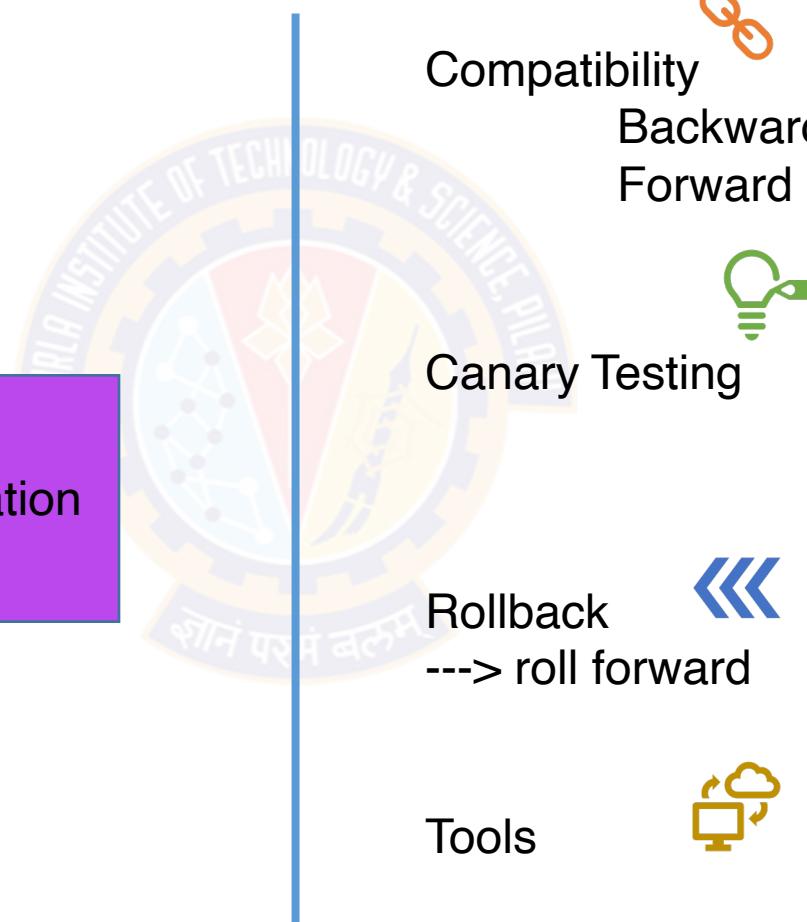
## Introduction to CD



# Deployment

## Considerations

Deployment Consideration



# Deployment

## Challenges in Deployment

- Deployment Process Waste

Build and operations teams waiting for documentation or fixes

Testers waiting for “good” builds of the software

Development teams receiving bug reports weeks after the team has moved on to new functionality

Discovering, towards the end of the development process, that the application’s architecture will not support the system’s nonfunctional requirements

*This leads to software that is undeployable because it has taken so long to get it into a production-like environment, and buggy because the feedback cycle between the development team and the testing and operations team is so long*

# Deployment

## New Approach

- End-to-End approach to delivering software
- Deployment of Application should be easy & one click to go
- A powerful feedback loop; rapid feedback on both the code and the deployment process
- Lowering the risk of a release and transferring knowledge of the deployment process to the development team
- Testing teams deploy builds into testing environments themselves, at the push of a button
- Operations can deploy builds into staging and production environments at the push of a button
- Developers can see which builds have been through which stages in the release process, and what problems were found
- Automate Build, Deploy, Test and Release System

**Note:** As a result, everybody in the delivery process gets two things: access to the things they need when they need them, and visibility into the release process to improve feedback so that bottlenecks can be identified, optimized, and removed.

This leads to a delivery process which is not only faster but also safer

# Deployment Pipeline

## What is Deployment Pipeline

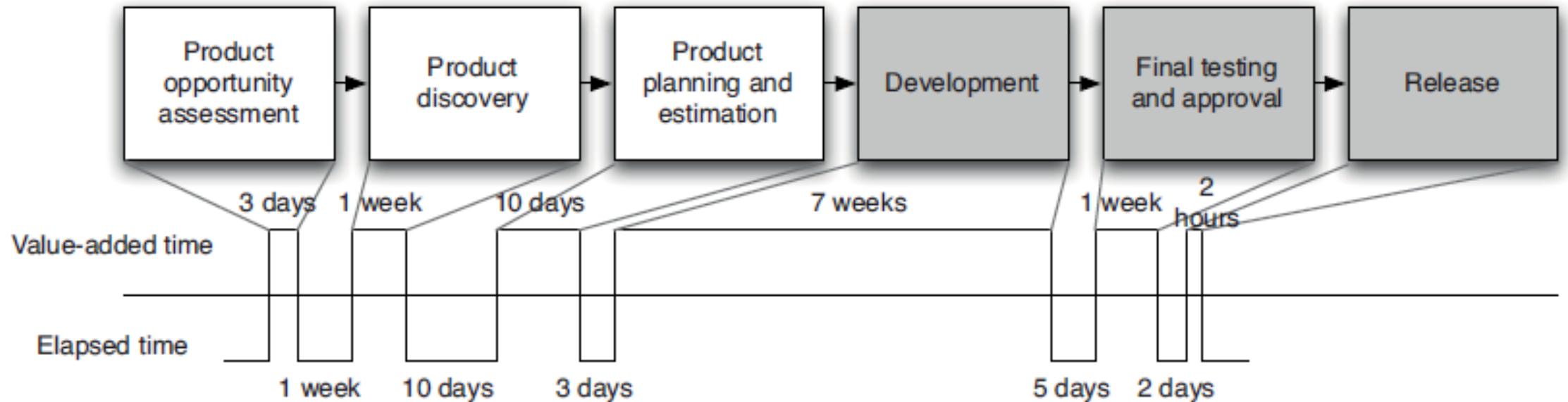
- An automated manifestation of your process for getting software from version control into the production



Increase the collaboration between many individuals

# Deployment Pipeline

## Value Stream Map of a Product Creation



A high-level value stream map for the creation of a new product

Cycle Time : 11 Weeks, 3 Days & 2 hours

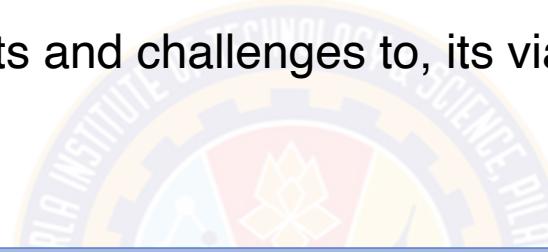
Waste Time : 5 Weeks

Lead Time : 16 Weeks, 3 Days & 2 hours

# Structure of Deployment Pipeline

## Expected steps to be followed

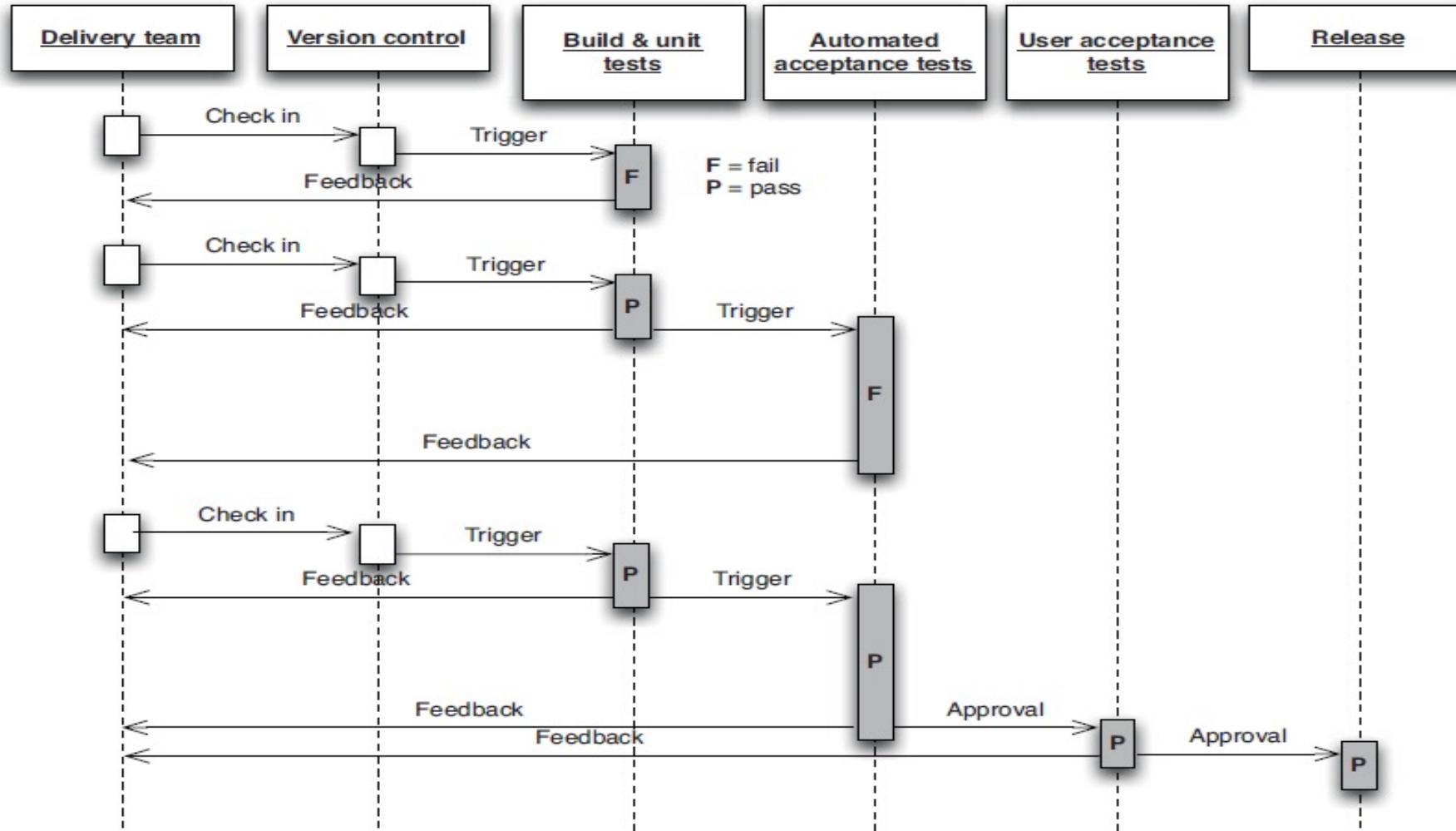
1. The input to the pipeline is a particular revision in version control
2. Every change creates a build
3. It pass through a sequence of tests and challenges to, its viability as a production release



- As the build passes each test of its fitness, confidence in it increases
- The objective is to eliminate unfit release candidates as early in the process as we can and get feedback on the root cause of failure to the team as rapidly as possible
- Any build that fails a stage in the process will not generally be promoted to the next

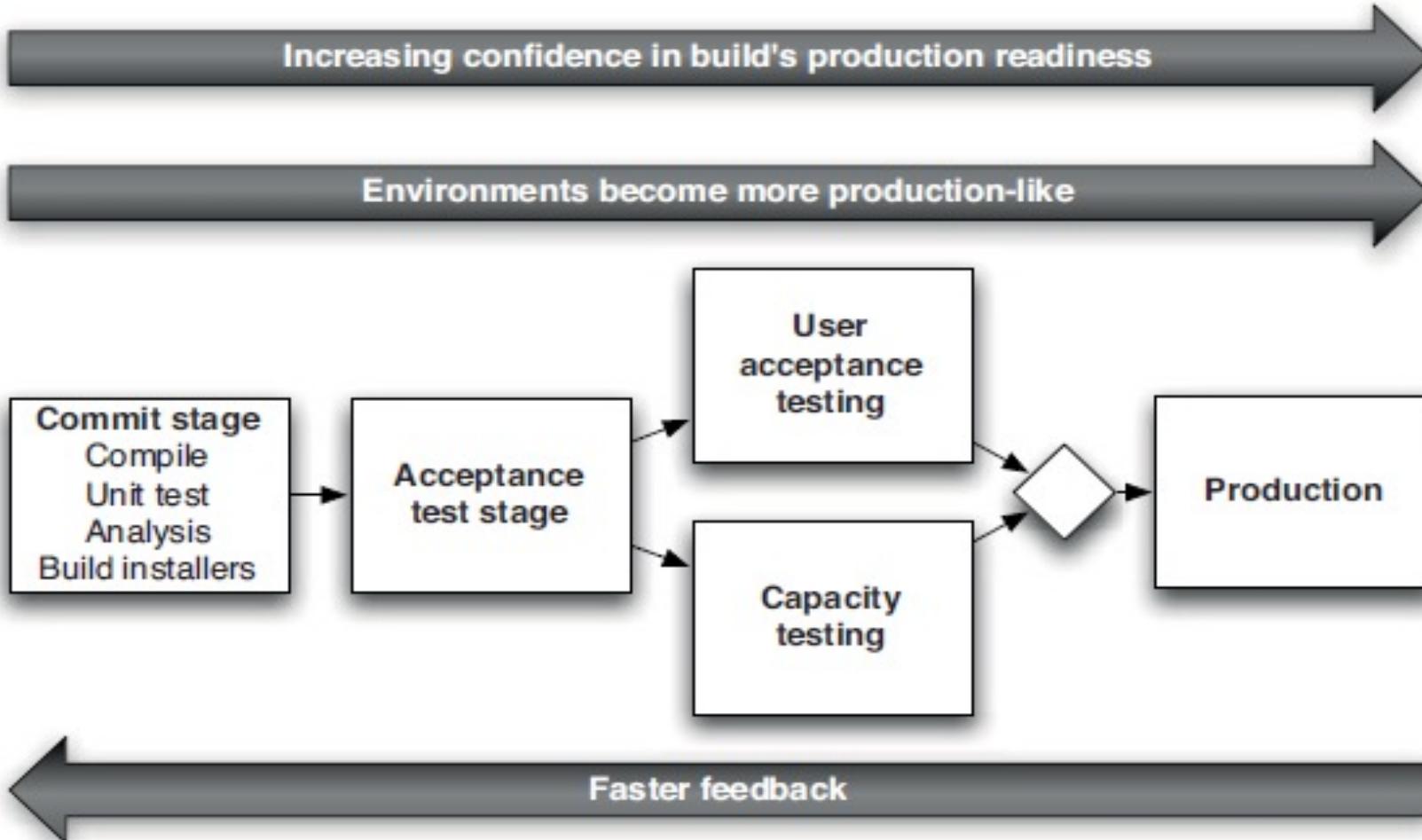
# Structure of Deployment Pipeline

## Example



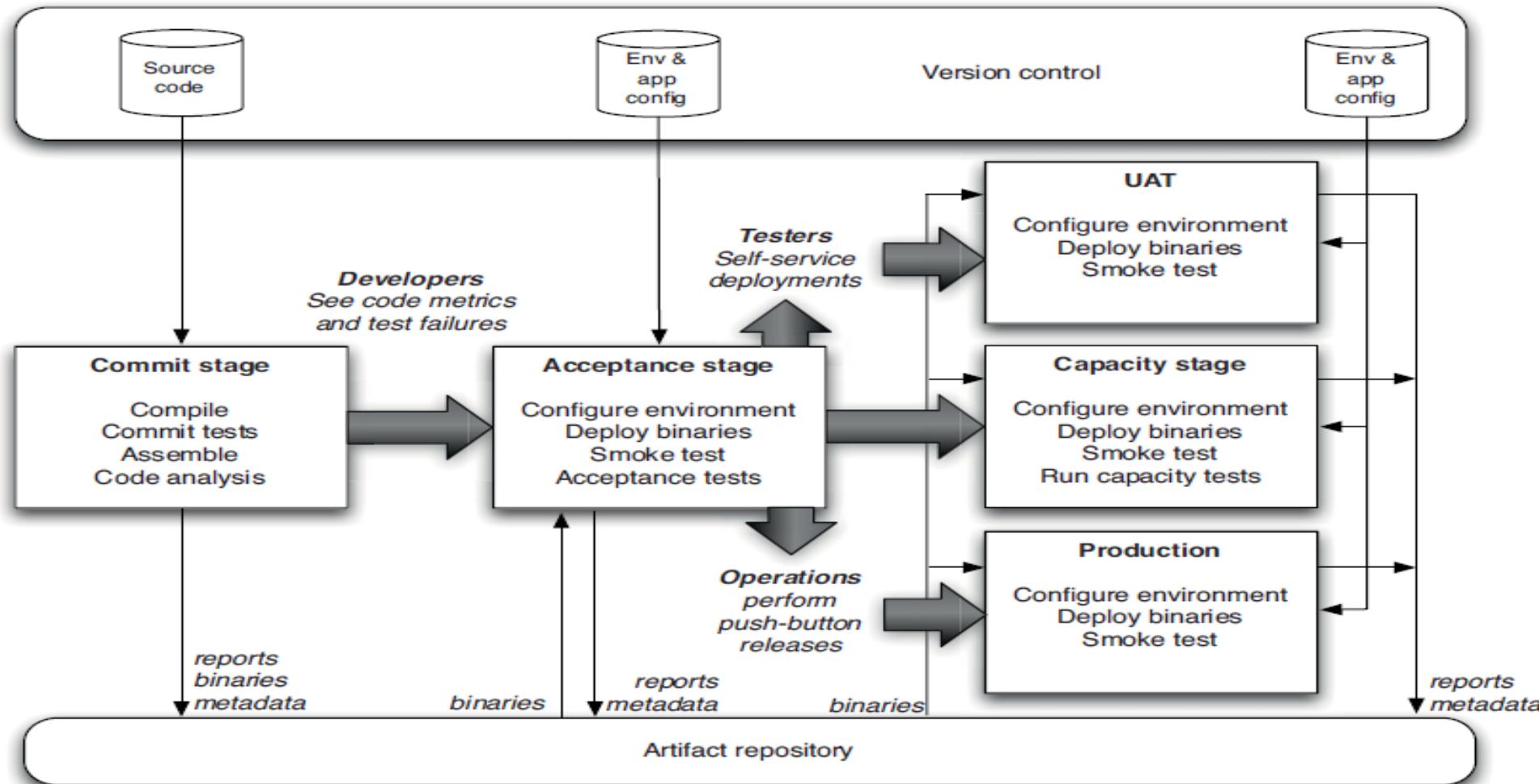
# Structure of Deployment Pipeline

## Broad Overview



# Deployment Pipeline

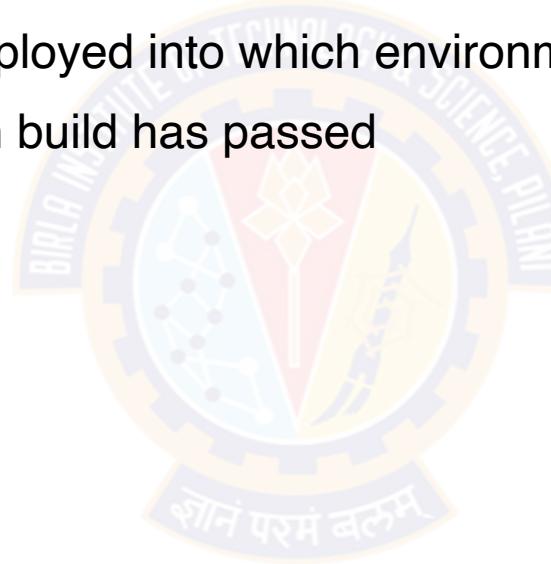
## Basic Deployment Pipeline



# Basic Deployment Pipeline

## Outcome

- The ultimate purpose of all this is to get feedback as fast as possible
- To make the feedback cycle fast
- To be able to see which build is deployed into which environment and
- Which stages in your pipeline each build has passed



**Note:** It means that if you see a problem in the acceptance tests (for example), you can immediately find out which changes were checked into version control that resulted in the acceptance tests failing

# Deployment Pipeline

## Antipatterns of dealing with Binaries

- The source code will be compiled repeatedly in different contexts: during the commit process, again at acceptance test time, again for capacity testing, and often once for each separate deployment target



- Every time you compile the code, you run the risk of introducing some difference
- The version of the compiler installed in the later stages may be different from the version that you used for your commit tests
- You may pick up a different version of some third-party library that you didn't intend
- Even the configuration of the compiler may change the behavior of the application

# Antipatterns of dealing with Binaries

## Violates two Principles

### Efficiency of Deployment pipeline

Recompiling violates this principle because it takes time, especially in large systems  
Delayed feedback

### Always build upon foundations

The binaries that get deployed into production should be exactly the same as those that went through the acceptance test process  
Recreation of binaries violates this principle

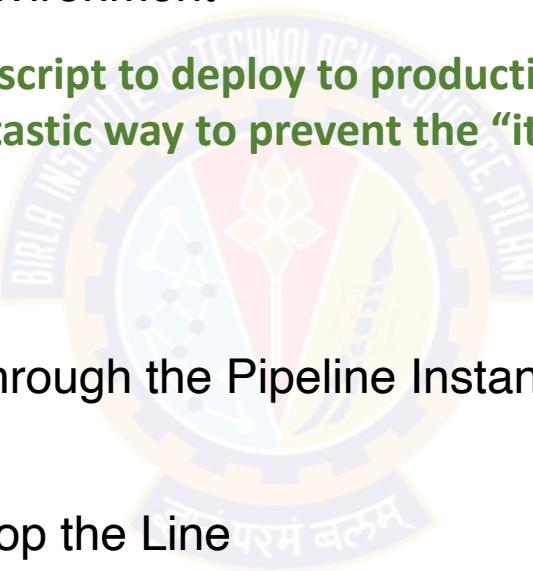
# Deployment Pipeline

## Deployment Pipeline Practices

1. Only Build Your Binaries Once
2. Deploy the Same Way to Every Environment

**Note: Using the same script to deploy to production that you use to deploy to development environments is a fantastic way to prevent the “it works on my machine” syndrome**

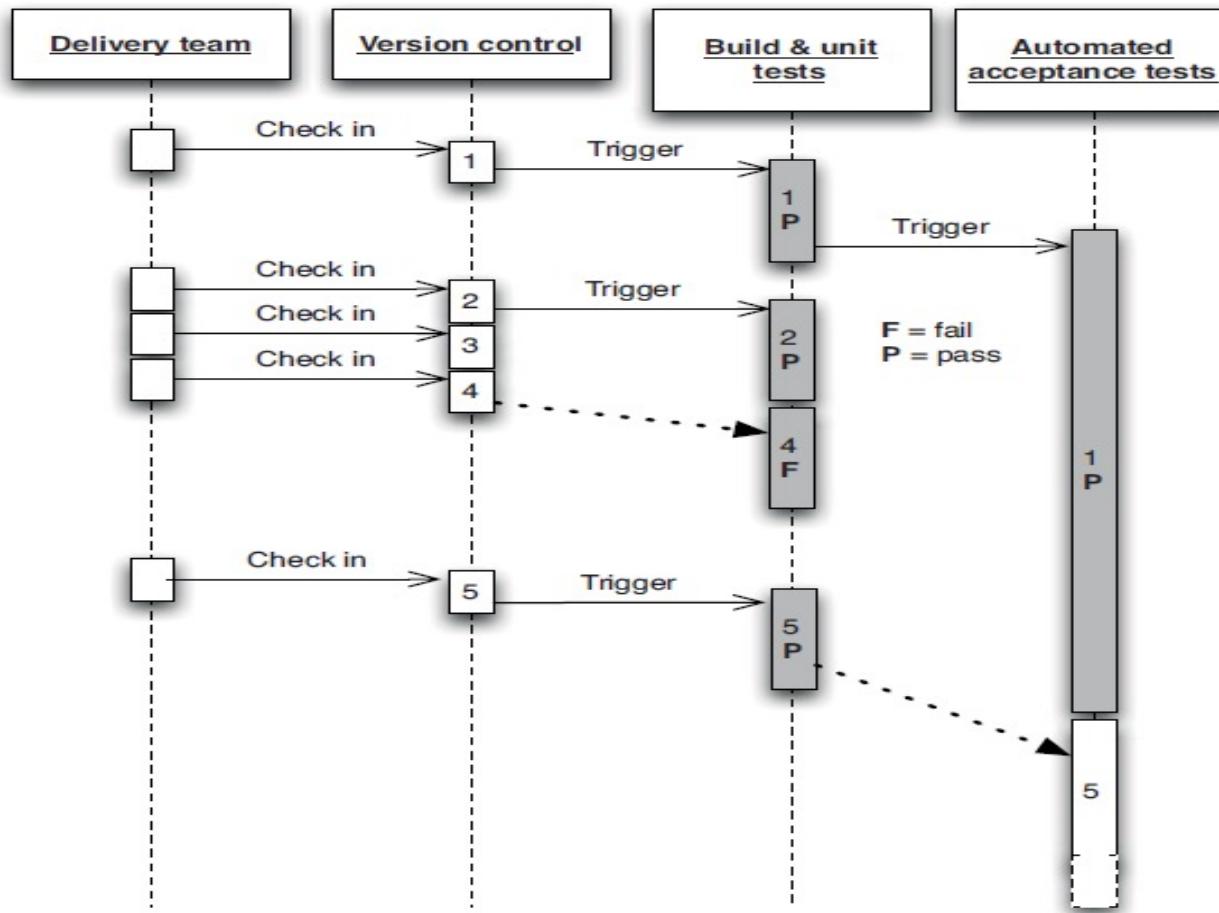
3. Smoke-Test Your Deployments
4. Deploy into a Copy of Production
5. Each Change Should Propagate through the Pipeline Instantly
6. Intelligent pipeline scheduling
7. If Any Part of the Pipeline Fails, Stop the Line



# Deployment Pipeline Practices

## Intelligent scheduling

- Intelligent scheduling is crucial to implementing a deployment pipeline



# Deployment Pipeline

## Stages in Deployment Pipeline

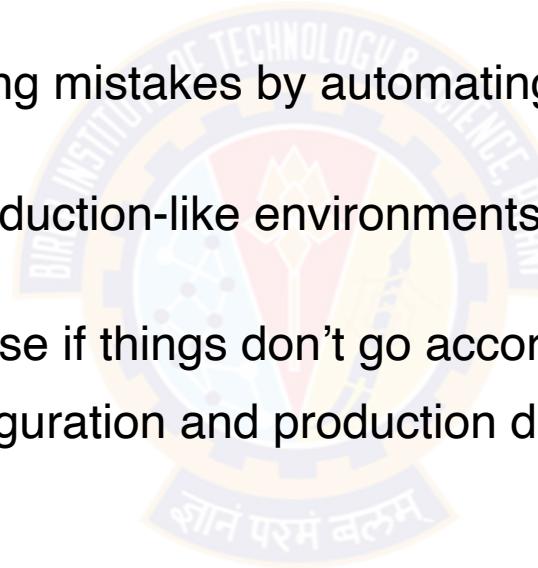
- The Commit Stage
- The Automated Acceptance Gate
- Subsequent Test Stages
- Preparing to Release



# Preparing to Release

## Release Plan

- Have a release plan that is created and maintained by everybody involved in delivering the software, including developers and testers, as well as operations, infrastructure, and support personnel
- Minimize the effect of people making mistakes by automating as much of the process as possible
- Practice the procedure often in production-like environments, so you can debug the process and the technology supporting it
- Have the ability to back out a release if things don't go according to plan
- Have a strategy for migrating configuration and production data as part of the upgrade and rollback processes



**Note: Goal is a completely automated release process, Releasing should be as simple as choosing a version of the application to release and pressing a button and Backing out should be just as simple**



# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Continuous Deployment

- Human Free Deployment
- Implementing a Deployment Pipeline
- Rolling back deployment
- Zero-downtime Release
- Deployment Strategies



# Human Free Deployments

## Automating Deployment and Release

- Reduces issues due manual mistakes
- Audit logs



# Human Free Deployments

## Benefits

- With automated deployment and release, the process of delivery becomes available to everyone
- Developers, testers, and operations teams no longer need to rely on ticketing systems and email threads to get builds deployed so they can gather feedback on the production readiness of the system
- Testers can decide which version of the system they want in their test environment without needing to be technical experts themselves, nor relying on the availability of such expertise to make the deployment
- Sales people can access the latest version of the application with the killer feature that will swing the deal with a client
- An important reason for the reduction in risk is the degree to which the process of release itself is rehearsed, tested, and perfected
- Since you use the same process to deploy your system to each of your environments and to release it, the deployment process is tested very frequently—perhaps many times a day

# Implementing a Deployment Pipeline

## The steps to implement a deployment pipeline

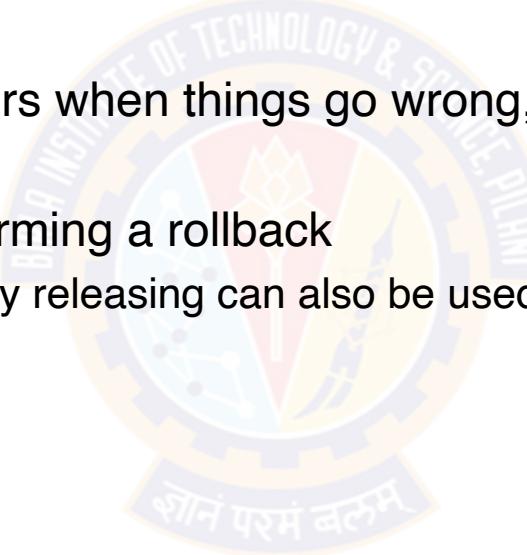
- Model your value stream and create a walking skeleton
- Automate the build and deployment process
- Automate unit tests and code analysis
- Automate acceptance tests
- Automate releases



# Deployment Consideration

## Rolling Back Deployments

- It is essential to roll back a deployment in case it goes wrong
- As, debugging problems in a running production environment is almost certain to result in late nights
- a way to restore service to your users when things go wrong, so you can debug the failure in the comfort of normal working hours
- There are several methods of performing a rollback
  - blue-green deployments and canary releasing can also be used to perform zero-downtime releases and rollbacks



# Rolling Back Deployments

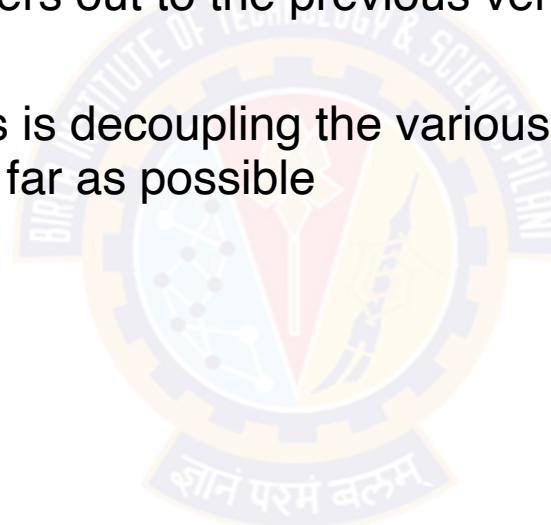
## Rolling Back Deployments Constraints

- Data : If your release process makes changes to your data, it can be hard to roll back
- There are two general principles you should follow when creating a plan for rolling back a release
- Ensure that the state of your production system, including databases and state held on the filesystem, is backed up before doing a release
- The second is to practice your rollback plan, including restoring from the backup or migrating the database back before every release to make sure it works

# Deployments

## Zero-Downtime Releases

- Hot deployment, in which the process of switching users from one release to another happens nearly instantaneously
- It must also be possible to back users out to the previous version nearly instantaneously too, if something goes wrong
- The key to zero-downtime releases is decoupling the various parts of the release process so they can happen independently as far as possible



# Deployment

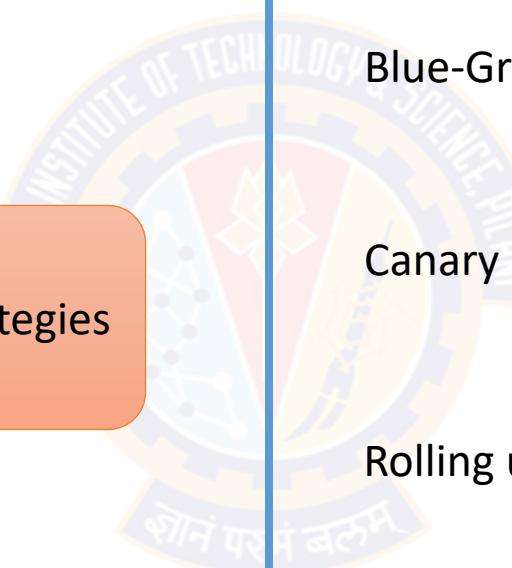
## Deployment Strategies

Deployment Strategies

Blue-Green Deployments

Canary Releasing

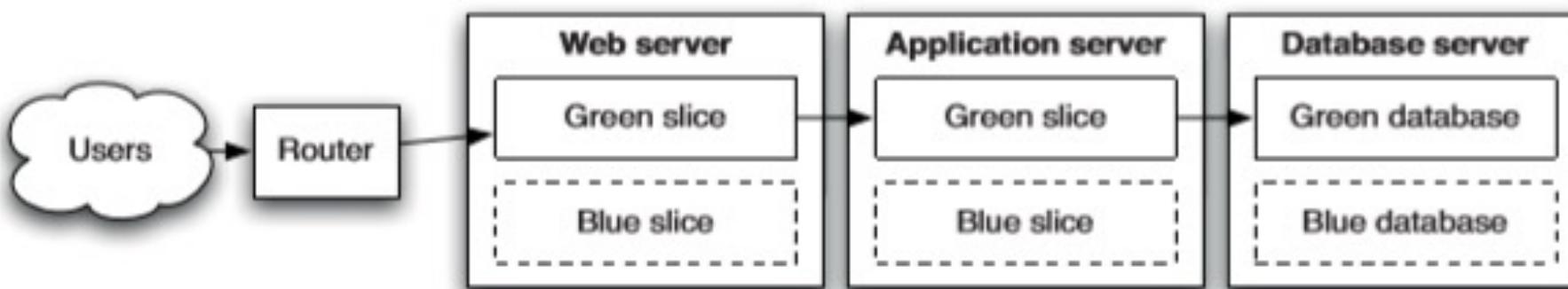
Rolling upgrade



# Blue-Green Deployments

## Introduction

- This is one of the most powerful techniques we know for managing releases
- The idea is to have two identical versions of your production environment, which we'll call blue and green



# Blue-Green Deployments

## Challenges

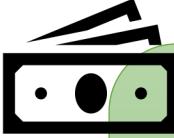
- It is usually not possible to switch over directly from the green database to the blue database because it takes time to migrate the data from one release to the next if there are schema changes

### Solutions:

- 1) Put the application into read-only mode shortly before switchover
- 2) Another approach is to design your application so that you can migrate the database independently of the upgrade process

# Blue-Green Deployments

## Setup



blue and green environments can be completely separate replicas of each other



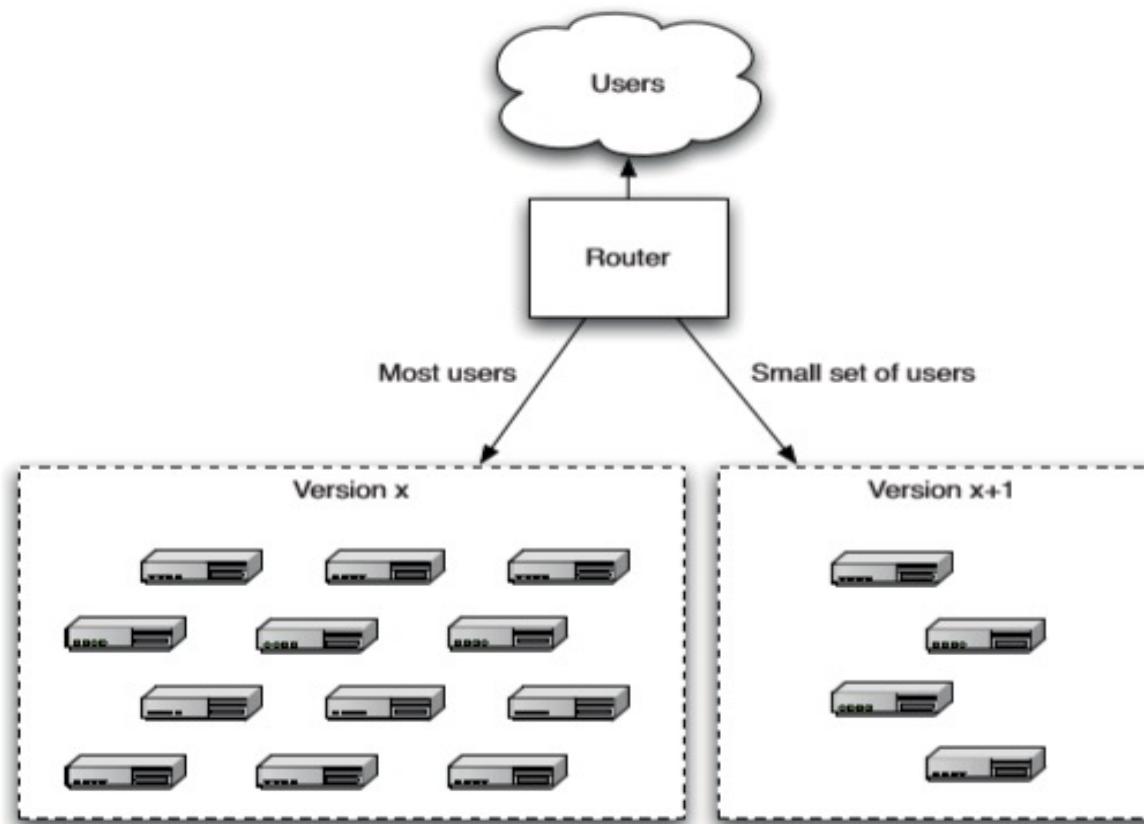
you can only afford a single production environment  
have two copies of your application running side by side on the same environment (with its own resources like port, filesystem etc.,)

Use virtualization

# Canary Releasing

## Introduction

- Involves rolling out a new version of an application to a subset of the production servers to get fast feedback
- Like a canary in a coal mine, this quickly uncovers any problems with the new version without impacting the majority of users



# Canary Releasing

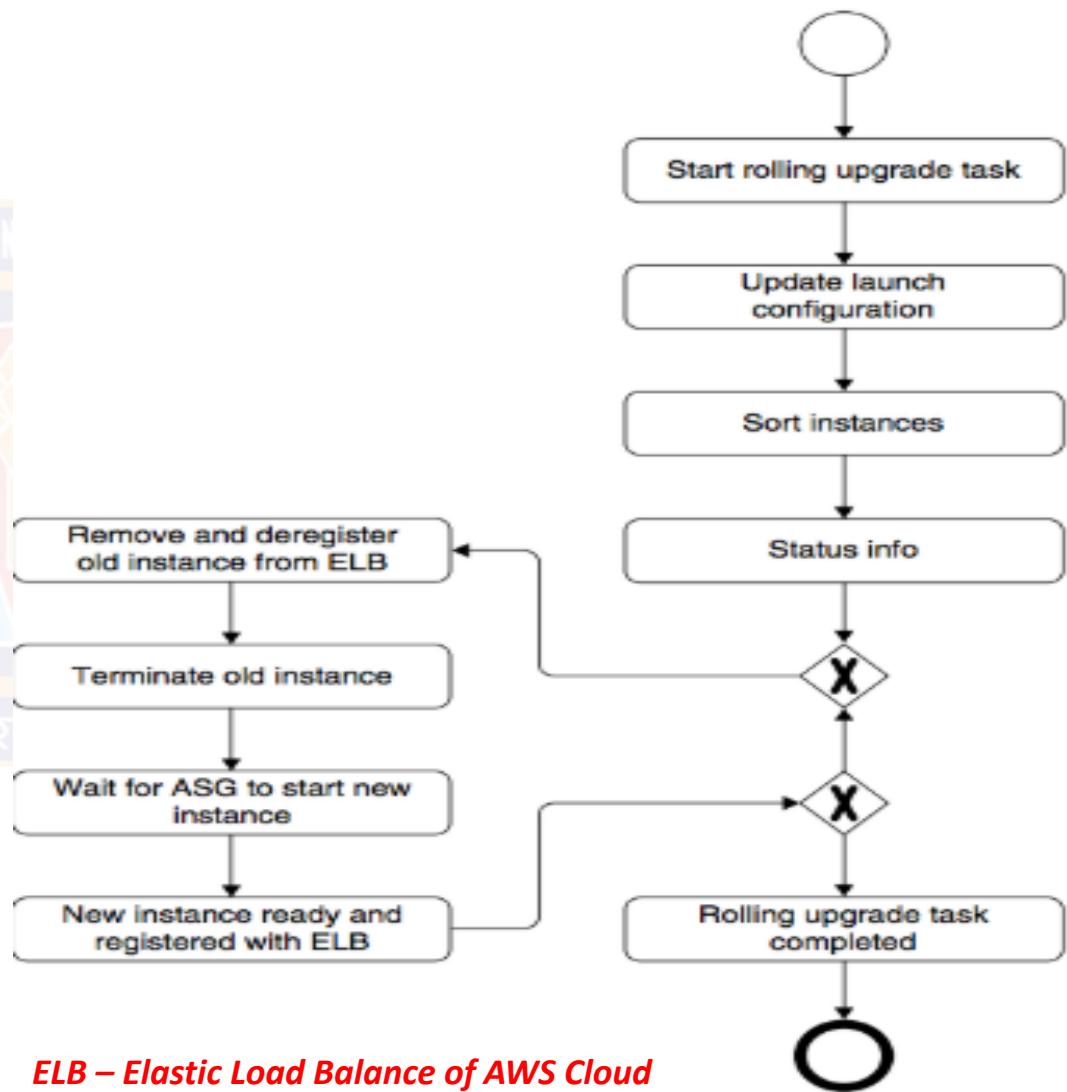
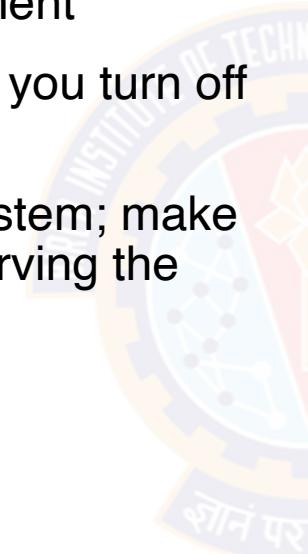
## Benefits

- It makes rolling back easy: Just stop routing users to the bad version, and you can investigate the logs at your freedom
- You can use it for A/B testing by routing some users to the new version and some to the old version
  - Some companies measure the usage of new features, and kill them if not enough people use them
  - Others measure the actual revenue generated by the new version, and roll back if the revenue for the new version is lower
- You can check if the application meets capacity requirements by gradually ramping up the load, slowly routing more and more users to the application and measuring the application's response time and metrics like CPU usage, I/O, and memory usage, and watching for exceptions in logs

# Rolling upgrade

## Introduction

- A rolling upgrade consists of deploying a small number of new version systems at a time directly to the production environment
- In this deployment simultaneously you turn off the old version system
- Before you remove the original system; make sure the new version system is serving the purpose



# Rolling upgrade

## Benefits

- Cost Effective
- Risk Effective



# Emergency Fixes

## Best Practice for Emergency Fixes

- Run every emergency fix through your standard deployment pipeline
- You should always consider how many people the defect affects, how often it occurs, and how severe the defect is in terms of its impact on users
- Some considerations to take into account when dealing with a defect in production:
  - Never do them late at night, and always pair with somebody else
  - Make sure you have tested your emergency fix process
  - Only under extreme circumstances bypass the usual process for making changes to your application
  - Make sure you have tested making an emergency fix using your staging environment
  - Sometimes it's better to roll back to the previous version than to deploy a fix, do some analysis to work out what the best solution is

# Deployment

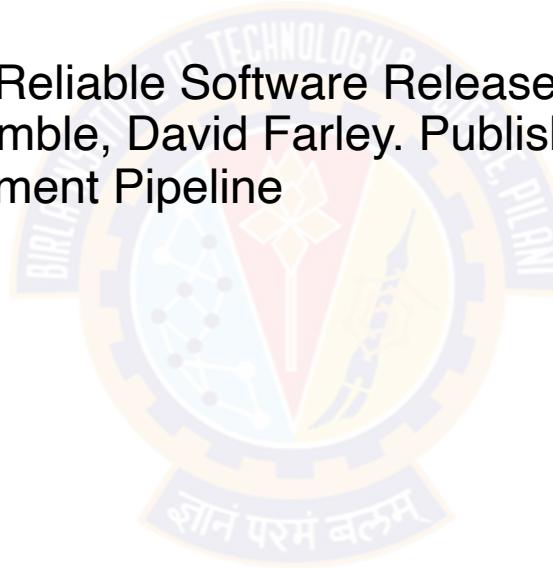
## Tips and Tricks

- The People Who Do the Deployment Should Be Involved in Creating the Deployment Process
  - Things Go Better When Development and Operations Are Friends
- Log Deployment Activities
  - If your deployment process isn't completely automated, including the provisioning of environments, it is important to log all the files that your automated deployment process copies or creates
- Don't Delete the Old Files, Move Them
- Deployment Is the Whole Team's Responsibility
  - A “build and deployment expert” is an antipattern. Every member of the team should know how to deploy, and every member of the team should know how to maintain the deployment scripts

# References

## CS 11 and 12

- Text Book 1: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu , Publisher: Addison Wesley (18 May 2015) : Chapter 6 : Deployment
- Text Book 2: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Publisher: Addison Wesley, 2011 : Chapter 5 : Anatomy of the Deployment Pipeline





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Review CS#7

## Continuous Deployment

- Introduction to Deployment
- Deployment Consideration
- Challenges of Deployment
- Deployment pipeline
- Structure of Deployment Pipeline
- Basic Deployment Pipeline
- Stages of Deployment Pipeline
- Human Free Deployment
- Implementing a Deployment Pipeline
- Rolling back deployment
- Zero-downtime Release
- Deployment Strategies



# Agenda

## Continuous Monitoring

- Introduction to Monitoring
- What to Monitor
- Goals of Monitoring
  - Failure detection
  - Performance degradation
  - Capacity planning
  - User Interaction
  - Intrusion detection
- How to Monitor
- Challenges in Monitoring
- Monitoring Tools
- ELK
- ELK Architecture
- ELK Features and Benefits



# Continuous Monitoring

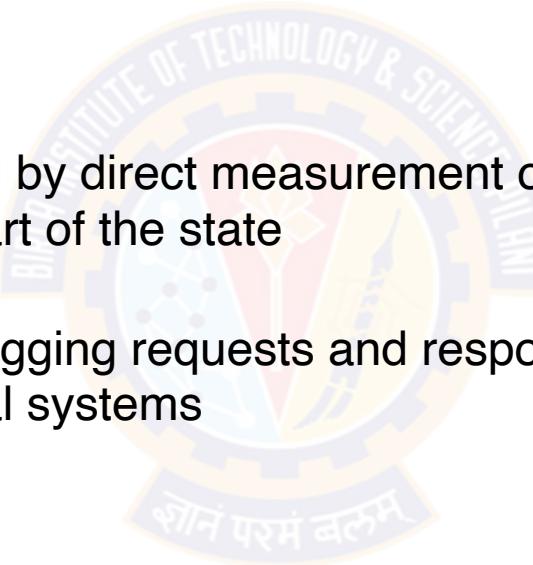
## Introduction to Monitoring

- The process of observing and recording system state changes and data flows



State changes can be expressed by direct measurement of the state or by logs recording updates that impact part of the state

Data flows can be captured by logging requests and responses between both internal components and external systems



# Continuous Monitoring

## Monitoring fall into five different categories

1) Identifying failures and the associated faults both at runtime and during postmortems held after a failure has occurred

2) Identifying performance problems of both individual systems and collections of interacting systems

3) Characterizing workload for both short- and long-term capacity planning and billing purposes

4) Measuring user reactions to various types of interfaces or business offerings

5) Detecting intruders who are attempting to break into the system

# Continuous Monitoring

## What to Monitor

- The data to be monitored for the most part comes from the various levels of the stack

| Goal of Monitoring                  | Source of Data                 |
|-------------------------------------|--------------------------------|
| Failure Detection                   | Application and Infrastructure |
| Performance Degradation Detection   | Application and Infrastructure |
| Capacity Planning                   | Application and Infrastructure |
| User reaction to business offerings | Application                    |
| Intruder detection                  | Application and Infrastructure |

Above Table lists the insights you might gain from the monitoring data and the portions of the stack where such data can be collected

# Continuous Monitoring

## Fundamental items to be monitored

### Inputs

### Resources

hard resources such as CPU, memory, disk, and network—even if virtualized  
soft resources such as queues, thread pools, or configuration specifications

### Outcomes

include items such as transactions and business-oriented activities

# Goals Monitoring

## Failure Detection

Failures of any element in physical infrastructure is possible

The total failures are relatively easy to detect

No data is flowing where data used to flow

Partial failures that are difficult to detect

Partial failures also manifest as performance problems

# Goals Monitoring

## Failure Detection

Detecting software failures :

- 1) The monitoring software performs health checks on the system from an external point
- 2) A special agent inside the system performs the monitoring
- 3) The system itself detects problems and reports them

Hardware Failure:  
datacenter provider's responsibility

Software Failure:  
Dependency Software failure  
Software Misconfiguration

# Goals of Monitoring

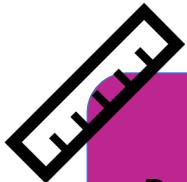
## Performance Degradation

- Degraded performance can be observed by comparing current performance to historical data—or by complaints from clients or end users
- Ideally your monitoring system catches performance degradation before users are impacted at a notable strength



# Goals of Monitoring

## Performance Degradation



### Performance measures

#### Latency

The time from the initiation of an activity to its completion  
It is the period from a user request to the satisfaction of that request

#### Throughput

The number of operations of a particular type in a unit time

#### Utilization

The relative amount of use of a resource  
Hard resources : CPU (80%), Memory, disk  
Soft resources: queues or thread pools

# Goals of Monitoring

## Capacity Planning



### Long-term Capacity Planning

Involves humans and has a time frame on the order of days, weeks, months, or even years

This capacity planning is intended to match hardware needs, whether real or virtualized, with workload requirements

#### Example:

In a physical datacenter, it involves ordering hardware

In a virtualized public datacenter, it involves deciding on the number and characteristics of the virtual resources

*Note: In capacity planning characterization of the current workload gathered from monitoring data and a projection of the future workload based on business considerations and the current workload*

# Goals of Monitoring

## Capacity Planning



### Short-term Capacity Planning

Planning is performed automatically and has a time frame on the order of minutes

In this capacity planning the context of a virtualized environment such as the cloud, creating a new virtual machine (VM) for an application or deleting an existing VM

#### Example:

Monitoring the usage of the current VM instances was an important portion of each option

*Note: Charging for use is an essential characteristic of the cloud as defined by the U.S. National Institute of Science and Technology*

# Goals of Monitoring

## User Interaction



User satisfaction depends

The latency of a user request

The reliability of the system with which the user is interacting

User interface modification

Types of User Interaction Monitoring:  
Real user monitoring (RUM)  
Synthetic monitoring

# Goals of Monitoring

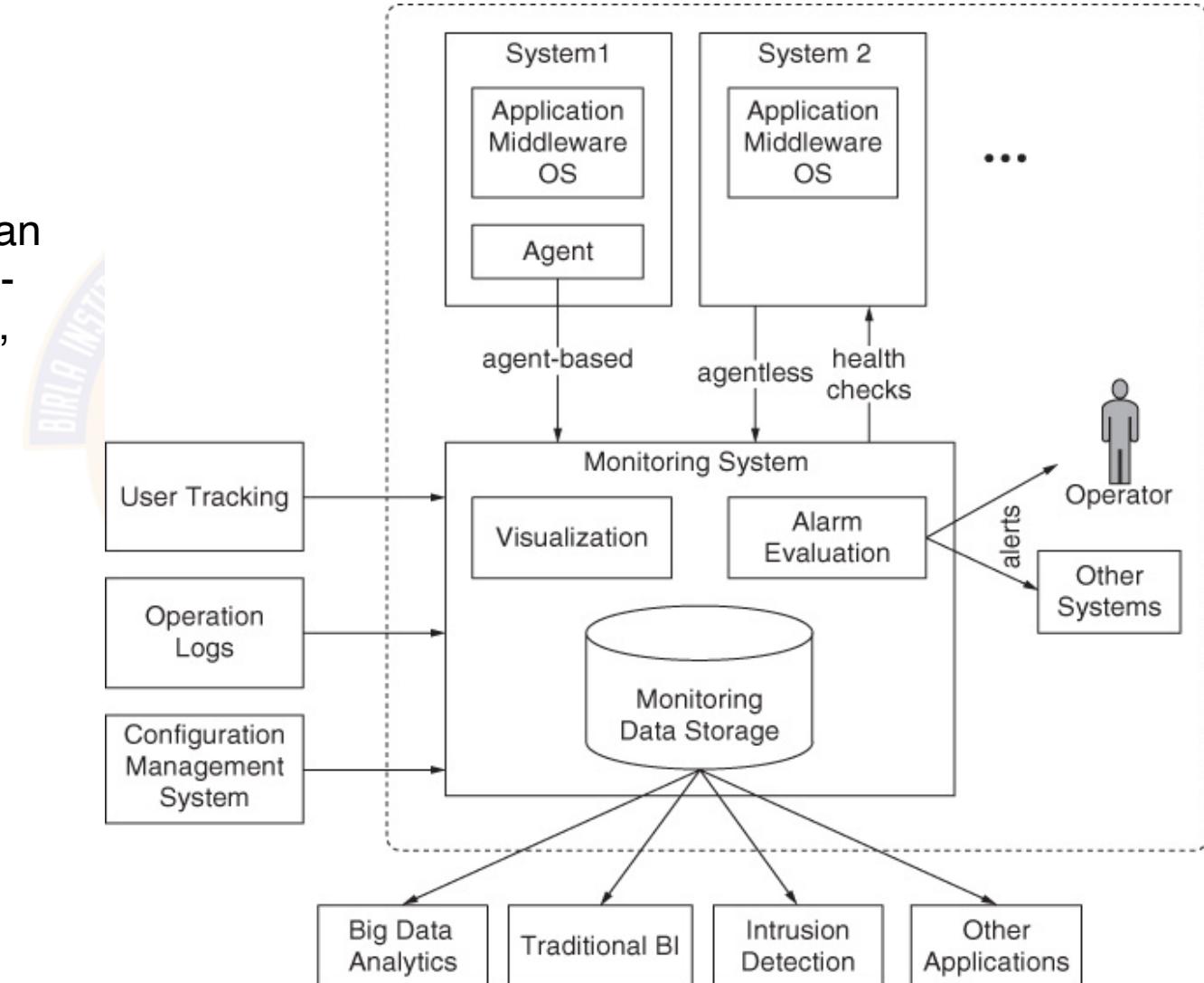
## Intrusion Detection

- Intruders can break into a system by disrupting an application
  - Example: Through incorrect authorization
- Applications can monitor users and their activities to determine whether the activities are consistent with the users' role in the organization
- An intrusion detector is a software application that monitors network traffic by looking for abnormalities
- Intrusion detectors use a variety of different techniques to identify attacks:
  - They use historical data from an organization's network to understand what is normal
  - They use libraries that contain the network traffic patterns observed during various attacks
  - Example: Current traffic on network vs Expected traffic in historical data
    - The organization may have a policy disallowing external traffic on particular ports

# Continuous Monitoring

## How to Monitor?

- Agentless
- Agent-based
- Health Check: External systems can also monitor system or application-level states through health checks, performance-related requests, or transaction monitoring



# Continuous Monitoring

## Why Monitoring

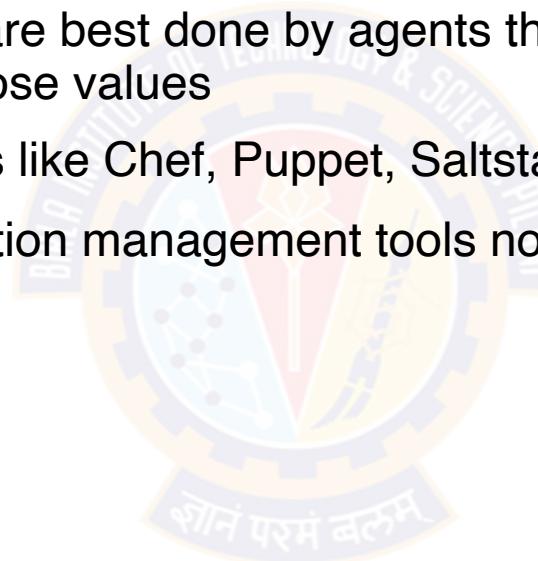
- A monitoring system allows operators to drill down into detailed monitoring data and logs; which helps in:
- Error diagnosis
- Root Cause Analysis
- Deciding on the best reaction to a problem



# Monitoring Operation Activities

## Monitoring Operations

- Operations tools monitor resources such as configuration settings to determine whether they conform to pre-specified settings and monitor resource specification files to identify any changes
- Both of these types of monitoring are best done by agents that periodically sample the actual values and the files that specify those values
- There are different Operation Tools like Chef, Puppet, Saltstack and Ansible etc.,
- The offerings of different configuration management tools now available with both Agent Based and Agentless

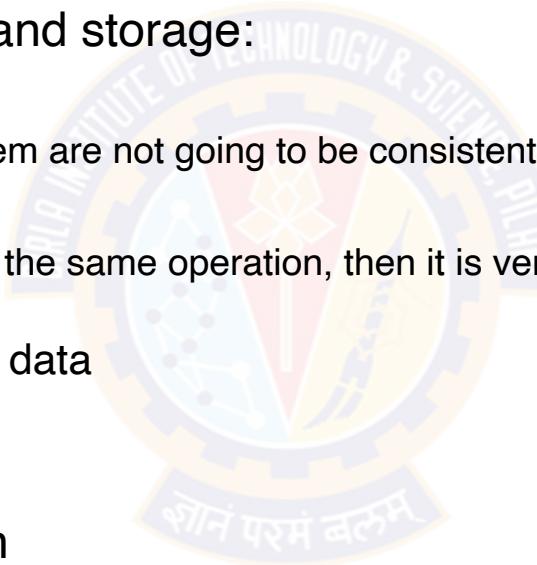


# Monitoring Operation Activities

## Collection and Storage



- The core of monitoring is recording and analyzing time series data, namely, a sequence of time-stamped data points
- Three key challenges in collection and storage:
  - Collating related items by time:
    - Time stamps in a distributed system are not going to be consistent
  - Collating related items by context:
    - If there is any parallel process for the same operation, then it is very difficult to reconstruct a sequence of events to diagnose a problem
  - Handling the volume of monitoring data
    - Big Data, Hadoop etc.
- Change in Monitoring Configuration



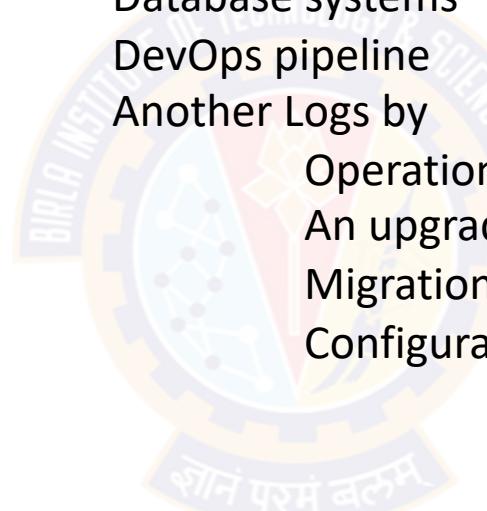
# Monitoring Operation Activities

## Log



sources of the logs

Use of Logs



Applications  
Web servers  
Database systems  
DevOps pipeline  
Another Logs by  
Operations tools  
An upgrade tool  
Migration tool  
Configuration management tool

During operations to detect and diagnose problems  
During debugging to detect errors  
During post-problem forensics to understand the sequence that led to a particular problem

# Monitoring Operation Activities

## Log



General rules about writing logs

Logs should have a consistent format

Logs should include an explanation for why this particular log message was produced

Log entries should include context information (Process ID, Request ID, VM ID etc.,)

Logs should provide screening information (Severity level, Alert level)

# Monitoring Operation Activities

## Graphing and Display

- It is useful to visualize all relevant data collected by monitoring system



# Monitoring Operation Activities

## Alarms and Alerts



### Alerts:

Alerts are raised for purposes of informing

Alerts are raised in advance of an alarm

Example: The datacenter temperature is rising

Alarms and alerts can be triggered by Events

A particular physical machine is not responding

By values crossing a threshold

The response time for a particular disk is greater than an acceptable value

By sophisticated combinations of values and trends

Percentage monitoring of a file system

CPU Utilization on peak in a Day



### Alarms:

Alarms require action by the operator

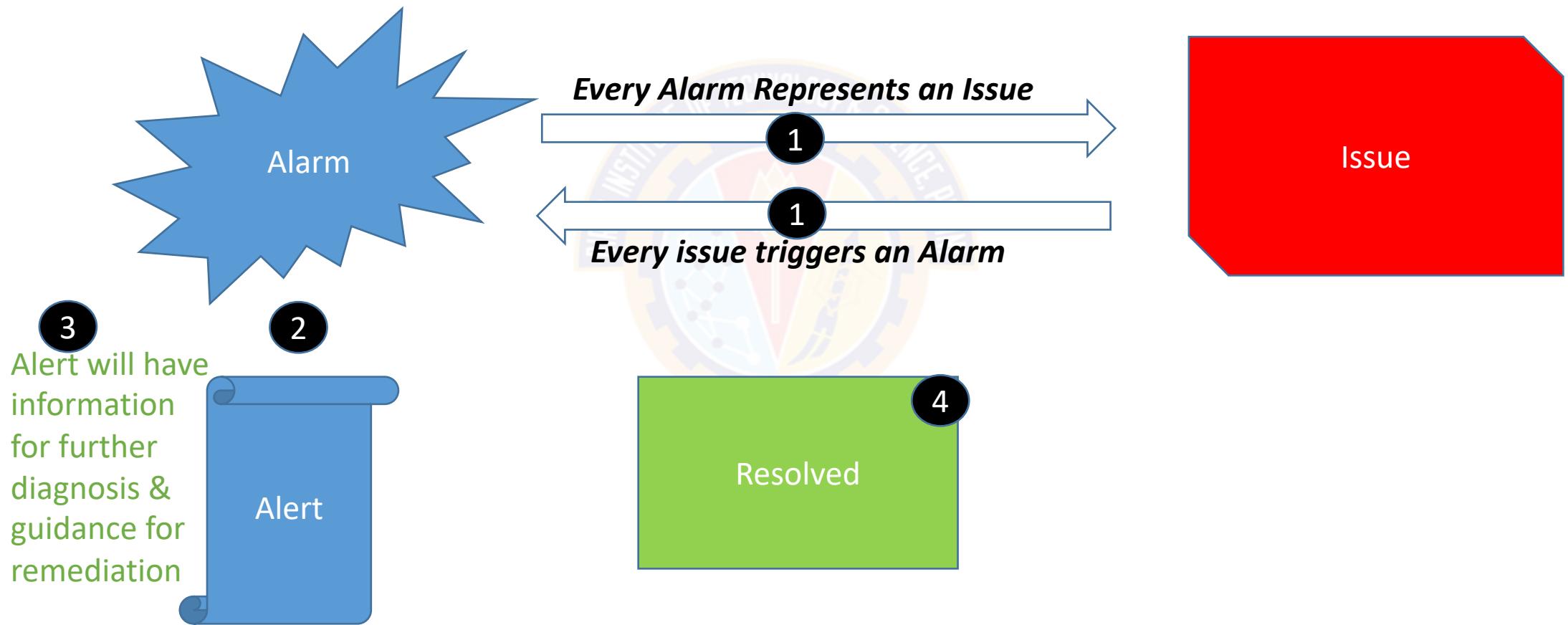
Or

Alarms require action by another system

Example: The datacenter is on fire

# Monitoring Operation Activities

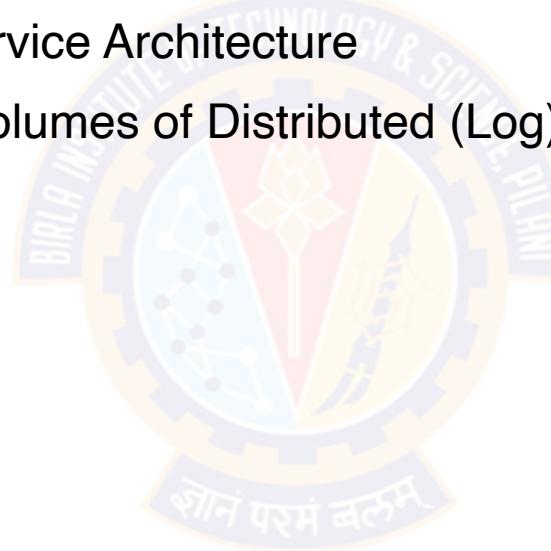
## Alarms and Alerts



# Monitoring

## Challenges in Monitoring by DevOps

- Challenge 1: Monitoring Under Continuous Changes
- Challenge 2: Bottom-Up vs. Top-Down and Monitoring in the Cloud
- Challenge 3: Monitoring a Microservice Architecture
- Challenge 4: Dealing with Large Volumes of Distributed (Log) Data



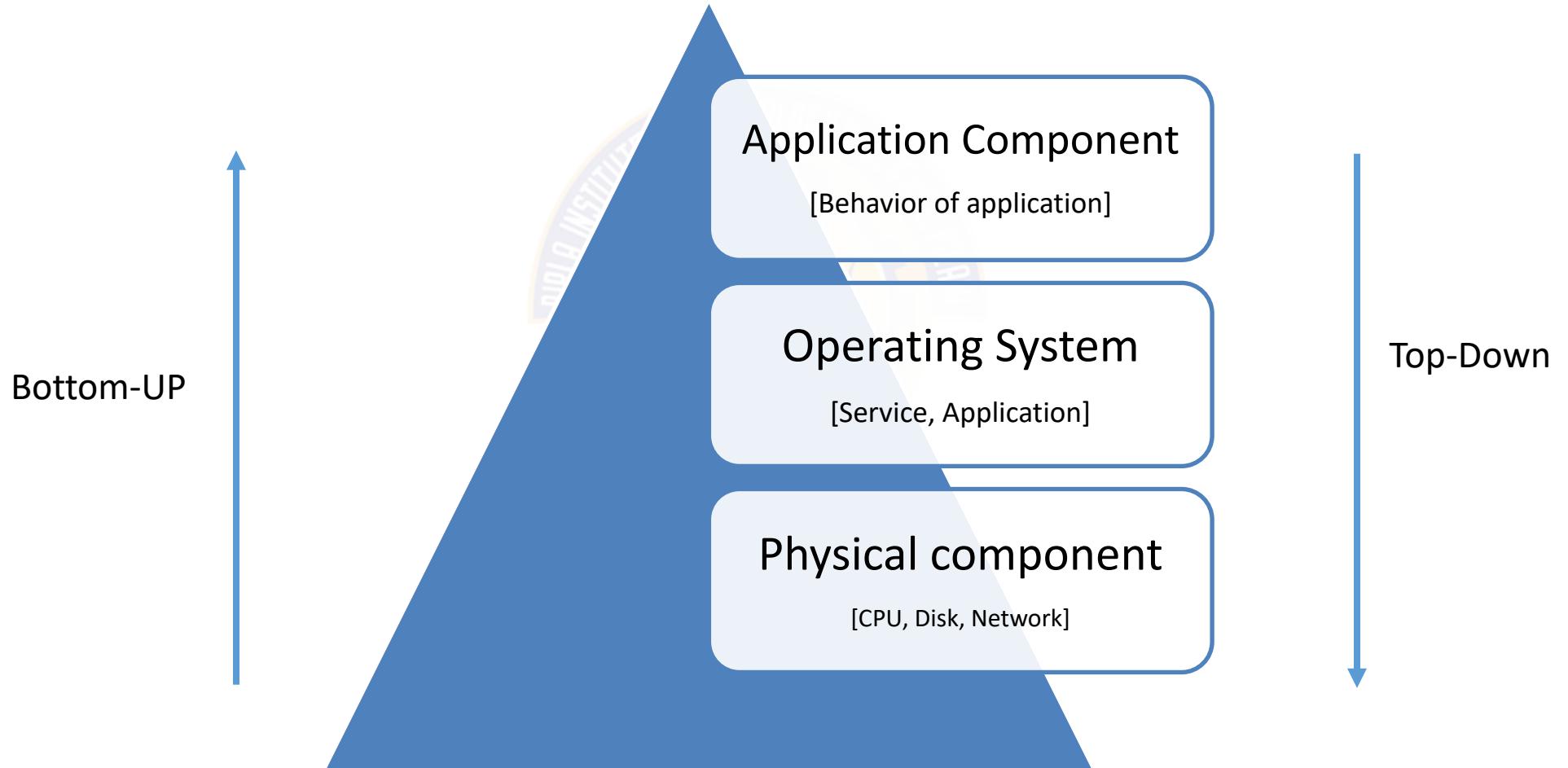
# Challenges in Monitoring

## Challenge 1: Monitoring Under Continuous Changes

- Here the solution is to automate the configuration of alarms, alerts, and thresholds as much as possible; the monitoring configuration process is just another DevOps process that can and should be automated
- Example:
  - When you provision a new server, a part of the job is to register this server in the monitoring system automatically
  - When a server is terminated, a de-registration process should happen automatically
  - For example, the monitoring results during canary testing for a small set of servers can be the new baseline for the full system and populated automatically

# Challenges in Monitoring

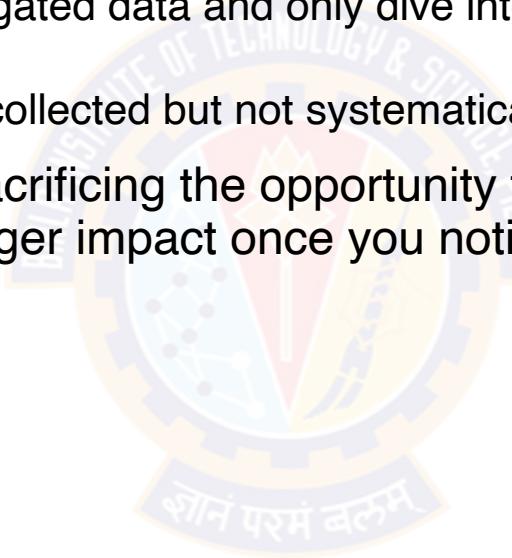
## Challenge 2: Bottom-Up vs. Top-Down and Monitoring in the Cloud



# Challenges in Monitoring

## Challenge 2: Bottom-Up vs. Top-Down and Monitoring in the Cloud

- Adopting a more top-down approach for monitoring cloud-based and highly complex systems is an attempt to solve these problems
  - You monitor the top level or aggregated data and only dive into the lower-level data in a smart way if you notice issues at the top level
  - The lower-level data must still be collected but not systematically monitored for errors
- Risk : By above solution you are sacrificing the opportunity to notice issues earlier; and it might already be too late to prevent a bigger impact once you notice that something is wrong at the top level

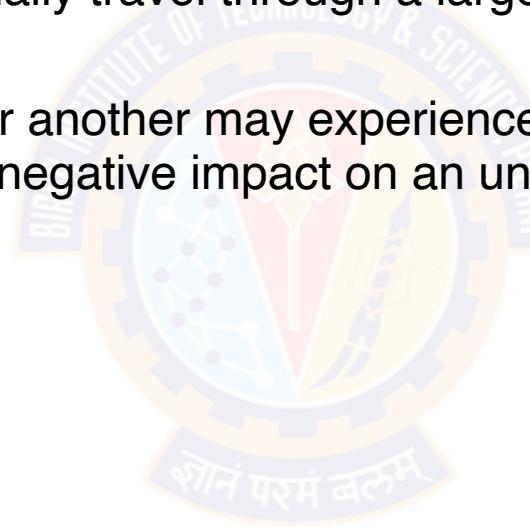


Note: There is no easy solution, bottom-up and top-down monitoring are both important and should be combined in practice

# Challenges in Monitoring

## Challenge 3: Monitoring a Microservice Architecture

- Adoption of a microservice architecture enables having an independent team for each microservice
- Every external request may potentially travel through a large number of internal services before an answer is returned
- In a large-scale system, one part or another may experience some slowdown at any given time, which may consequently lead to a negative impact on an unacceptable portion of the overall requests



Note: Need of intelligent monitor systems; one can monitor at microservice level

# Challenges in Monitoring

## Challenge 4: Dealing with Large Volumes of Distributed (Log) Data

- Operators should use varied and changeable intervals rather than fixed ones, depending on the current situation of the system
  - If there are initial signs of an anomaly or when a periodic operation is starting, set finer-grained monitoring
  - Return to bigger time intervals when the situation is resolved or the operation completed
- Use a modern distributed logging or messaging system for data collection
  - A distributed logging system such as Logstash can collect all kinds of logs and conduct a lot of local processing before shipping the data off
  - This type of system allows you to reduce performance overhead, remove noise, and even identify errors locally

# Continuous Monitoring

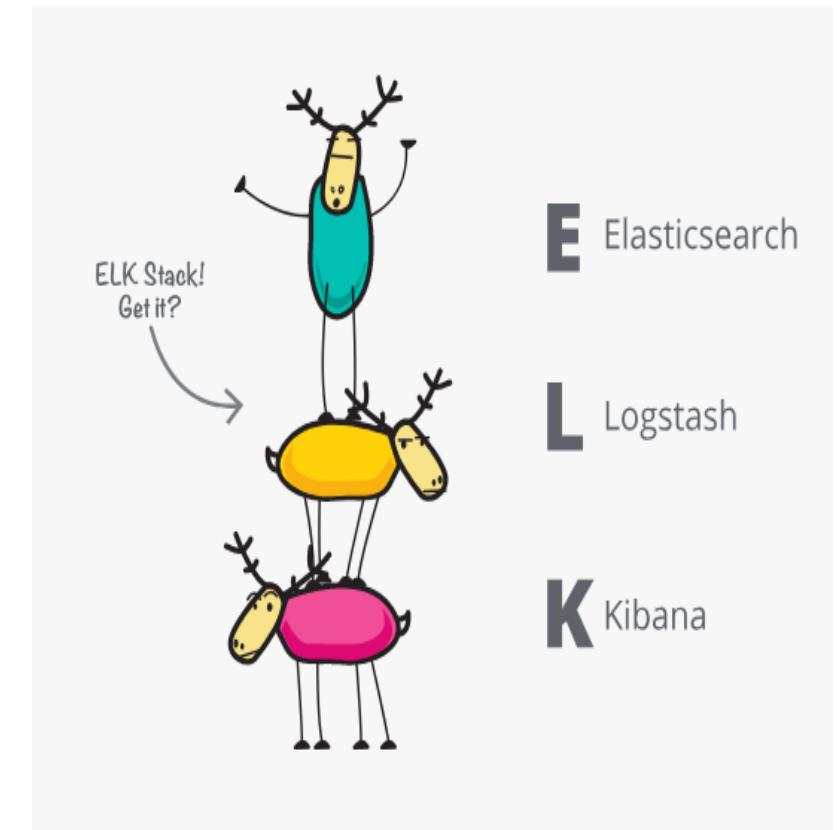
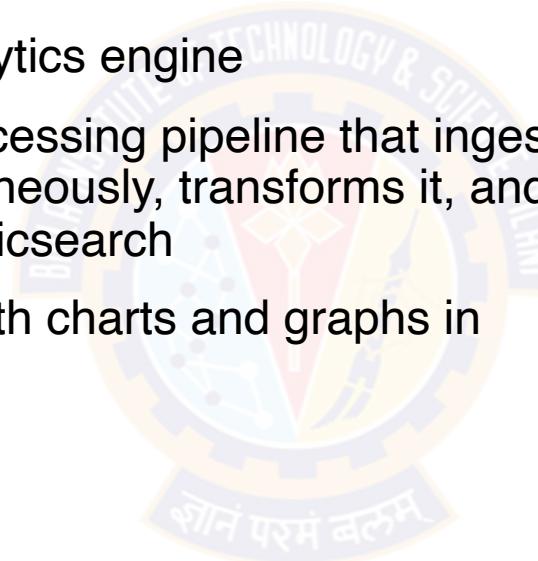
## Monitoring Tools



# Continuous Monitoring

## ELK

- ELK is the acronym for three open source projects
- Elasticsearch, Logstash, and Kibana
- Elasticsearch is a search and analytics engine
- Logstash is a server-side data processing pipeline that ingests data from multiple sources simultaneously, transforms it, and then sends it to a "stash" like Elasticsearch
- Kibana lets users visualize data with charts and graphs in Elasticsearch

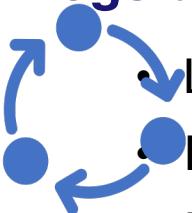


## Elasticsearch

- ELK is started with Elasticsearch
- The open source, distributed, RESTful, JSON-based [Key Pair value] search engine
- Easy to use, scalable and flexible, it earned hyper-popularity among users and a company formed around it
- Elasticsearch lets you perform and combine many types of searches — structured, unstructured, geo, metric
- Elasticsearch aggregations let you zoom out to explore trends and patterns in your data
- Elasticsearch is a NoSQL database that is based on the Lucene search engine



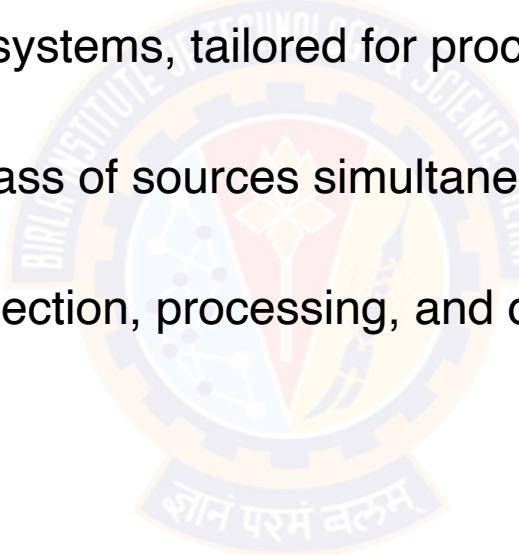
## Logstash



Logstash is an open source

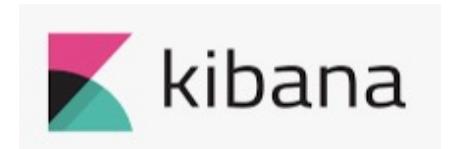
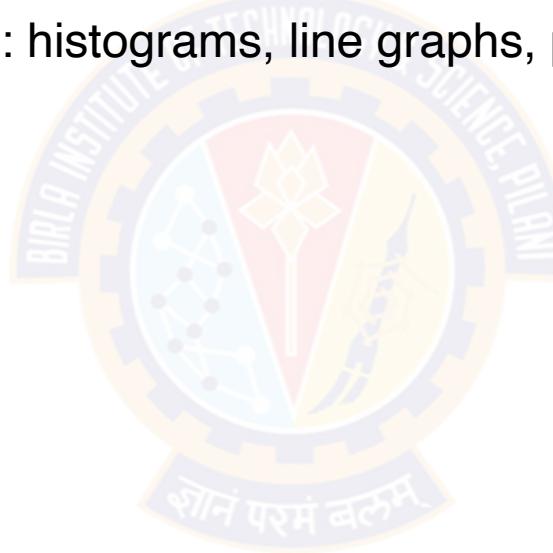
It is a server-side data processing

- It is a distributed log management systems, tailored for processing large amounts of text-based logs
- Logstash consumes data from a mass of sources simultaneously, transforms it, and then sends it to your favorite “stash”
- Logstash works in three stages collection, processing, and dispatching



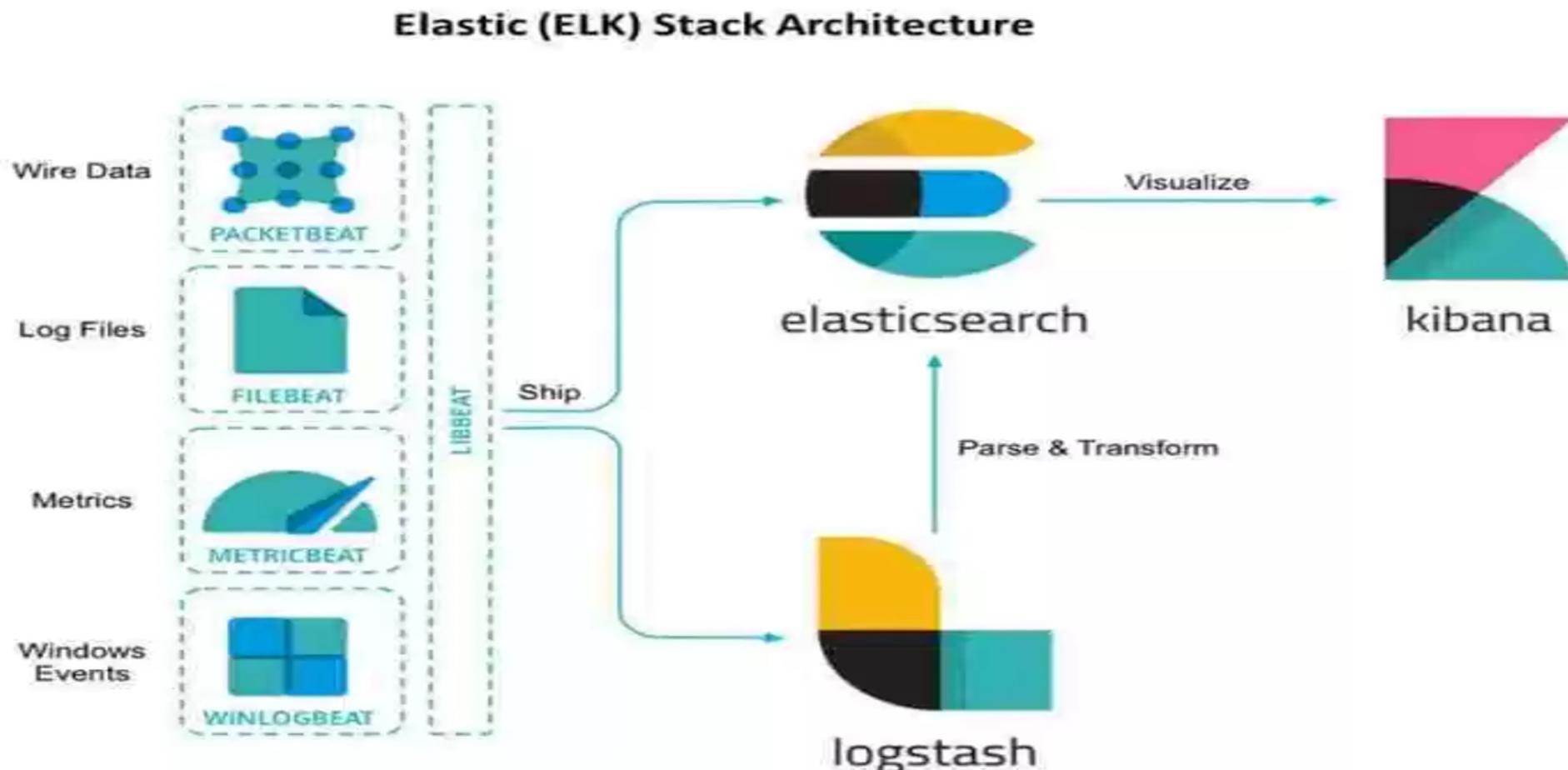
## Kibana

- Kibana lets you visualize your Elasticsearch data and navigate the Elastic Stack
- Kibana gives you the freedom to select the way you give shape to your data
- Kibana core ships with the classics: histograms, line graphs, pie charts, sunbursts, and more



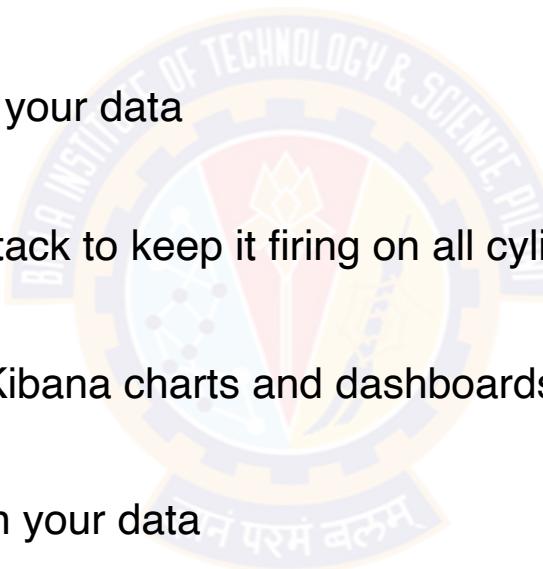
# ELK

## Architecture

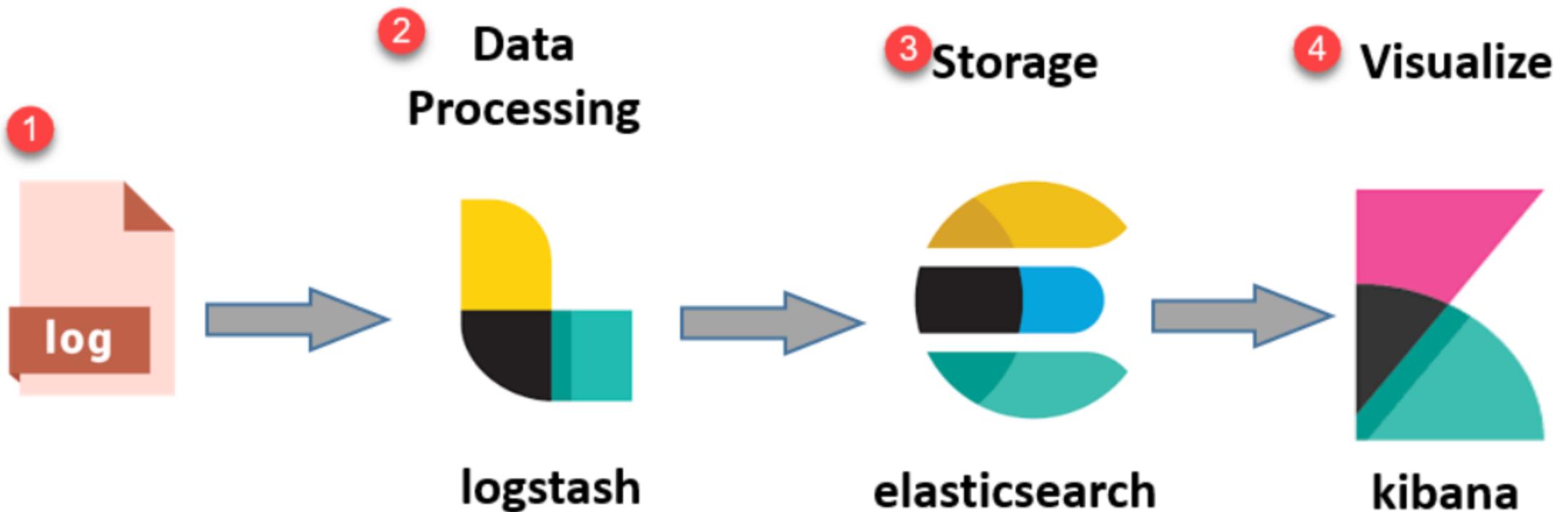


## Features & Benefits

- Security:
  - Protect your Elasticsearch data in a robust and granular way
- Alerting:
  - Get notifications about changes in your data
- Monitoring:
  - Maintain a pulse on your Elastic Stack to keep it firing on all cylinders
- Reporting:
  - Create and share reports of your Kibana charts and dashboards
- Graph:
  - Explore meaningful relationships in your data
- Machine Learning:
  - Automate anomaly detection on your Elasticsearch data



## Quick Review



# References

## Text Book Mapping

- Text Book 1: DevOps: A Software Architect's Perspective (SEI Series in Software Engineering) by Len Bass, Ingo Weber, Liming Zhu , Publisher: Addison Wesley (18 May 2015) : Chapter 7 : Monitoring





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

## Configuration Management

- Introduction to Configuration [Infrastructure Concept]
- Managing Application Configuration
- Managing Environments
- Infrastructure as code
- Server Provisioning
- Automating and Managing Server Provisioning
- Configuration management tools- Chef, Puppet
- Managing on-demand infrastructure
- Auto Scaling



# Configuration Management



## Introduction to Configuration

- Started in the 1950s by the United States Department of Defense as a technical management discipline

Process of establishing and maintaining the consistency of something's functional and physical attributes as well as performance throughout its lifecycle

- Configuration management refers to the process by which all artifacts relevant to your project, and the relationships between them, are stored, retrieved, uniquely identified, and modified***

This includes the policies, processes, documentation, and tools required to implement this system of consistent performance, functionality, and attributes

# Introduction to Configuration Management

## Identify your CM strategy



Can I ***exactly reproduce any of my environments***, including the version of the operating system, its patch level, the network configuration, the software stack, the applications deployed into it, and their configuration?

Can I ***easily make an incremental change*** to any of these individual items and ***deploy the change to any, and all, of my environments?***

Can I easily see each change that occurred to a particular environment and trace it back to see exactly what the change was, who made it, and when they made it?

Is it easy for every member of the team to get the information they need, and to make the changes they need to make?

# Introduction to Configuration Management

## Managing Software Configuration

### Configuration information

Used to change the behavior of software at build time, deploy time, and run time

Delivery teams need to consider carefully:  
What configuration options should be available

How to manage them throughout the application's life,

How to ensure that configuration is managed consistently across components, applications, and technologies



**Treat configuration of your system in the same way you treat your code**

# Introduction to Configuration Management

## Facts and Myth

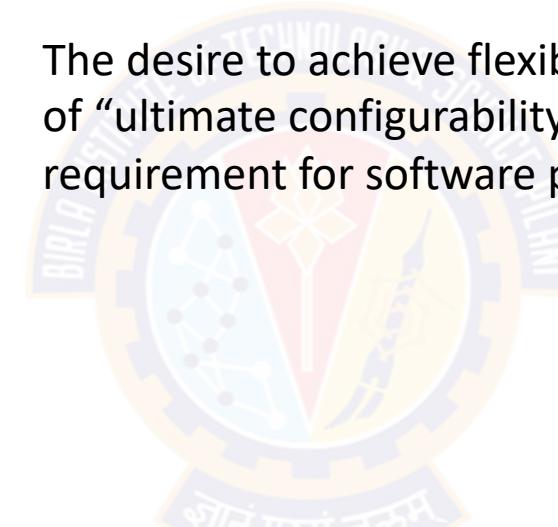


### Flexibility & Configuration

Everyone wants flexible software

Flexibility usually comes at a cost

The desire to achieve flexibility may lead to the common antipattern of “ultimate configurability” which is, all too frequently, stated as a requirement for software projects



### Myth



Configuration information is somehow less risky to change than source code

# Introduction to Configuration Management

## Configuration Vs Source Code

One can stop system easily by changing the configuration compare to by changing the source code

Source Code

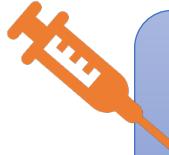
Change the source code  
The compiler will rule out nonsense  
And automated tests should catch most other errors

Configuration information

Change the Configuration  
configuration information is free-form and untested  
Ex: Change in the URL

# Introduction to Configuration Management

## Injecting Configuration information



Configuration information can be injected into application at several points

Build

Deploy

Test

Release

**Generally, we consider it bad practice to inject configuration information at build or packaging time**

**It is usually important to be able to configure your application at deployment time so that you can tell it where the services it depends upon such as database, messaging servers, or external systems belong**

# Manging Configuration

## Level of Configurations

- Application Configuration
- Environment Configuration



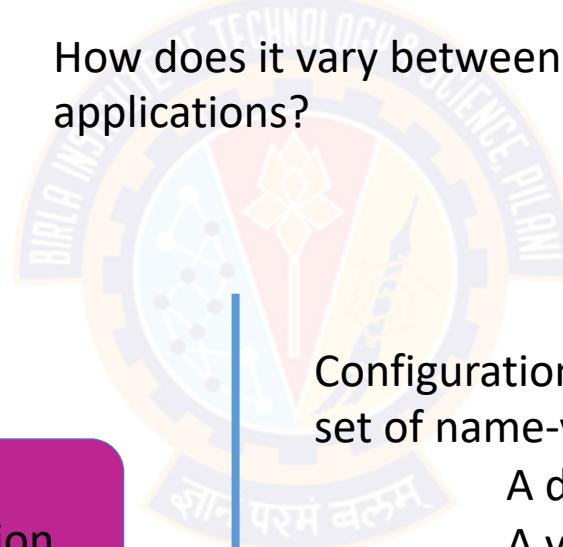
# Manging Application Configuration

## How does it works?

Consideration

Considerable Solution

How do you represent your configuration information?  
How do your deployment scripts access it?  
How does it vary between environments, applications, and versions of applications?

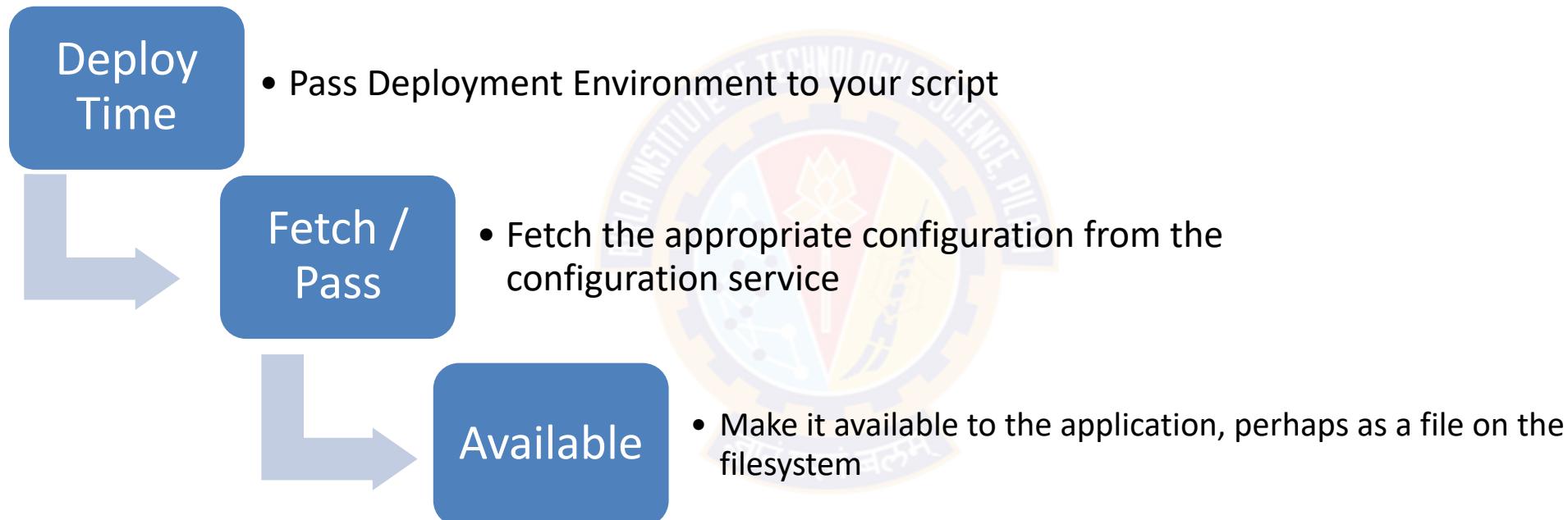


Configuration information is often modeled as a set of name-value strings  
A database  
A version control system  
Or A directory or registry

**Note: Version control is the easiest, just check in configuration file, and get the history of your configuration over time for free also It is worth keeping a list of the available configuration options for application in the same repository as its source code**

# Manging Application Configuration

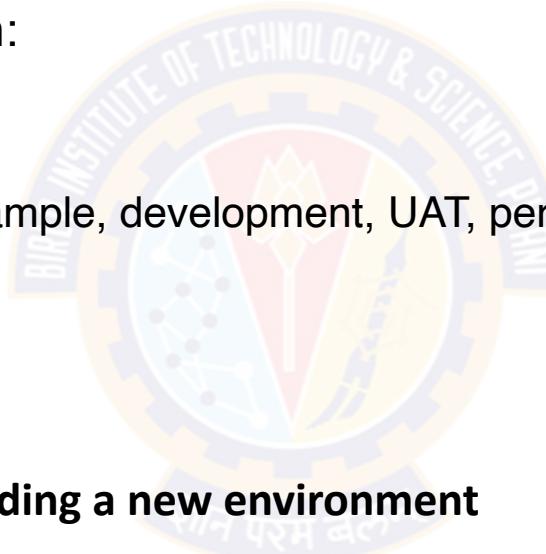
## How does it Works? Cont..



# Manging Application Configuration

## Modeling Configuration

- Each configuration setting can be modeled as a tuple, so the configuration for an application consists of a set of tuples
- Tuple value typically depends upon:
  - The application
  - The version of the application
  - The environment it runs in (for example, development, UAT, performance, staging, or production)



Use cases when modeling configuration information

**Adding a new environment**

**Creating a new version of the application**

**Promoting a new version of your application from one environment to another**

**Relocating a database server**

# Manging Application Configuration

## Principles of Managing Application Configuration

- Keep the available configuration options for your application in the same repository as its source code, but keep the values somewhere else
- Configuration settings have a lifecycle completely different from that of code, while passwords and other sensitive information should not be checked in to version control at all
- Configuration should always be performed by automated processes using values taken from configuration repository, so that one can always identify the configuration of every application in every environment
- Test your configuration scripts
- Use clear naming conventions for configuration options
- Ensure that configuration information is modular and encapsulated
  - So that changes in one place don't have knock-on effects for other, apparently unrelated, pieces of configuration

# Manging Application Configuration

## Testing Configuration

- **There are two parts to testing configuration:**

The first stage is to ensure that references to external services in configuration settings are good

Example: As part of deployment script, ensure that the messaging bus are configured to use is actually up and running at the address configured

Deployment or installation script should fail if anything application depends on is not available; this acts as a great smoke test for configuration settings

The second stage is to actually run some smoke tests once application is installed to make sure it is operating as expected

Example: This should involve just a few tests exercising functionality that depends on the configuration settings being correct

# Managing Environments

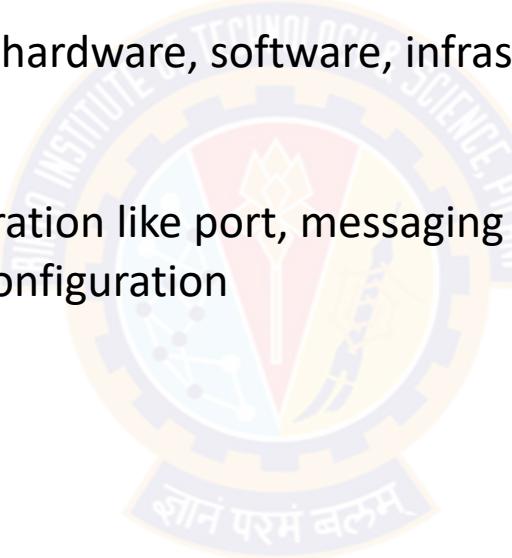
## Environments ?

No application is Island

Every application depends on hardware, software, infrastructure, and external systems in order to work

Any application depends on:

- Application Configuration like port, messaging bus etc.
- Operating System Configuration



# Managing Environments

## Problems in Managing Configuration



- If the **collection of configuration** information is **very large**
- One **small change** can break the **whole application** or severely degrade its performance
- Once it is broken, finding the problem and **fixing it** takes an **unknown amount of time** and requires senior personnel
- It is **extremely difficult** to precisely reproduce **manually configured environments** for testing purposes
- It is **difficult** to maintain such environments without the configuration, and hence behavior, of different nodes drifting apart

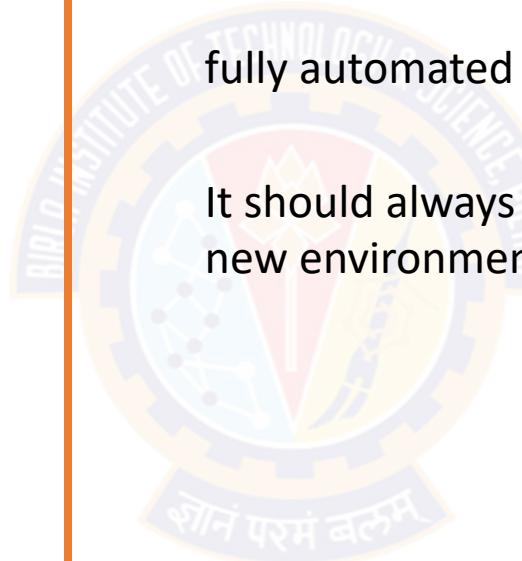
# Managing Environments

## How to Manage Environments

Best Way

fully automated Process

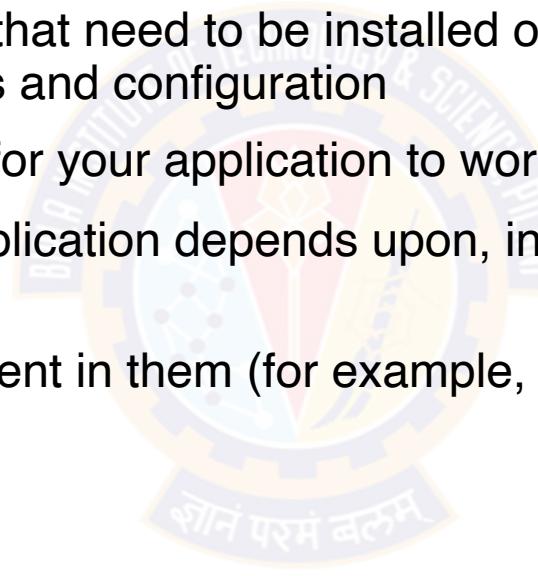
It should always be cheaper to create a new environment than to repair an old one



# Managing Environments

## The kinds of environment configuration information should be concerned

- The various operating systems in environment, including their versions, patch levels, and configuration settings
- The additional software packages that need to be installed on each environment to support application, including their versions and configuration
- The networking topology required for your application to work
- Any external services that your application depends upon, including their versions and configuration
- Any data or other state that is present in them (for example, production databases)



**Note:** There are two principles of an effective configuration management strategy: Keep binary files independent from configuration information, and keep all configuration information in one place

# Infrastructure as Code [IaC]

## Introduction



IaC

Method of writing and deploying machine-readable definition files that generate service components

IaC helps IT operations teams manage and provision IT infrastructure automatically through code without relying on manual processes

IaC is often described as “programmable infrastructure”

# Infrastructure as Code [IaC]

## IaC for DevOps



- In generic software development, a fundamental constraint is the need for the environment
  - i.e. Testing or Staging environment = Live Environment

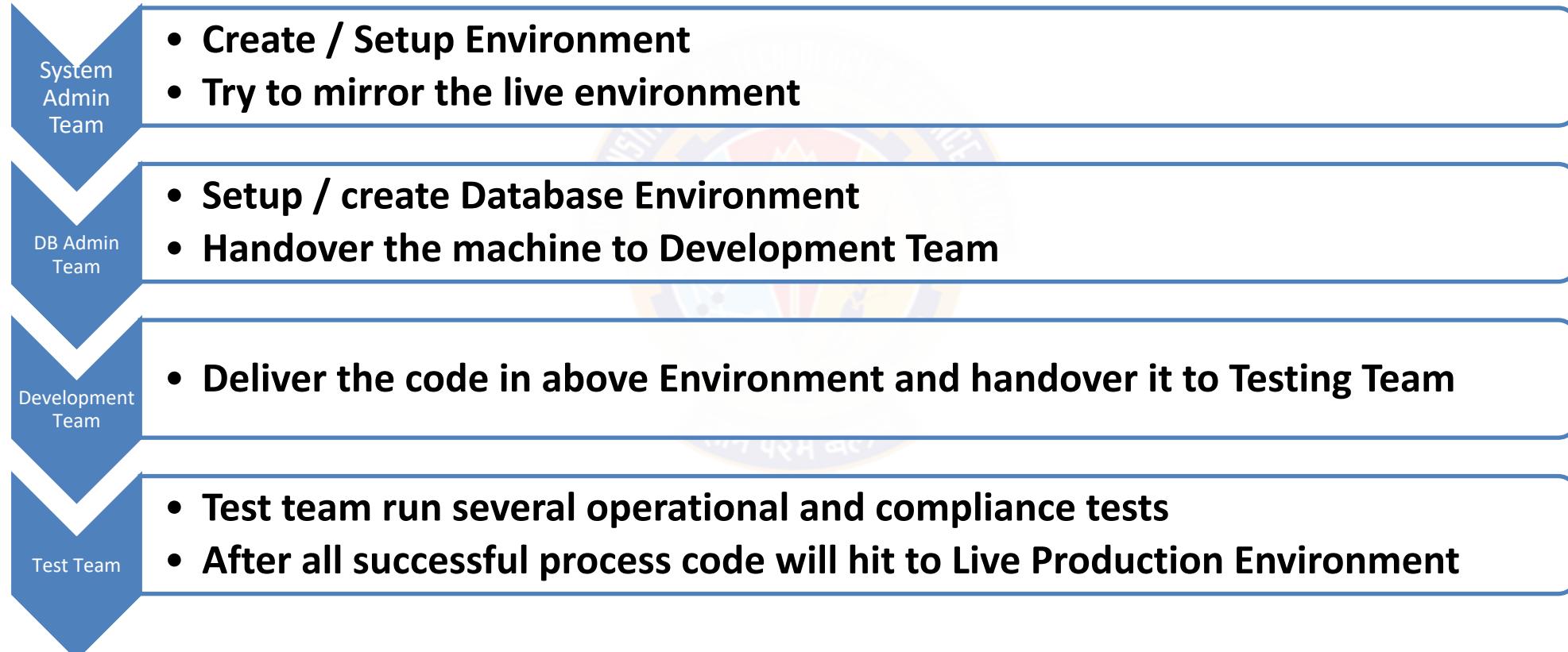


### Note:

This is the only way of assuring that the new code will not crash with existing code definitions – generating errors or conflicts that may compromise the entire system

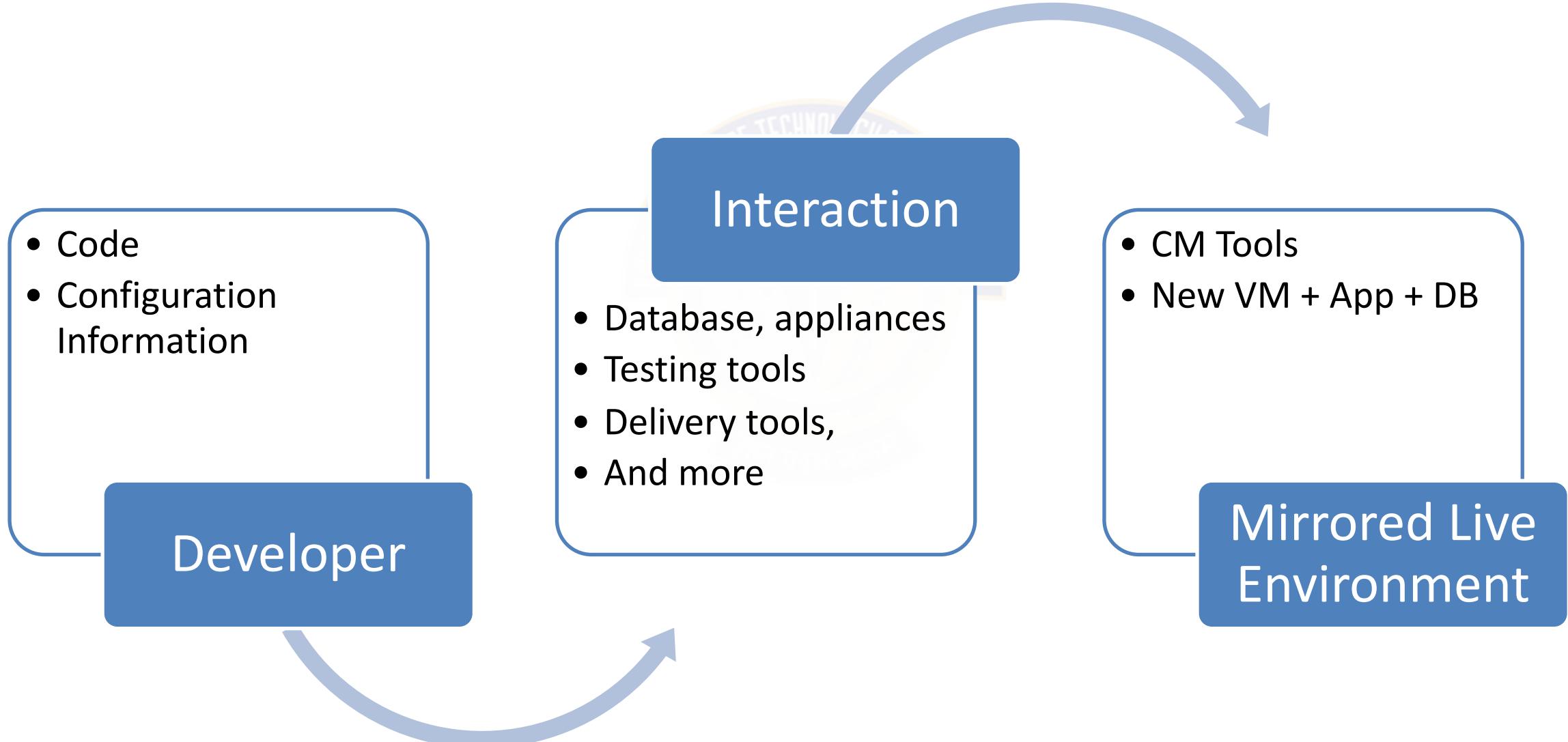
# Infrastructure as Code [IaC]

## Before IaC and DevOps?



# Infrastructure as Code [IaC]

What is achieved with the help of DevOps and IaC?



# Infrastructure as Code [IaC]

## Benefits of IaC Summary



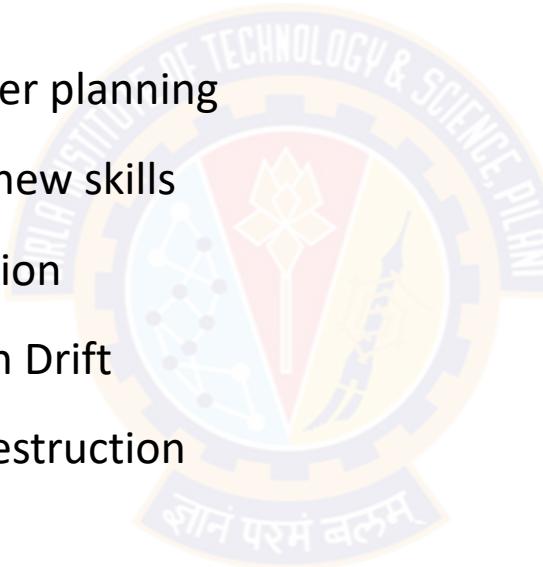
- Reducing Shadow IT
- Improving Customer Satisfaction
- Operating Expenses [OPEX]
- Capital Expenditure [CAPEX]
- Standardization
- Safer Change Management

# Infrastructure as Code [IaC]

## Risks involved with IaC



- Missing proper planning
- IaC requires new skills
- Error replication
- Configuration Drift
- Accidental Destruction



# Provisioning Servers

## What is Provisioning Servers

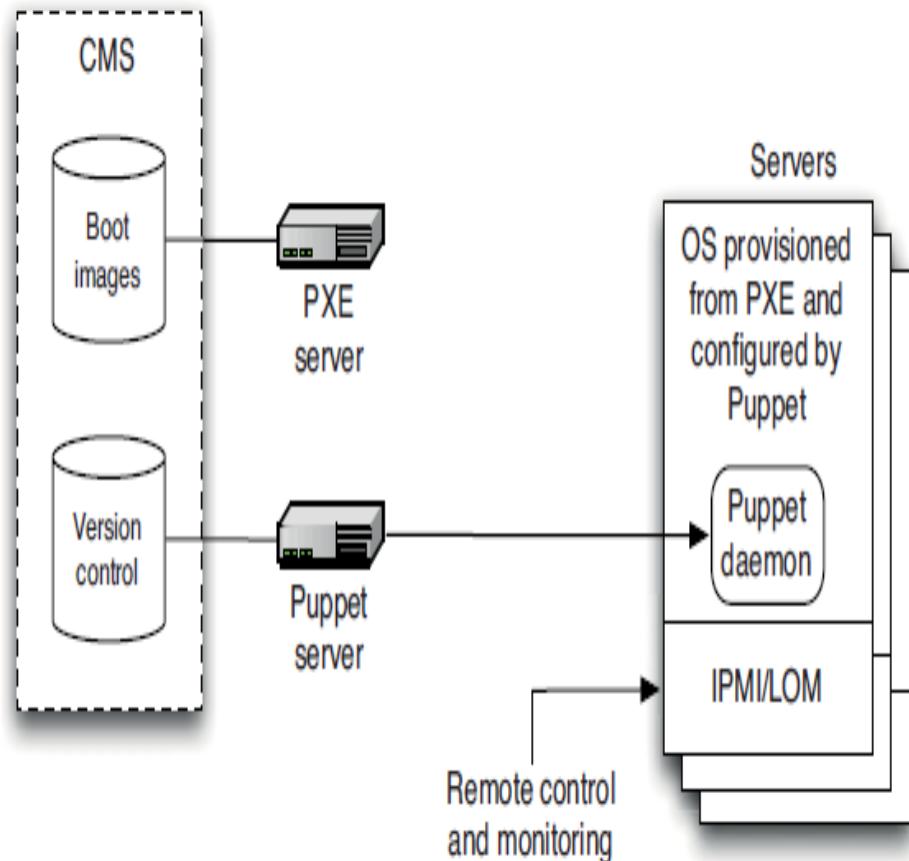
Set Up  
Hardware

Install &  
Configure

Install &  
Configure  
Application

# Provisioning Servers

## An Automate Approach to Provisioning



DHCP

*Dynamic Host Configuration Protocol*

- Client request for IP
- Respond to the Request

PXE

*Preboot Execution Environment*

- Request for Boot Image Info
- Boot Image File Name

TFTP

*Trivial File Transfer Protocol*

- Request file of Boot Image
- Boot Image File to the client

# An Automate Approach to Provisioning

## Benefits

- Reduction to Man Hours for installation and configuration
- Reduce Operational Cost
- Reduces Errors
- Better Quality Assurance



# Configuration management in DevOps

## Clear thought

### Configuration Management

Configuration management deals with the state of any given infrastructure or software system at any given time

VS

### Change Management

Change Management deals with how changes are made to those configurations



# Configuration management in DevOps

## Traditional Approach

Infrastructure as Document

Configuration as Document

# Configuration management in DevOps

## New Approach

Infrastructure as code : “It is defining the entire environment definition as a code or a script instead of recording in a formal document”

Configuration as code : “It is nothing but defining all the configurations of the servers or any other resources as a code or script and checking them into version control”

# Configuration management in DevOps

## Tools

- Puppet
- Chef
- Ansible
- Terraform
- Cloud formation etc.

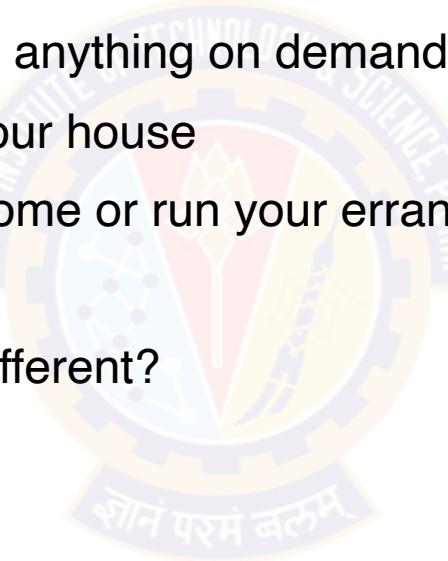


# Managing on-demand infrastructure

## Why



- We live in a society where you can get anything on demand, right from your smartphone
- You can have groceries delivered to your house
- You can hire someone to clean your home or run your errands
- And ride-sharing services like Uber
- So why should infrastructure be any different?

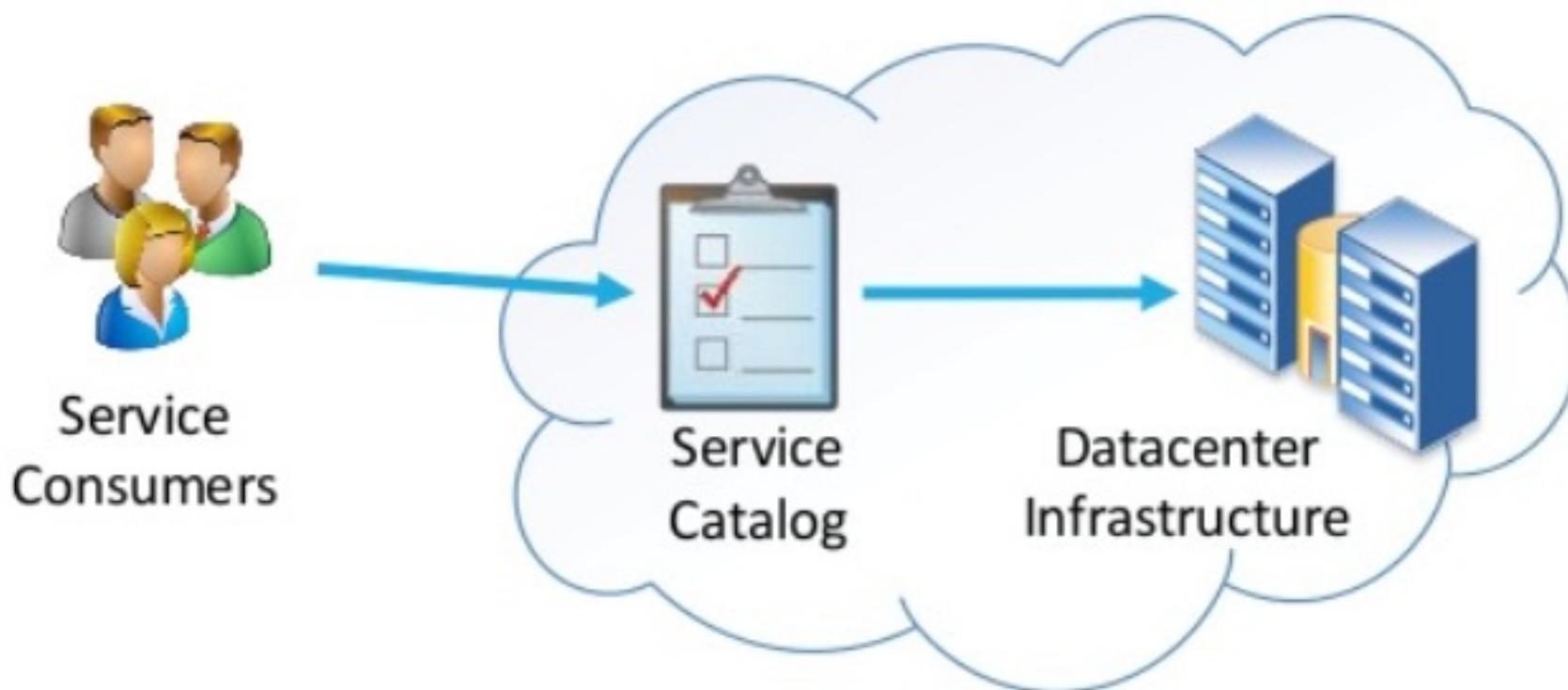


# Managing on-demand infrastructure

## Cloud Provider



### Infrastructure On Demand



# Managing on-demand infrastructure

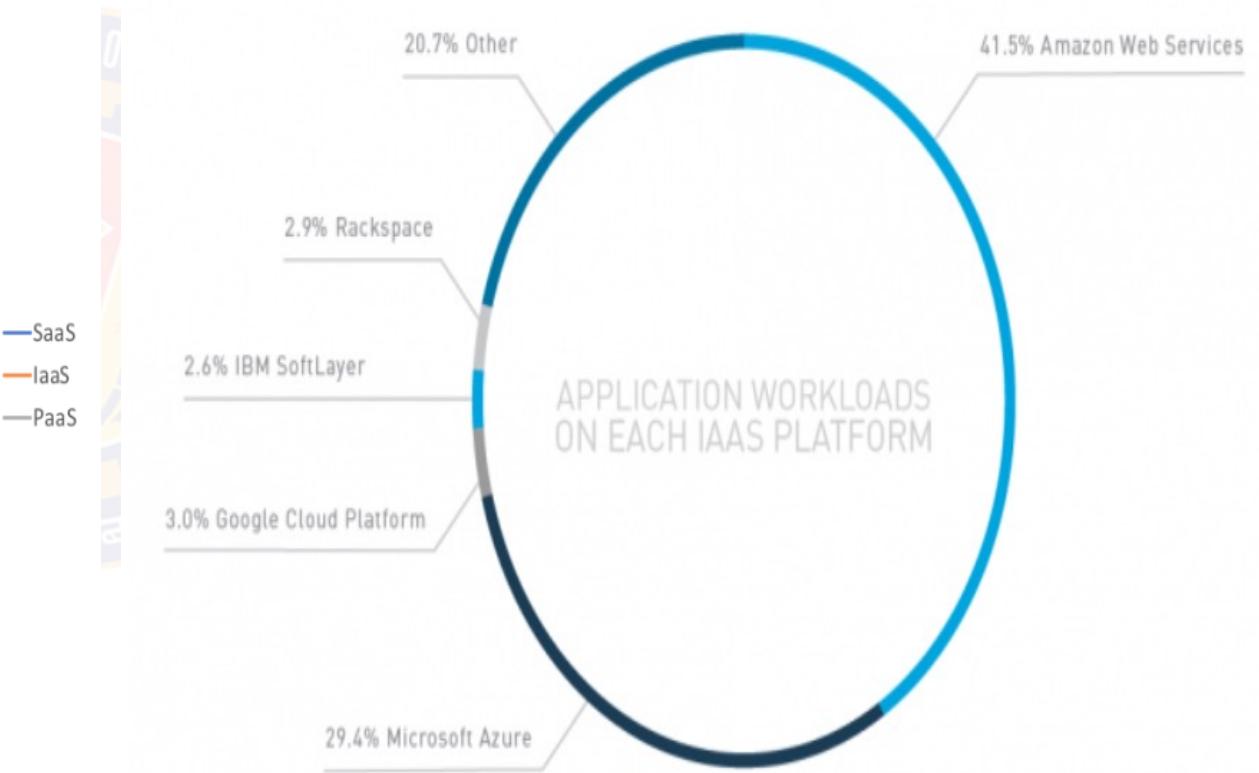
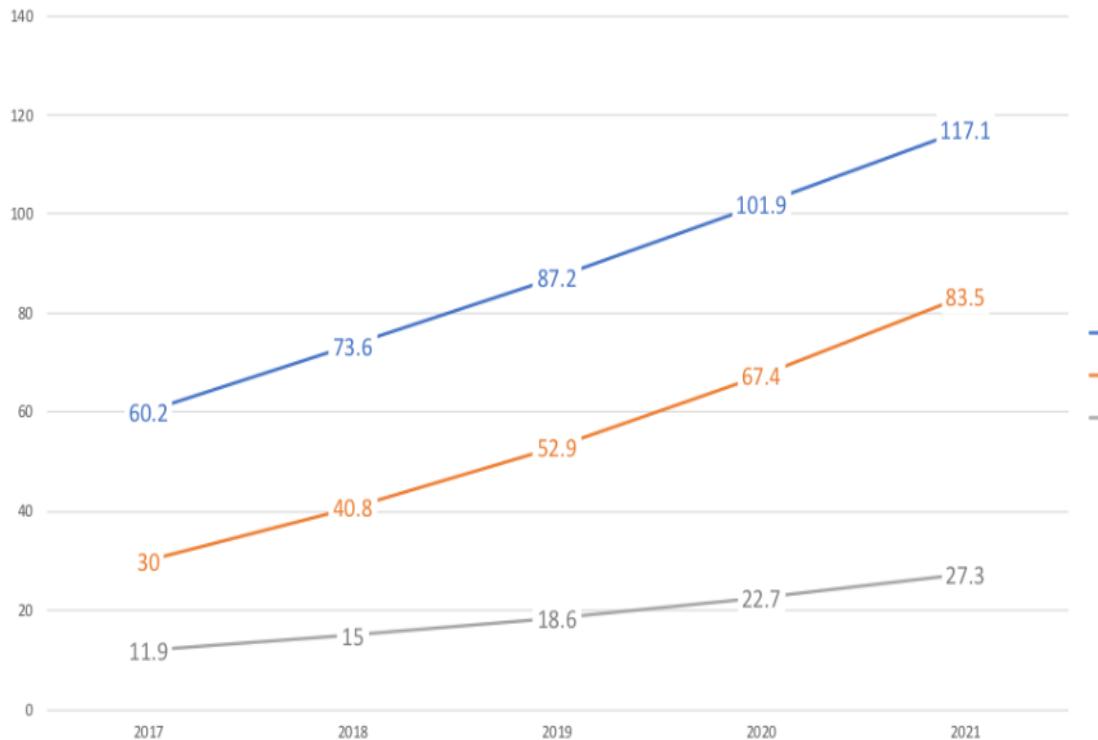
## Cloud Companies



# Managing on-demand infrastructure

## Stats

CLOUD MARKET REVENUE IN BILLIONS OF DOLLARS



# Managing on-demand infrastructure

## Benefits of On-Demand Infrastructure

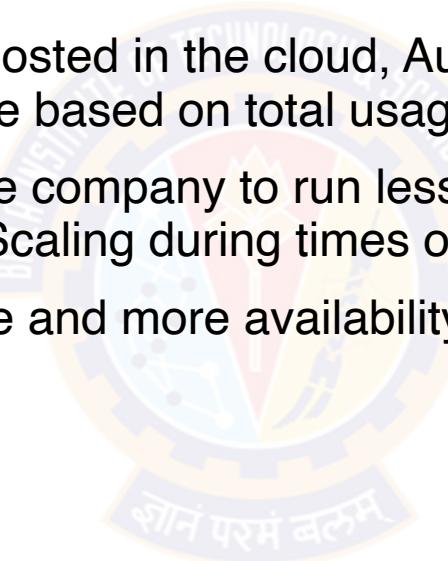
- Cost savings
- Scalability and flexibility
- Faster time to market
- Support for DR and high availability
- Focus on business growth



# Auto Scaling

## Auto Scaling Offers

- Auto Scaling typically means allowing some servers to go to sleep during times of low load
- Saving on Power and Energy
- For companies using infrastructure hosted in the cloud, Auto Scaling can mean lower bills, because most cloud providers charge based on total usage rather than maximum capacity
- Auto Scaling can help by allowing the company to run less time-sensitive workloads on machines that get freed up by Auto Scaling during times of low traffic
- Auto Scaling can offer greater uptime and more availability in cases where production workloads are variable and unpredictable



# Auto Scaling

## Approaches

### Scheduled Auto Scaling Approach

This is an approach to Auto Scaling where changes are made to the minimum size, maximum size, or desired capacity of the Auto Scaling group at specific times of day

E.g. E-commerce sites: Flipkart Big Billion Day, Amazon's Great India Sale

### Predictive Auto Scaling

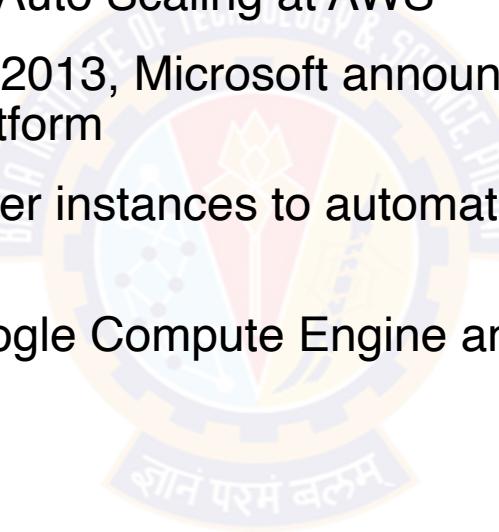
The idea is to combine recent usage trends with historical usage data as well as other kinds of data to predict usage in the future, and Auto Scale based on these predictions

E.g. Online Social and Media Hosting

# Auto Scaling

## Who Offers Auto Scaling?

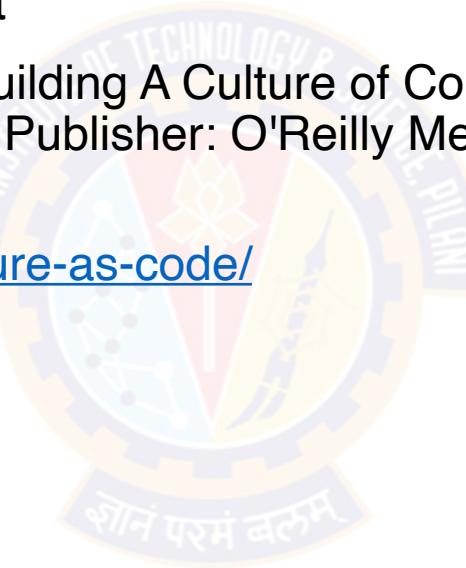
- Amazon Web Services (AWS) : In 2009, Amazon launched its own Auto Scaling feature along with Elastic Load Balancing
- Netflix is the well know consumer of Auto Scaling at AWS
- Microsoft's Windows Azure : Around 2013, Microsoft announced that Auto Scaling support to its Windows Azure cloud computing platform
- Oracle Cloud Platform: It allows server instances to automatically scale a cluster in or out by defining an Auto Scaling rule
- Google Cloud Platform : In 2015 Google Compute Engine announced a public beta of its Auto Scaling feature for use



# References

## Text Book Mapping

- Text Book 2: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Publisher: Addison Wesley, 2011: Chapter 2 : Configuration Management
- Reference Book1:Effective DevOps: Building A Culture of Collaboration, Affinity, and Tooling at Scale by Jennifer Davis , Ryn Daniels. Publisher: O'Reilly Media, June 2016: Chapter 4 Foundation Terminology and concepts
- <https://www.bmc.com/blogs/infrastructure-as-code/>





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Review CS#9

## Configuration Management

- Introduction to Configuration [Infrastructure Concept]
- Managing Application Configuration
- Managing Environments
- Infrastructure as code
- Server Provisioning
- Automating and Managing Server Provisioning
- Configuration management tools- Chef, Puppet
- Managing on-demand infrastructure
- Auto Scaling



# Agenda

## Configuration Management Tools and Virtualization & Containerization

- CM – Tools
  - Chef Configuration Management Tool
  - Puppet & Puppet Enterprise
  - Ansible
- Pre Virtualization World
- Virtualization
- Hypervisors Vs Containerization
- Docker



# Configuration Management Tools

## Configuration Management Tool Provides

What they Offer in common

- Quick Provisioning of New Servers
- Quick Recovery from Critical Events
- No More Snowflake Servers [Complex or Unique Servers]
- Version Control for the Server Environment
- Easy Replication of Environments

Offer in common for Servers

- Automation Framework
- Idempotent Behavior
- Templating System
- Extensibility



# Configuration Management Tool

## Chef

### Why

Automate the infrastructure provisioning  
Infrastructure as Code  
Configuration as Code  
Ruby DSL language  
Solution is compatible with Physical, Virtual, and Cloud Machines  
Capability to get integrated with any of the cloud technology  
Open Source

### Who

Facebook  
Etsy  
Cheezburger  
Indiegogo



# Chef

## Chef Components

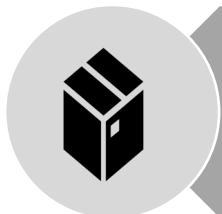


### Chef Server

A hub for configuration data

Chef server stores cookbooks, the policies that are applied to nodes

And metadata that describes each registered node that is managed by Chef



### Actual Server (Node)

These are the machines that are managed by Chef

The Chef client is installed on each node

It is used to configure the node to its desired state



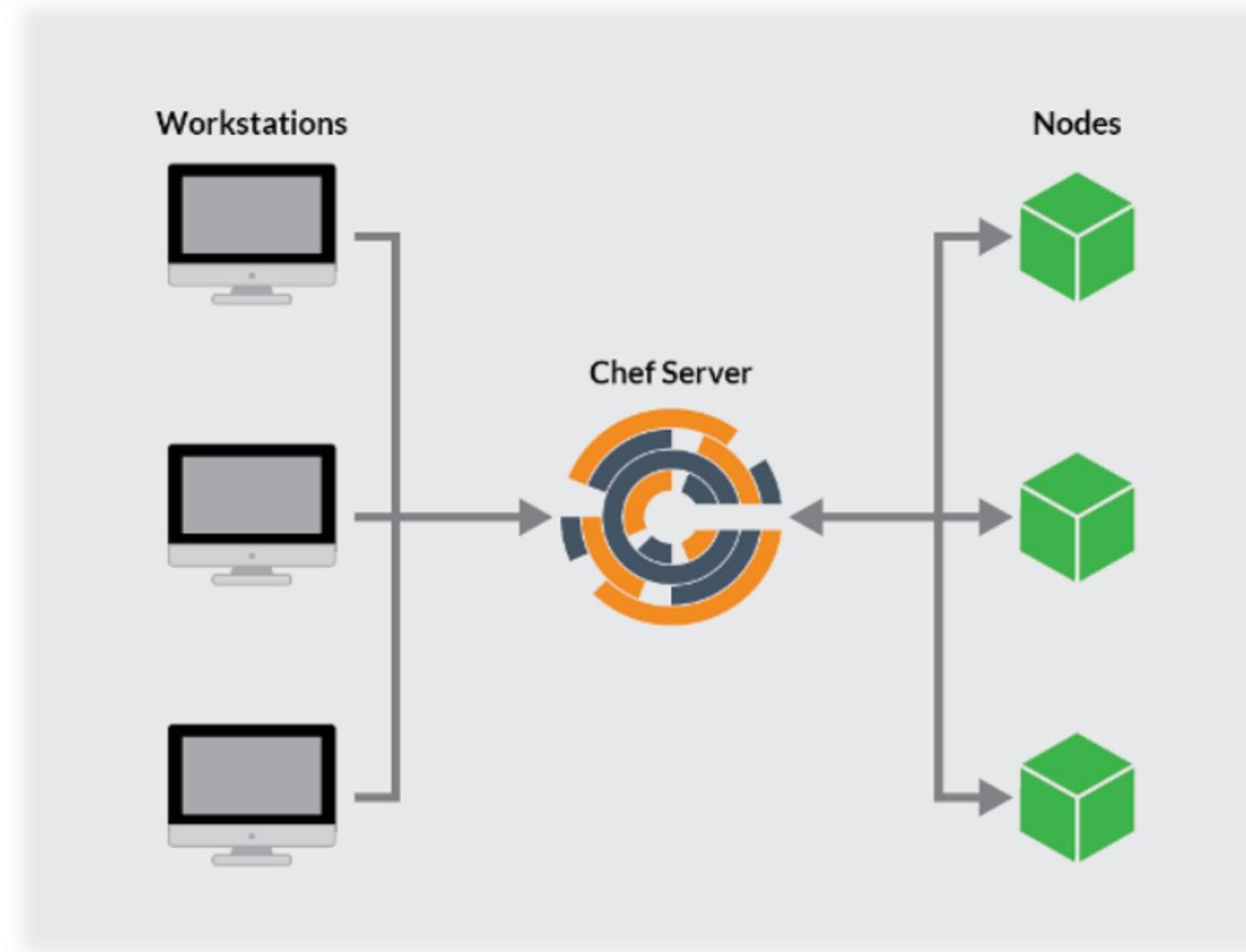
### Chef Workstation

It is the location where users interact with Chef

At Chef Workstation, Users can author and test cookbooks



## Chef Components the Workflow



# Chef Terms

## Cookbooks



Cookbooks

Fundamental unit of configuration and policy distribution

Defines a scenario and contains everything that is required to support that scenario

The chef-client uses Ruby as its reference language for creating cookbooks and defining recipes, with an extended DSL for specific resources

~/chef-repo/cookbooks/lamp\_stack/apache.rb

```
1 package "apache2" do
2   action :install
3 end
```

~/chef-repo/cookbooks/lamp\_stack/apache.rb

```
1 service "apache2" do
2   action [:enable, :start]
3 end
```



# Chef Terms

## Chef Supermarket



It is the location in which community cookbooks are shared and managed

Cookbooks that are part of the Chef Supermarket may be used by any Chef user

How community cookbooks are used varies from organization to organization

## The chef-repo



It is the repository structure in which cookbooks are authored, tested, and maintained and from which policy is uploaded to the Chef server

The chef-repo should be synchronized with a version control system such as Git and then managed



# Chef

## Workstation Components and Tools



- It is a package that contains everything that is needed to start using Chef  
Chef-client, chef and knife command line tools



- Chef is the command-line tool
- Chef is used to work with items in a chef-repo



- knife is also the command-line tool
- It interact with nodes or work with objects on the Chef server



- It is a testing harness for rapid validation of Chef code



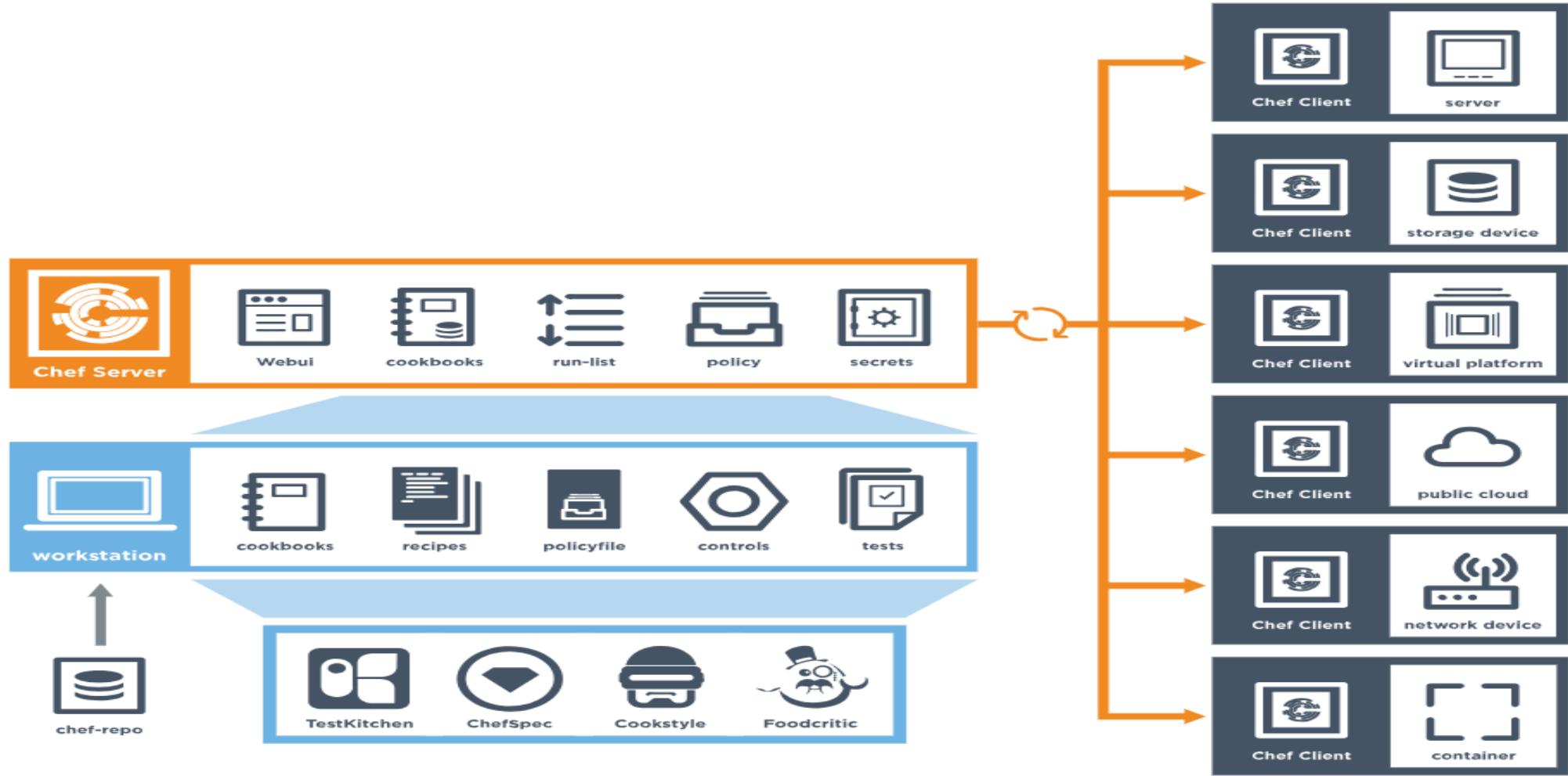
- It is a Chef's open source security & compliance automation framework



- It is a tool for running ad-hoc tasks

# Chef

## Lets Summarize



# Configuration Management Tool

## Puppet

Why

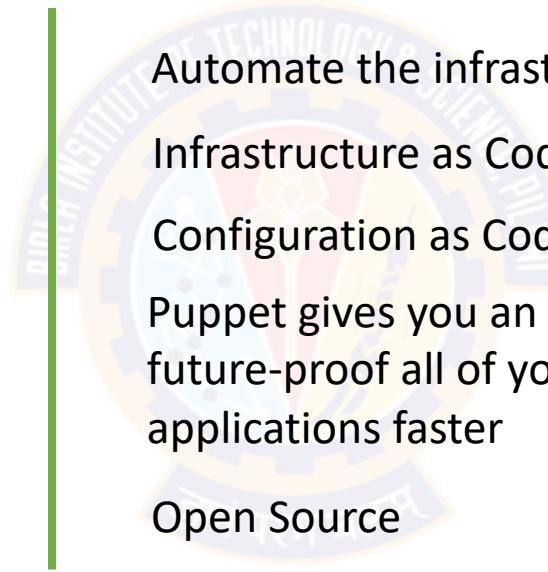
Automate the infrastructure provisioning

Infrastructure as Code

Configuration as Code

Puppet gives you an automatic way to inspect, deliver, operate and future-proof all of your infrastructure and deliver all of your applications faster

Open Source



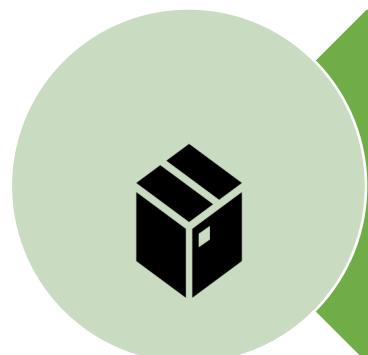
# Puppet

## Puppet Components



### Puppet Master

- Puppet master is a Ruby application
- It compiles configurations for any number of Puppet agent nodes
- Puppet Server is an application that runs on the Java Virtual Machine (JVM)
- Supported platforms:
  - Red Hat Enterprise Linux
  - Fedora
  - Debian
  - and Ubuntu



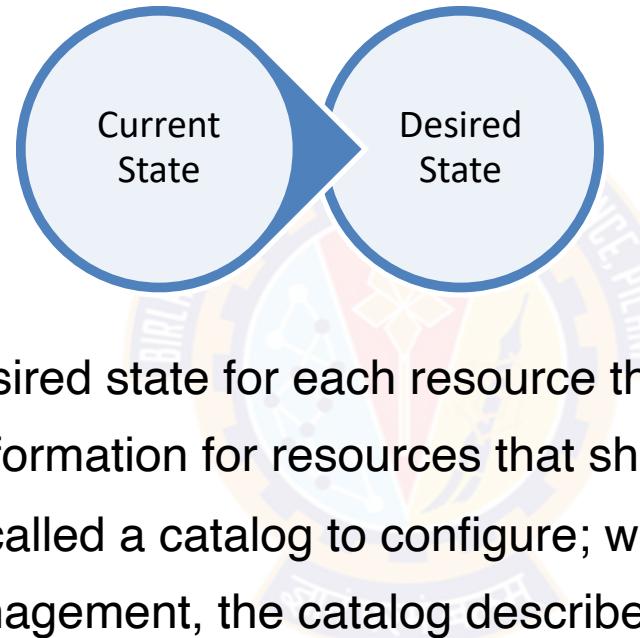
### Puppet Agent

- Managed nodes run the Puppet agent application, as a background service
- Periodically, the agent sends facts to a master and requests a catalog
- The master compiles the catalog using several sources of information, and returns the catalog to the agent



# Puppet Terms

## Catalog



- The catalog describes the desired state for each resource that should be managed
- It also specifies dependency information for resources that should be managed in a certain order
- The agent uses a document called a catalog to configure; which it downloads from master
- For each resource under management, the catalog describes its desired state
- The "puppet apply command" compiles its own catalog

# Puppet Terms

## Factor

- Factor is the cross-platform system profiling library in Puppet
- It discovers and reports per-node facts, which are available in your Puppet manifests as variables
- Before requesting a catalog, the agent uses Factor to collect system information



```
$ facter -p os
{
  architecture => "x86_64",
  family => "RedHat",
  hardware => "x86_64",
  name => "Centos",
  release => {
    full => "6.8",
    major => "6",
    minor => "8"
  },
  selinux => {
    enabled => false
  }
}
```

# Puppet Terms

## Resources

- Puppet code is composed primarily of resource declarations
- A resource describes something about the state of the system
- Such as a certain user or file should exist
- Here is an example of a user resource declaration



### Resource Example:

```
user { sonika':  
    ensure  => present,  
    uid     => '1000',  
    gid     => '1000',  
    shell   => '/bin/bash',  
    home   => '/home/sonika'  
}
```

# Puppet Terms

## Manifests

- Manifests are Puppet programs
- Manifests are composed of puppet code
- And their filenames use the .pp extension
- The default main manifest in Puppet installed via apt is /etc/puppet/manifests/site.pp



## Manifest: Example

```
file { "/var/bits/devops":  
    ensure => "present",  
    owner => "sonika",  
    group => "bits",  
    mode => "664",  
    content => "This is a test file created  
    using puppet.  
    Puppet is really cool",  
}
```

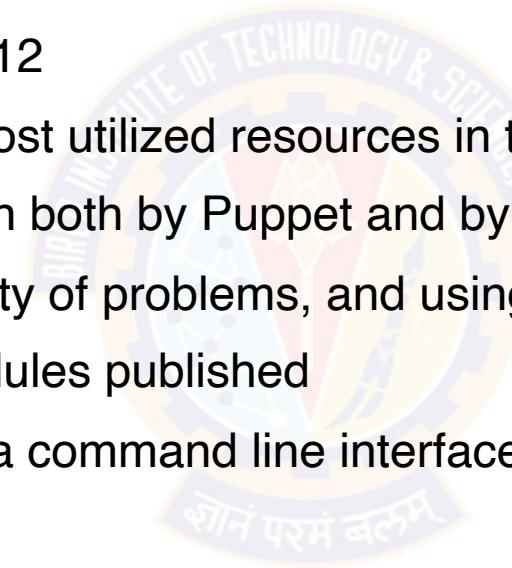


# Puppet Terms

## The Puppet Forge

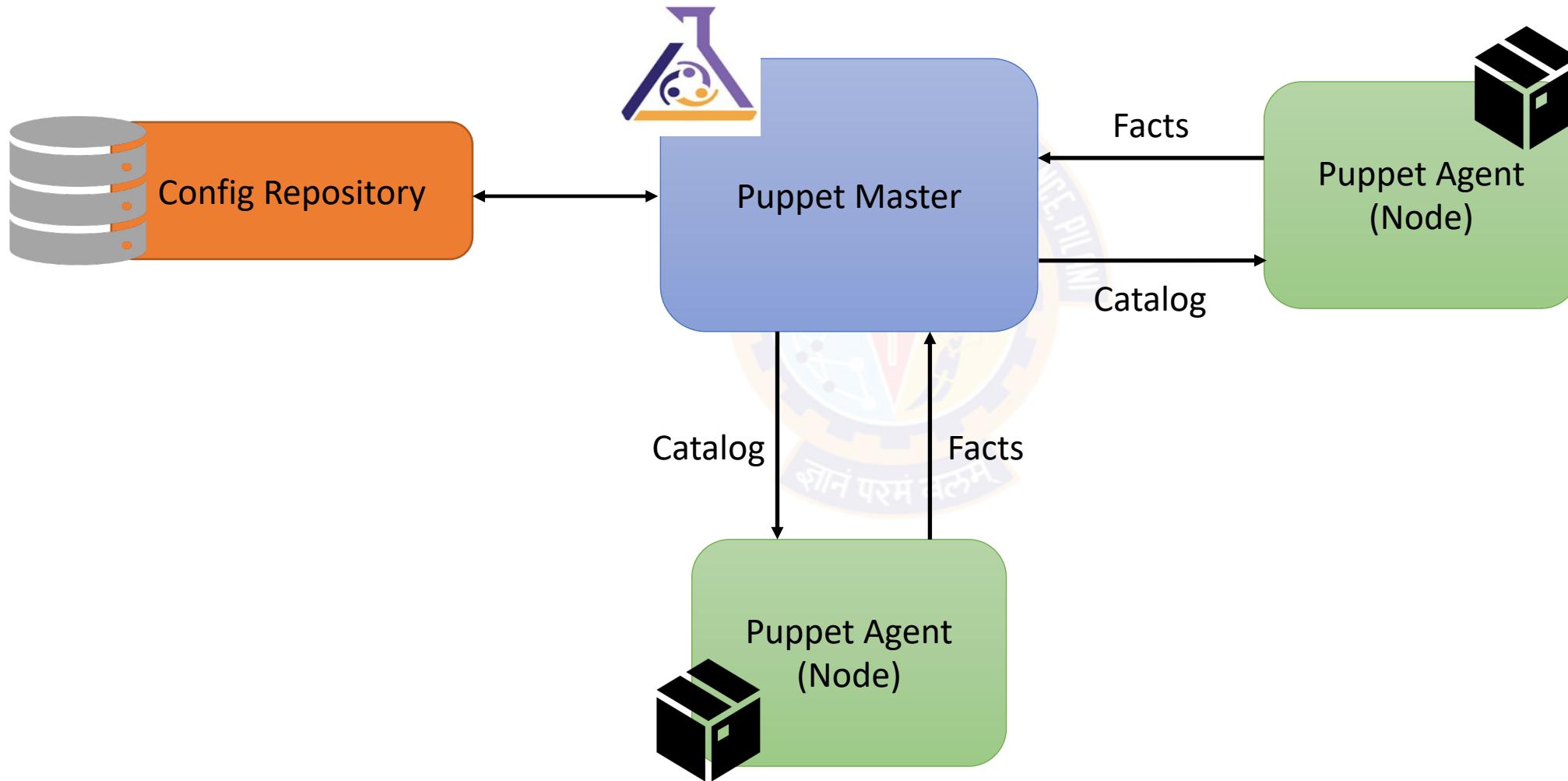


- Puppet Forge was launched in 2012
- The Puppet Forge is one of the most utilized resources in the Puppet world
- It is a repository of modules written both by Puppet and by the Puppet user community
- These modules solve a wide variety of problems, and using them can save you time and effort
- Puppet Forge has over 5,500 modules published
- The puppet module tool provides a command line interface for managing modules from the Forge



# Puppet Architecture

## Client-Server Architecture & Working



# Configuration Management Tool

## Ansible

### Why

#### Agentless

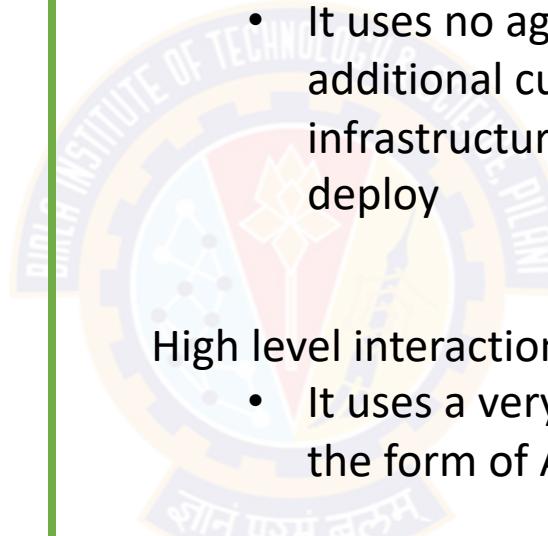
- It uses no agents and no additional custom security infrastructure, so it's easy to deploy

#### High level interaction

- It uses a very simple language (YAML, in the form of Ansible Playbooks)

#### Multi-tier deployment

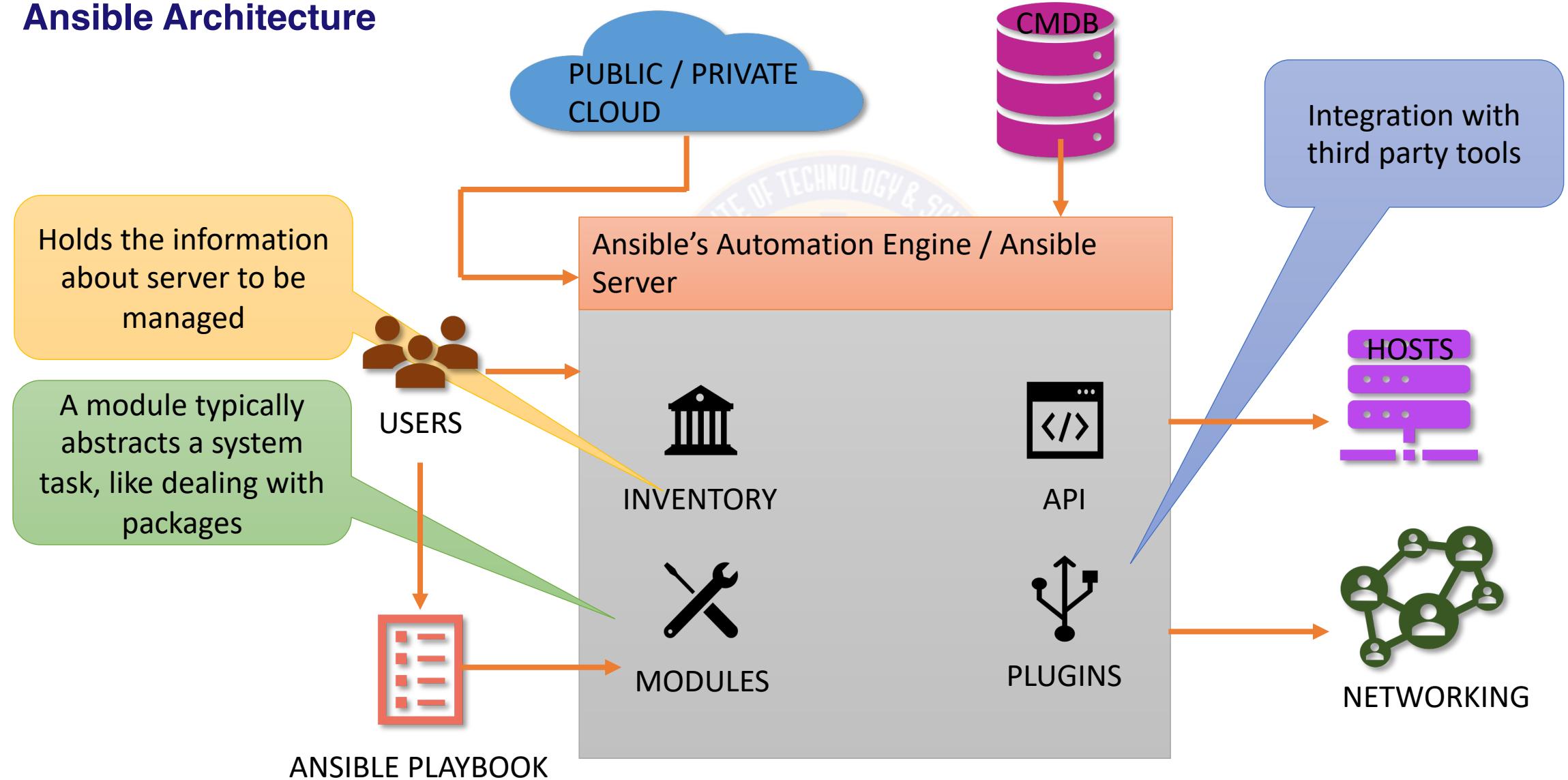
- Ansible models your IT infrastructure by describing how all of your systems inter-relate, rather than just managing one system at a time



ANSIBLE

# Ansible

## Ansible Architecture



ANSIBLE

# Ansible

## Playbook, Facts, Handlers

- The entry point for Ansible provisioning, where the automation is defined through tasks using YAML format
  - Play: A provisioning executed from start to finish is called a play. In simple words, execution of a playbook is called a play
  - Task: A block that defines a single procedure to be executed, e.g. Install a package
- Facts: Global variables containing information about the system, like network interfaces or operating system
- Handlers: Used to trigger service status changes, like restarting or stopping a service

# Ansible

## Examples

```
---  
[webservers]  
www1.example.com  
www2.example.com  
  
[dbservers]  
db0.example.com  
db1.example.com
```

### Example of Inventory

```
---  
- hosts: webservers  
  serial: 5 # update 5 machines at a time  
  roles:  
    - common  
    - webapp  
  
- hosts: content_servers  
  roles:  
    - common  
    - content
```

### Example of Playbook

```
---  
- hosts: webservers  
  vars:  
    http_port: 80  
    max_clients: 200  
    remote_user: root  
  tasks:  
    - name: ensure apache is at the latest version  
      yum:  
        name: httpd  
        state: latest  
    - name: write the apache config file  
      template:  
        src: /srv/httpd.j2  
        dest: /etc/httpd.conf  
      notify:  
        - restart apache  
    - name: ensure apache is running  
      service:  
        name: httpd  
        state: started  
  handlers:  
    - name: restart apache  
      service:  
        name: httpd  
        state: restarted
```

### Example: Playbook of webapp

# Puppet Vs Ansible

## Lets Compare

### Puppet

- Puppet is introduced in 2005
- Agent based Solution
- Puppet is model-driven and was built with systems administrators in mind; Moderate complex to setup and use
- Pull Base Methodology; end nodes contact back to Master
- Matured knowledge base
- Puppet Forge: Almost 6000 Modules available

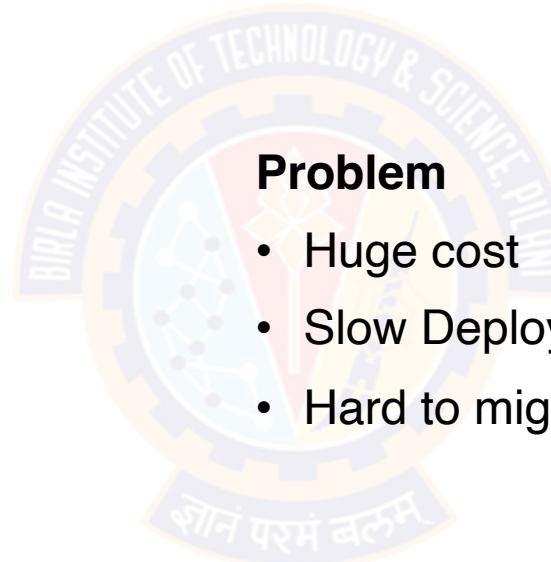
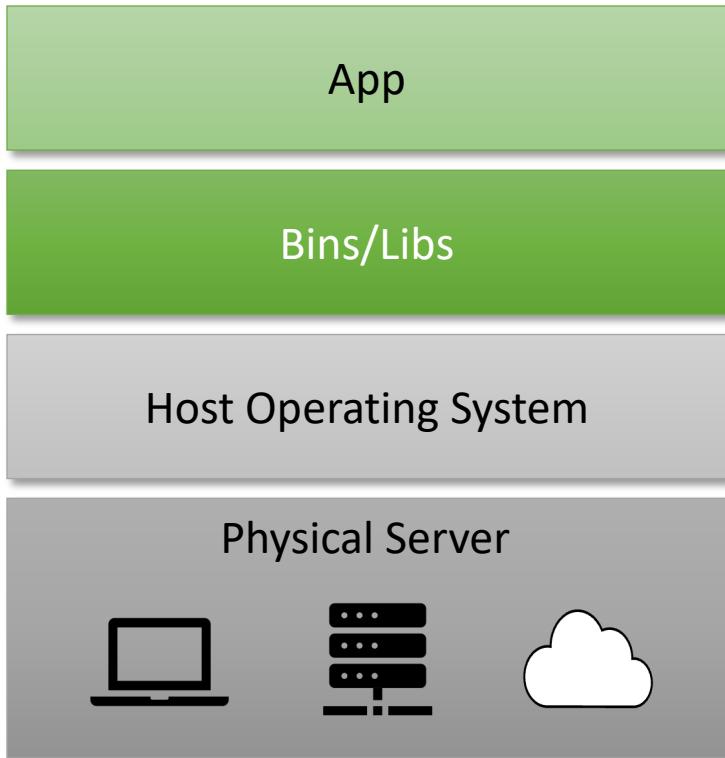
### Ansible

- Ansible is introduced in 2012
- Agent less Solution
- Ansible is easy to setup and understand than Puppet; More end user friendly
- Push Base Methodology; Master push tasks to end nodes
- Growing knowledge base
- Ansible Galaxy is still growing



# Pre virtualization World

## How it was



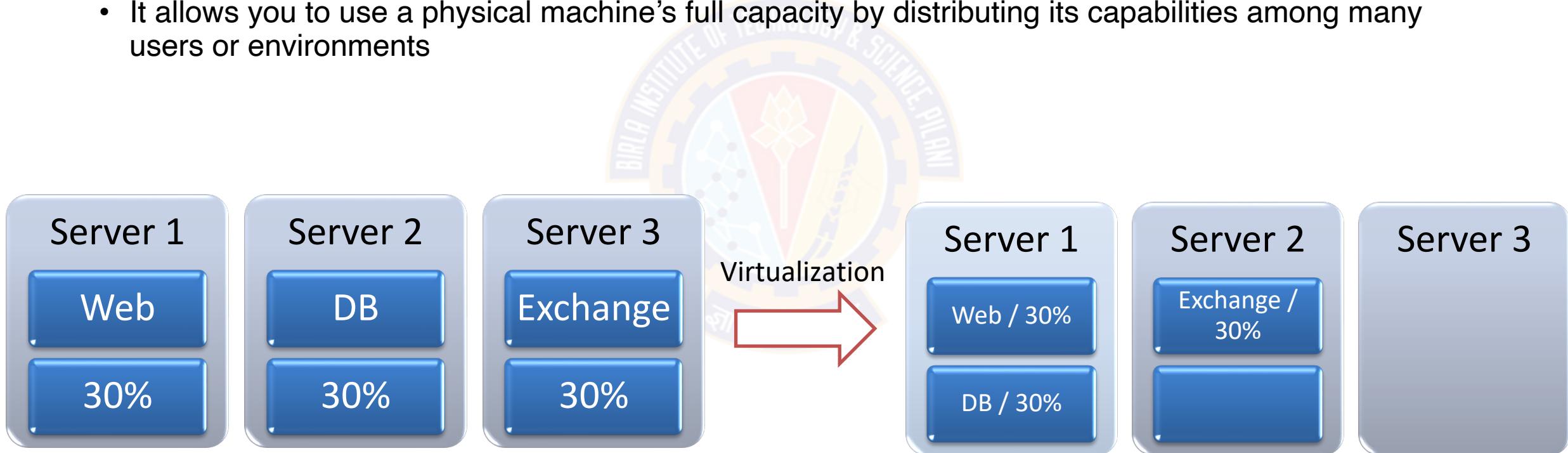
### Problem

- Huge cost
- Slow Deployment
- Hard to migrate

# Virtualization!!!

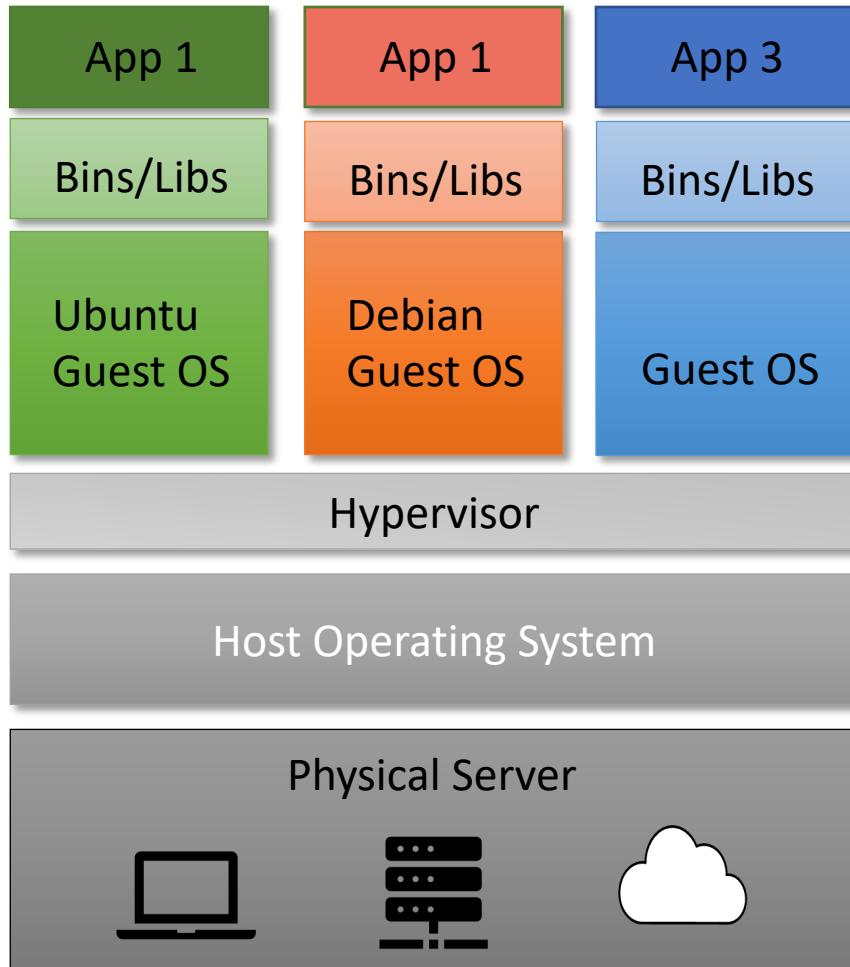
## How it is

- Virtualization is technology that lets you create useful IT services using resources that are traditionally bound to hardware
- It allows you to use a physical machine's full capacity by distributing its capabilities among many users or environments



# Virtualization!!!

## H/w level virtualization



### Benefits:

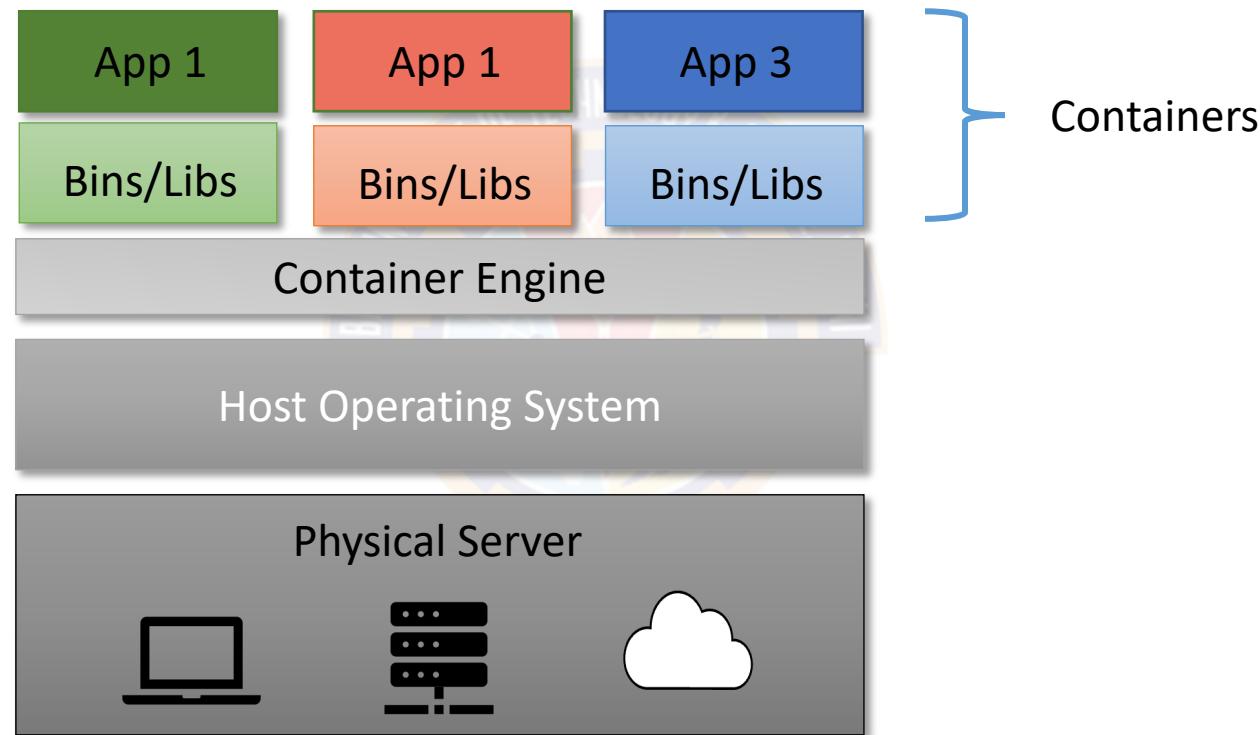
- More cost effective
- Easy to scale

### Limitations:

- Kernel Resource Duplication
- Application portability issue

# Virtualization!!!

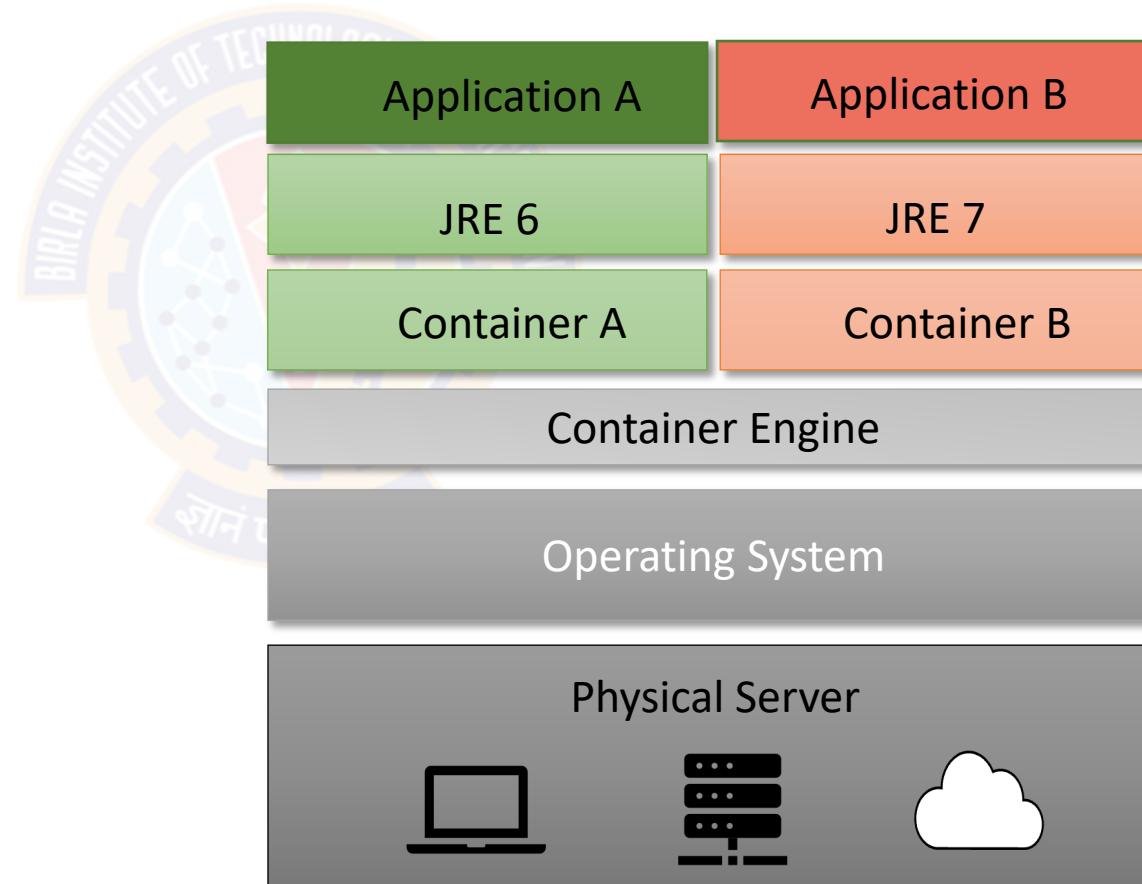
## OS level virtualization



# Container based Virtualization

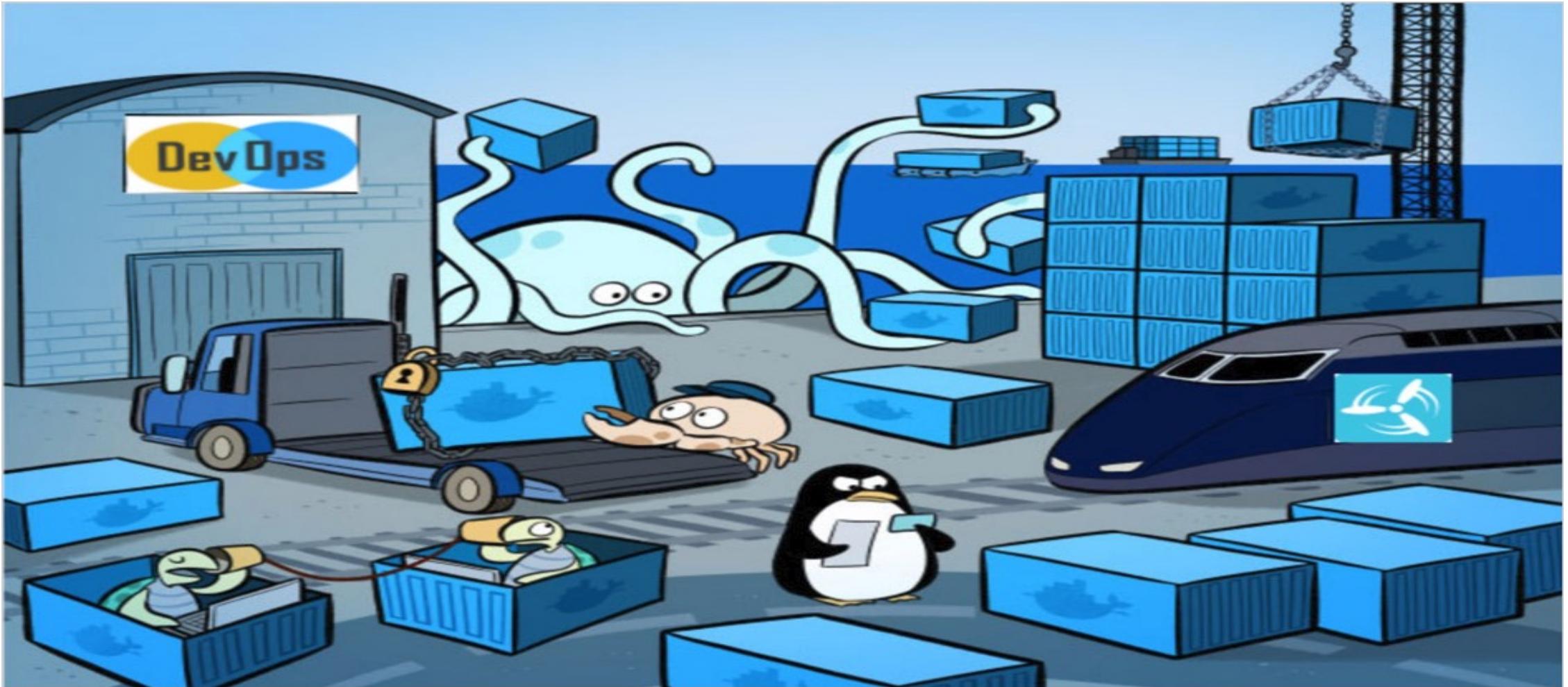
## Benefits

- More cost effective
- Faster deployment speed
- Great portability



# Container based Virtualization

## Docker



# Docker

## Why



Open platform for developing, shipping, and running applications

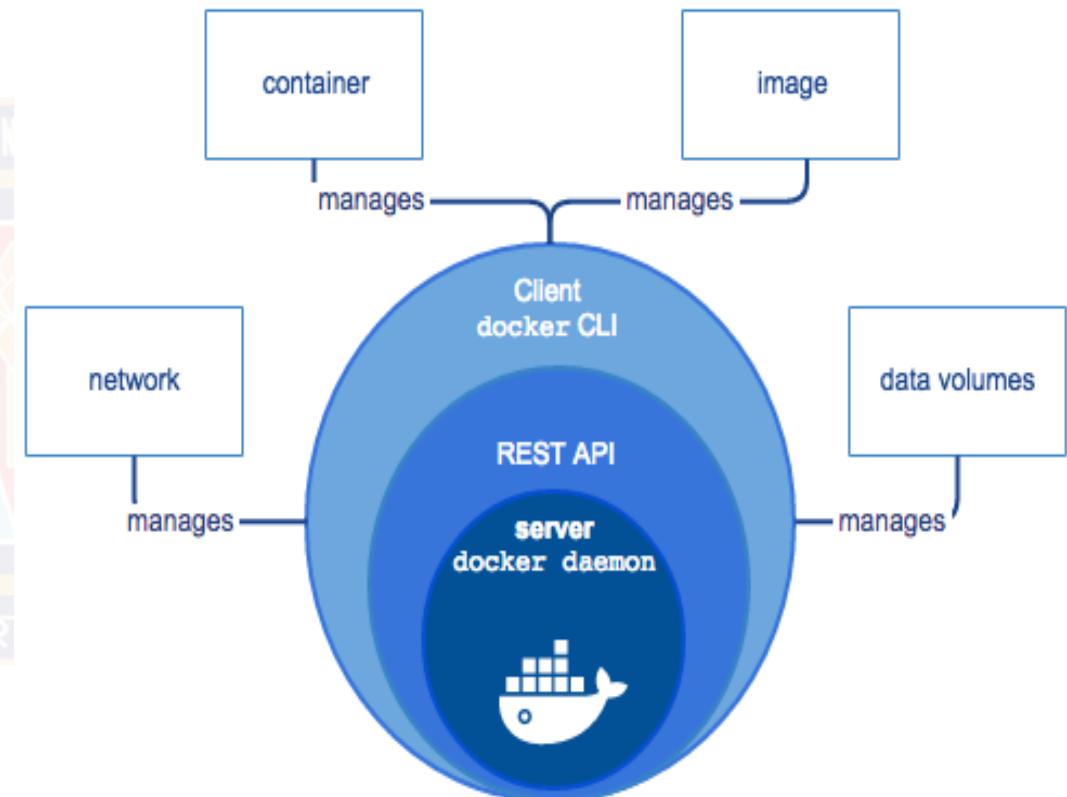
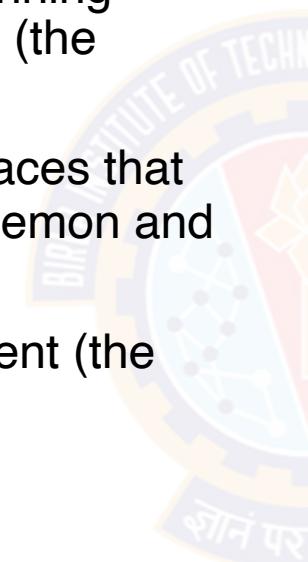
Enables you to separate your applications from your infrastructure so you can deliver software quickly

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production

# Docker

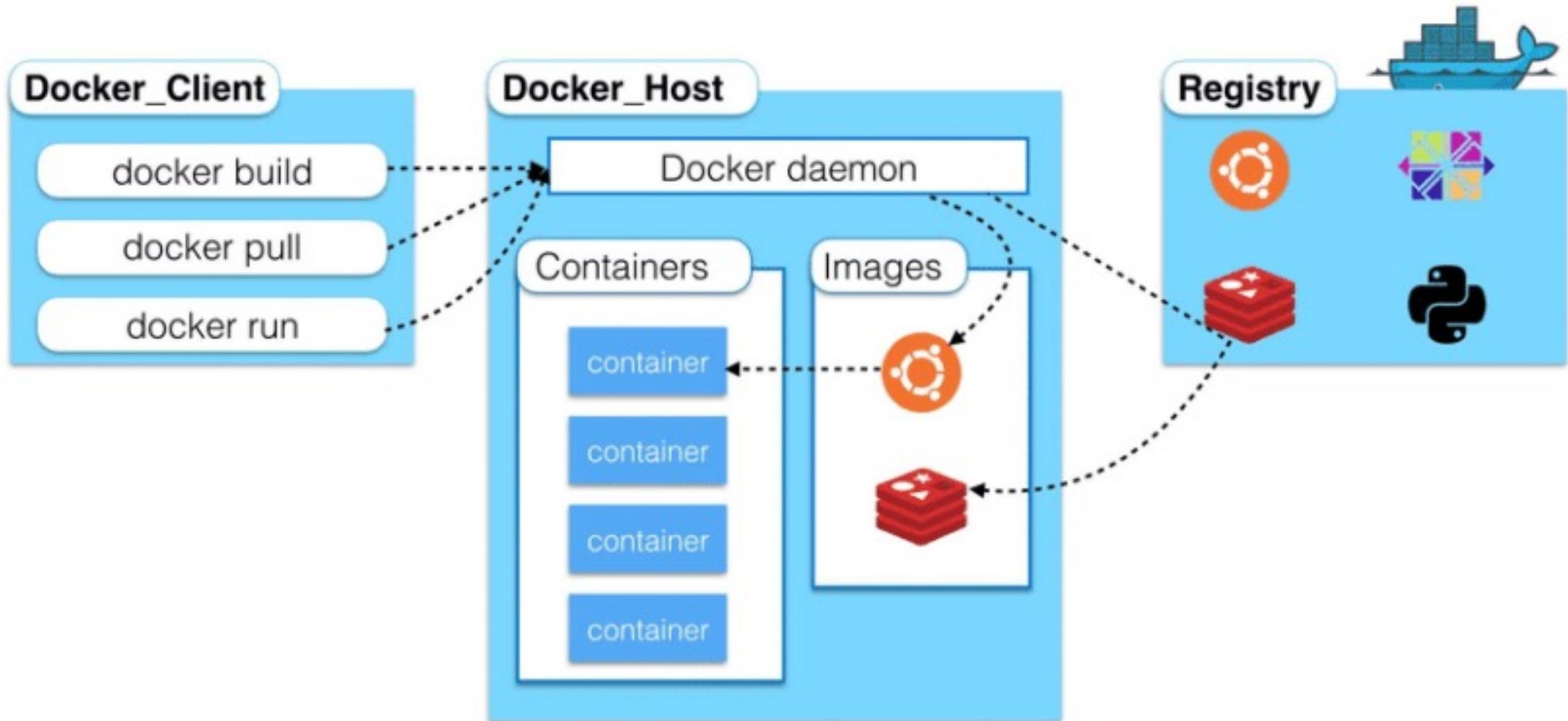
## Docker Engine

- Client-server application
- A server which is a type of long-running program called a daemon process (the `dockerd` command)
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do
- A command line interface (CLI) client (the `docker` command)



# Docker architecture

## How it Works!!!



# Docker Concepts

## Images and Containers

### Image



- An image is an executable package
- An image is a read-only template with instructions
- To build your own image, you create a Dockerfile

### Containers



- A container is launched by running an image
- A container is a runtime instance of an image
- One can create, start, stop, move, or delete a container using the Docker API or CLI

# Docker Concepts

## Service

- Services are really just containers in production
- A service only runs one image, but it codifies
  - Way the Image Runs
  - What Ports it should use
  - How many replicas and etc.
- Scaling a service
  - No of container instances
  - Assign more compute resources



```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - webnet
networks:
  webnet:
```

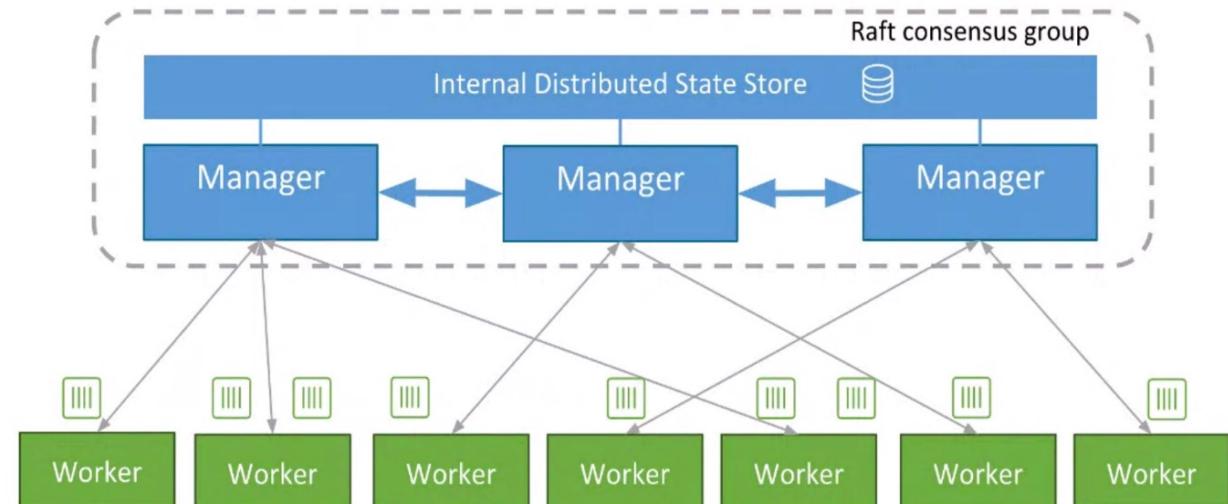
*Docker-compose.yml*

# Docker Concepts

## Swarms

- A swarm is a group of machines that are running Docker and joined into a cluster
- Docker Command gets executed on a cluster by a swarm manager
- The machines in a swarm can be physical or virtual
- After joining a swarm, machines are referred to as nodes
- Swarm Strategies:
  - emptiest node
  - global

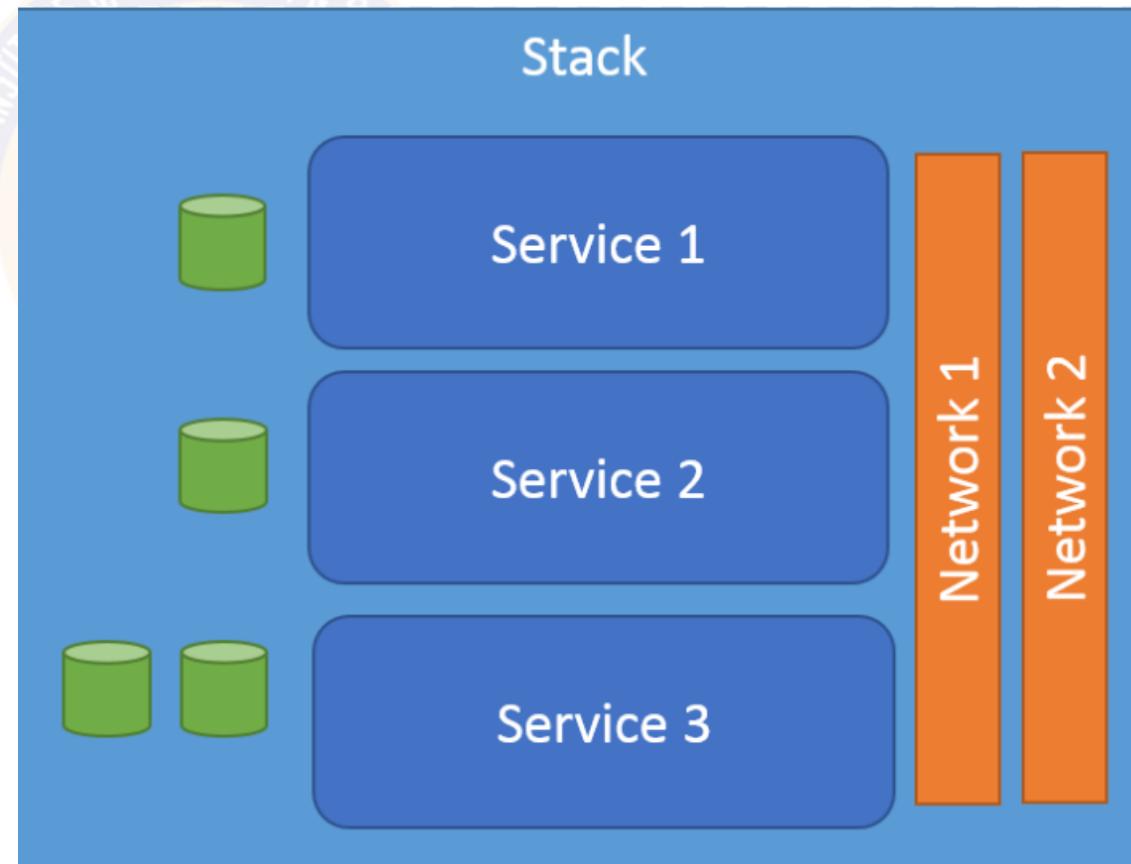
### Swarm Architecture



# Docker Concepts

## Stacks

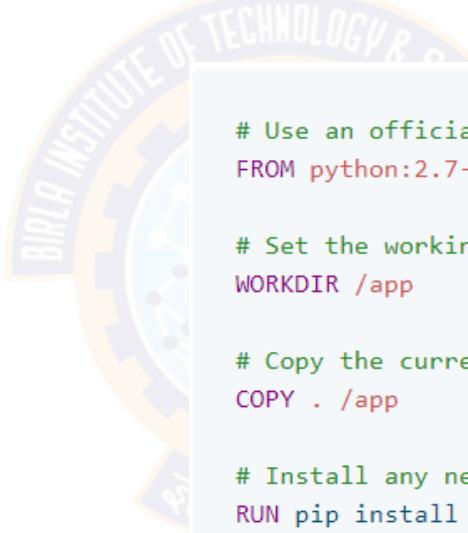
- A stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together
- A single stack is capable of defining and coordinating the functionality of an entire application



# Docker Concepts

## Docker file

- Dockerfile defines what goes on in the environment inside a container
- This file has information about all resources, that is network interface and disk drives used for the container



```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

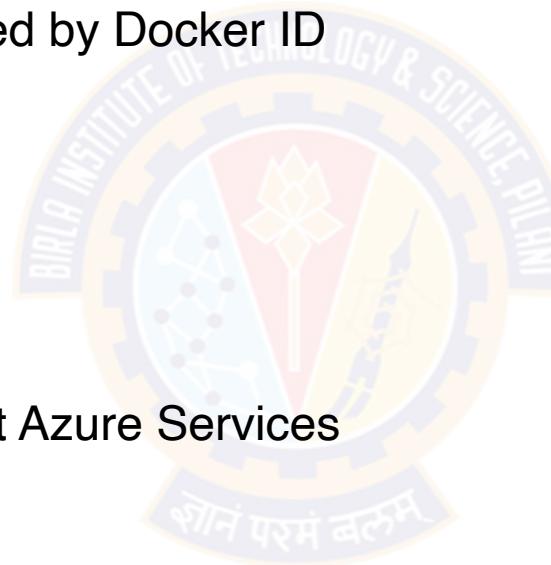
# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

# Docker Concepts

## Docker Cloud

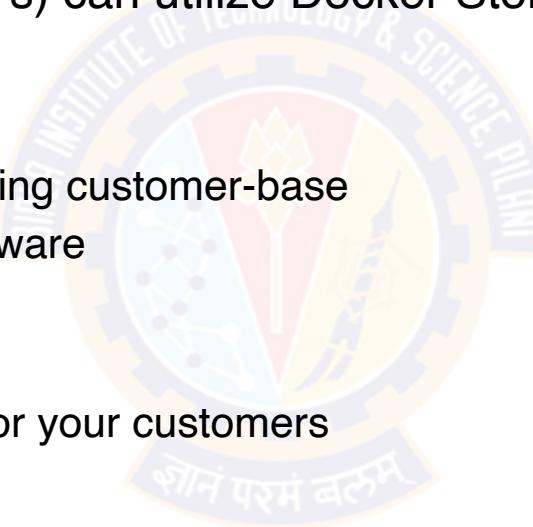
- Docker Cloud provides a hosted registry service with build and testing facilities for Dockerized application images
- Access to Docker Cloud is managed by Docker ID
- Manage Builds and Images
- Manage Swarms
- Manage Infrastructure
- Manage Nodes and Apps
- Integration with AWS and Microsoft Azure Services



# Docker Concepts

## Docker Store

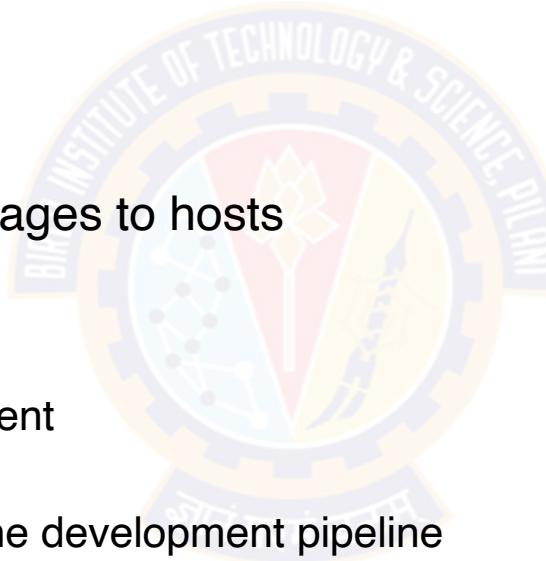
- For developers and operators, Docker Store is the best way to discover high-quality Docker content
- Independent Software Vendors (ISVs) can utilize Docker Store to distribute and sell their Dockerized content
- Store Experience:
  - Access to Docker's large and growing customer-base
  - Customers can try or buy your software
  - Use of Docker licensing support
  - Docker handle checkout
  - Seamless updates and upgrades for your customers
  - Become Docker Certified



# Docker Concepts

## Docker Hub

- Docker Hub is a cloud-based registry service
- Allows link to code repositories
- Build images and test them
- Stores Images
- Links to Docker Cloud to deploy images to hosts
- It is a Centralize Solution for:
  - container image discovery
  - distribution and change management
  - user and team collaboration
  - workflow automation throughout the development pipeline



# Docker Concepts

## How is Docker Store different from Docker Hub

| Docker Hub  | Docker Store   |
|---|--|
| Contents by Docker community  | Contents submitted for approval by qualified Store Vendor Partners |
| Anyone can push new images to the community                           | Contents are published and maintained by Entity                    |
| No guarantees around the quality or compatibility                     | Docker Certified quality assurance                                 |
| <b>Common Thing: The Docker official Images are Available in both</b> |  |

# References

## External

- <https://docs.chef.io/>
- <https://docs.puppet.com/>
- <https://forge.puppet.com/>
- <https://docs.ansible.com/>
- <https://www.redhat.com/en/topics/virtualization/what-is-virtualization>
- <https://docs.docker.com/engine/docker-overview/#docker-engine>
- <https://docs.docker.com/>





# Q&A



# Thank You!

In our next session:



**BITS Pilani**  
Pilani | Dubai | Goa | Hyderabad

# Introduction to DevOps

**Sonika Rathi**

---

Assistant Professor  
BITS Pilani

# Agenda

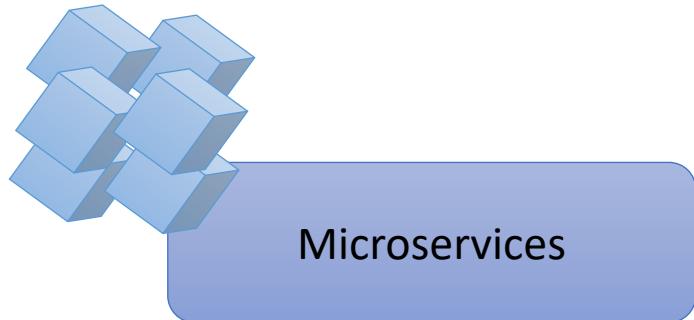
## Microservices and Current Trends

- Introduction to Microservices
  - Need of Microservices for DevOps
  - Benefits of Microservice and DevOps
- Introduction to Kubernetes
  - Kubernetes Architecture
  - Kubernetes Benefits
- AWS Lambda
  - Function as a Service [FaaS]
  - Serverless Computing
  - AWS Lambda - Success Story
  - AWS Lambda Benefits



# Microservices

## Introduction



Software development technique

A type of the service-oriented architecture (SOA); that structures an application as a collection of loosely coupled services

It improves modularity

Enables small autonomous teams to develop, deploy and scale their respective services independently

Allows the architecture of an individual service to emerge through continuous refactoring

# Microservices

## Introduction

Who

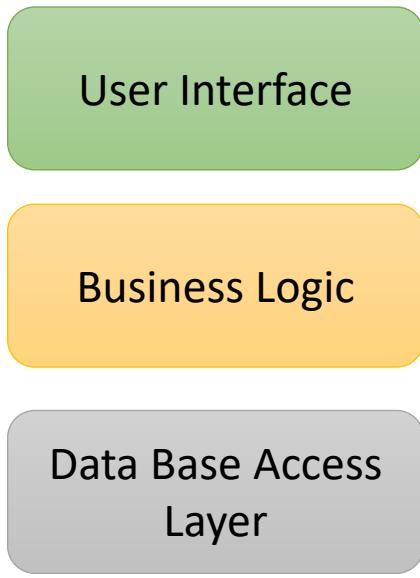


Uber  
Netflix  
Amazon  
Ebay  
Gilt  
Tesla

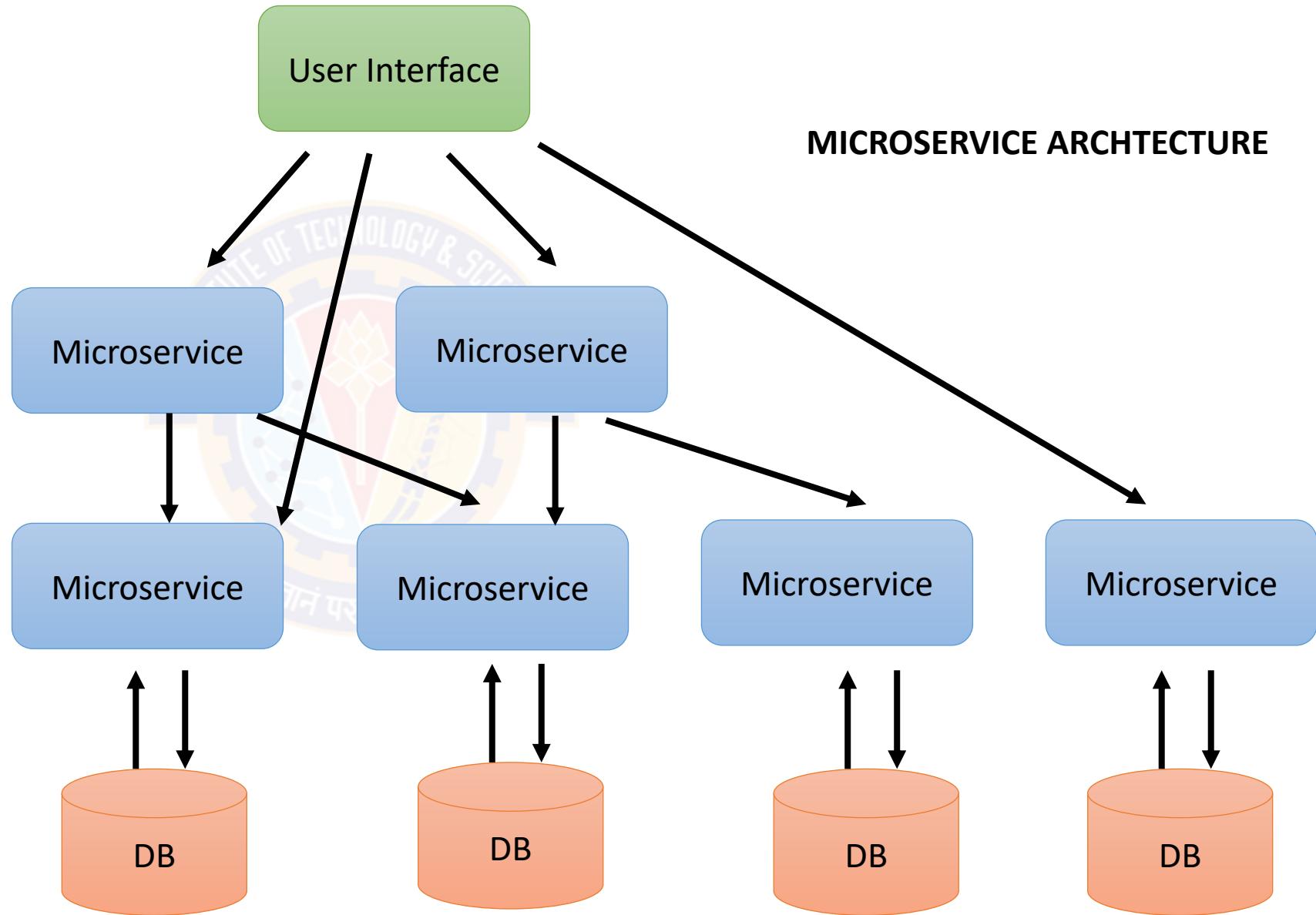


# Microservice Architecture

## MONOLITHIC ARCHTECTURE

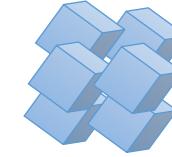
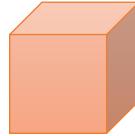


## MICROSERVICE ARCHTECTURE



# Microservices

Lets compare



| Monolithic – Software Development                 | Microservices – Software Development                      |
|---|---|
| Traditional Software Development [Waterfall]      | New Approach to Software Development [Agility]            |
| Large Team Working on single complex Application  | Services are encouraged to be small by handful Developers |
| No Single Developer understand entire Application | Developers understand the Services                        |
| Limited Reuse is realized                         | Use of REST API – reused by other services                |
| Scaling is Challenge                              | Services can be scaled as they exists Independent         |
| Challenge for Operational Agility                 | Services and Teams become Agile                           |
| Single Development Stack                          | Autonomous Service Development Stacks                     |

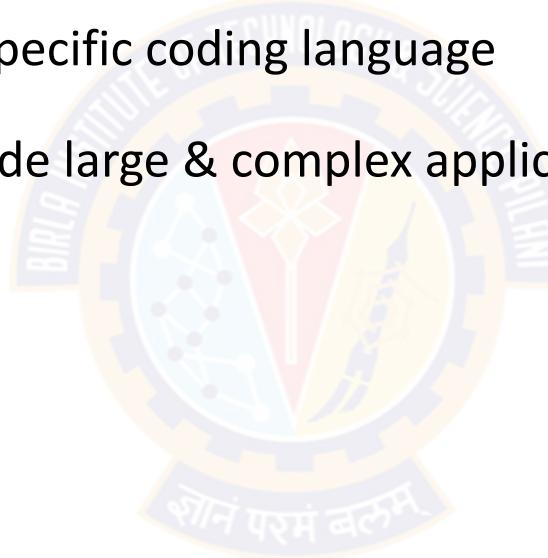
# Microservices

## How does it works

The Idea is to break down applications into smaller, independent services

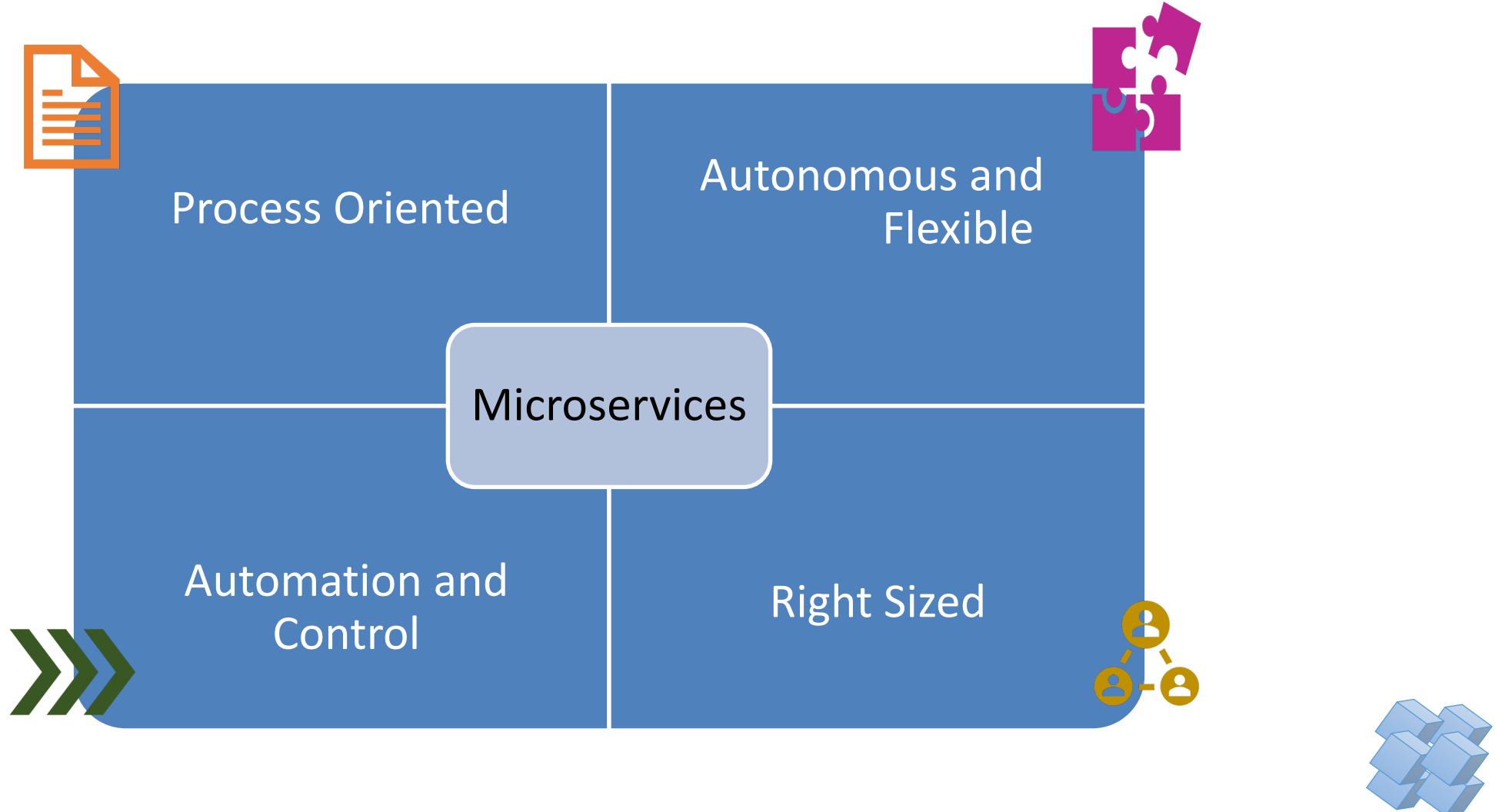
That will not dependent upon a specific coding language

Using Microservices Ideology divide large & complex applications in to smaller building blocks



# Microservices

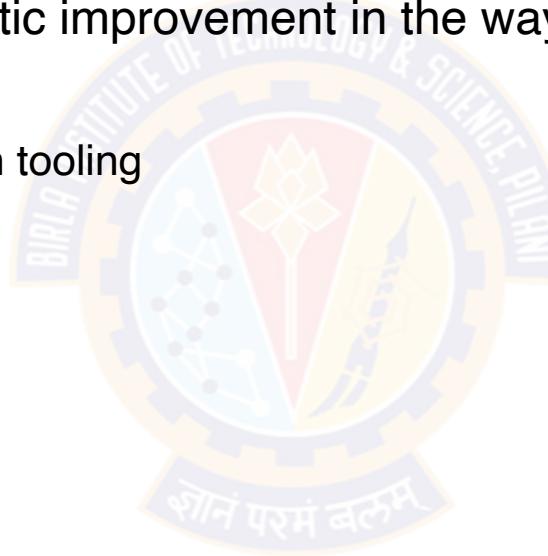
## Characteristics of Microservices



# Microservices and DevOps

## Need of Microservices for DevOps

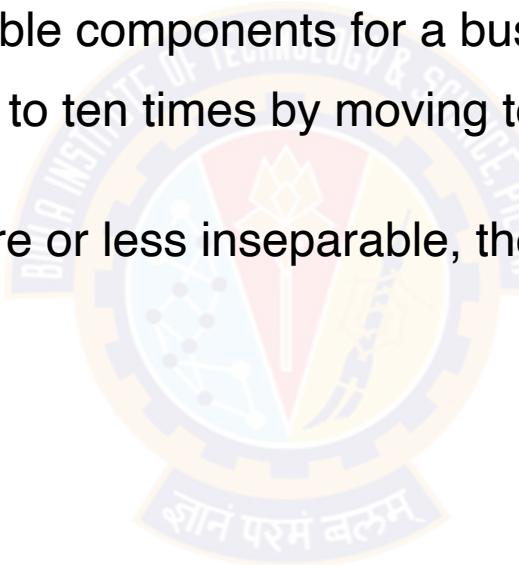
- DevOps promotes small, more empowered and autonomous teams, along with automation and measurements
- It has great potential to be a fantastic improvement in the way IT and business work together
- Challenging Area:
  - Setup Pipeline for new automation tooling
  - The mindset of Architects



# Microservices and DevOps

## Microservices enabling DevOps

- With small autonomous DevOps teams, the world will start producing small autonomous components, called microservices
- It makes sense to produce deployable components for a business function
- This will help in speed improve five to ten times by moving to the cloud and using a high productivity platform
- DevOps and microservices are more or less inseparable, they can hardly exist in separation



# Microservices and DevOps

## Benefits

### Benefits

Deployability

Reliability

Availability

Scalability

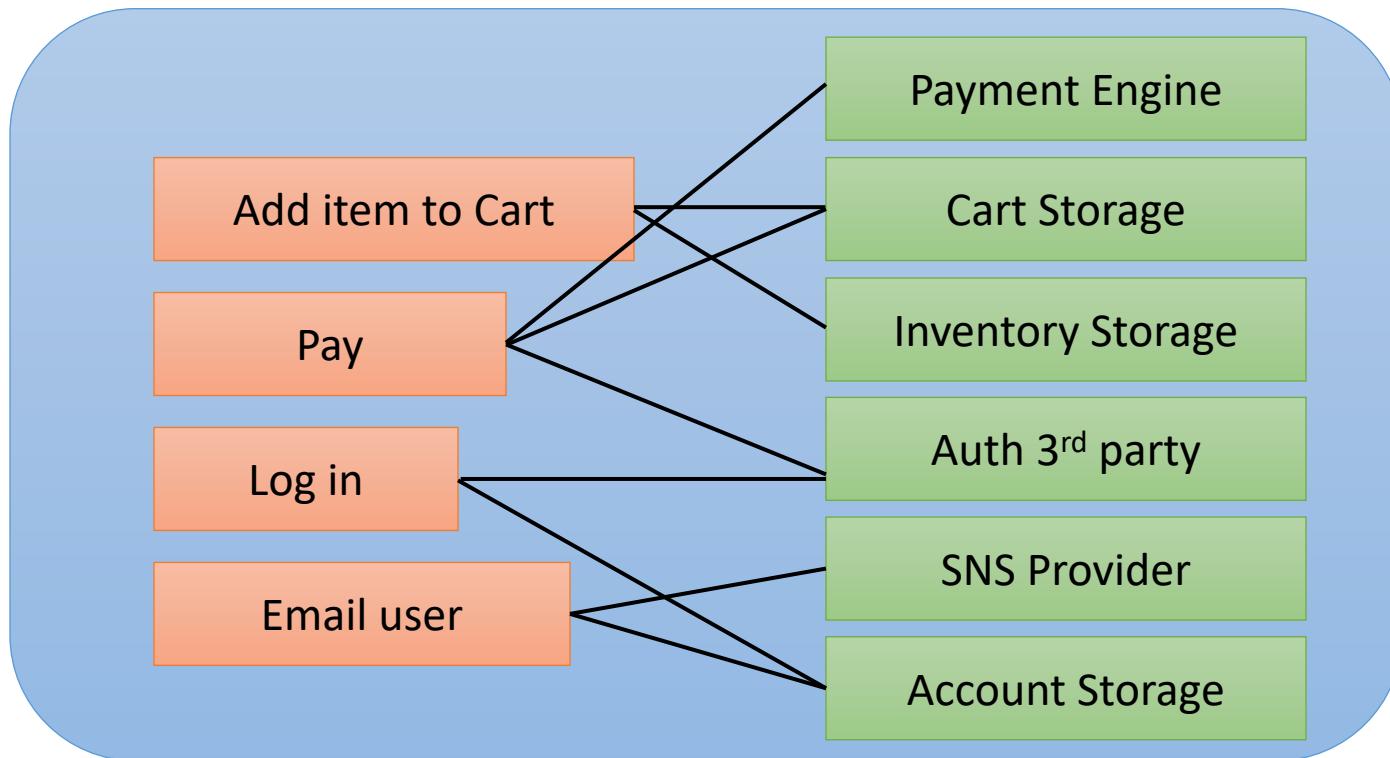
Modifiability

Management



# Microservices Example

Let's understand with example

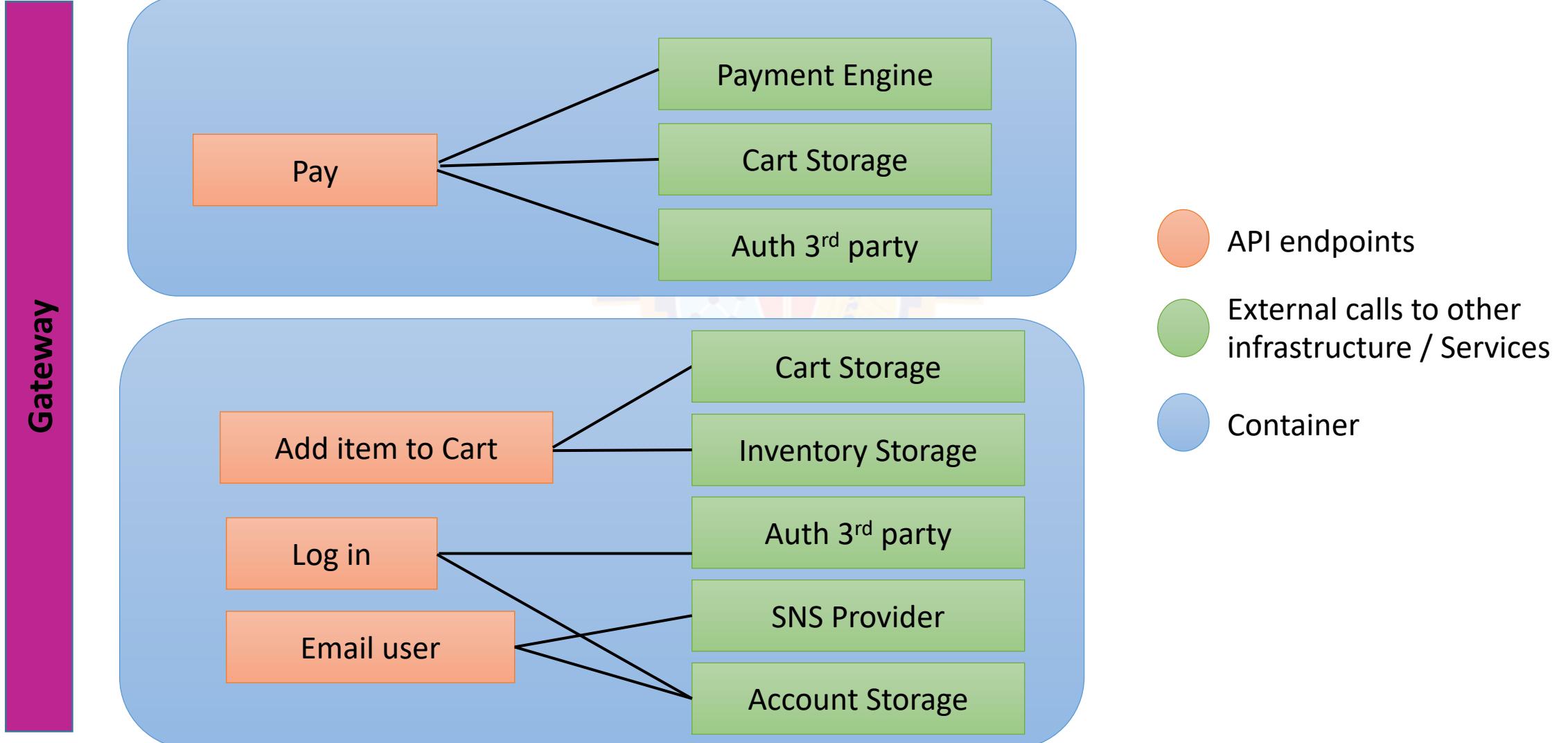


Monolith application example

- API endpoints
- External calls to other infrastructure / Services
- Container

# Microservices Example

Our Goal is to split out Payments service from the monolith

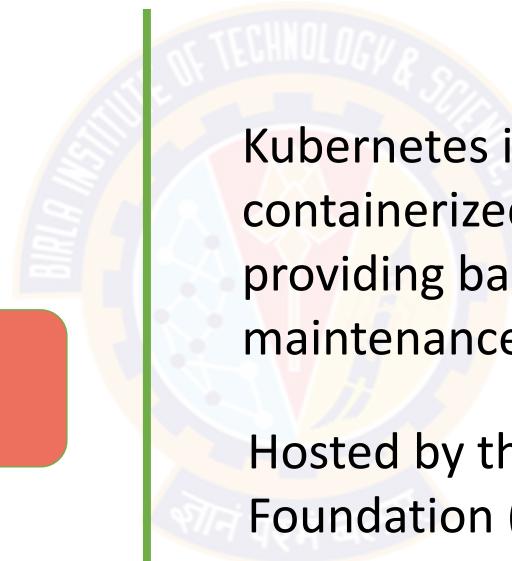


# Kubernetes

## Introduction



**Kubernetes**



Kubernetes is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications

Hosted by the Cloud Native Computing Foundation (CNCF)

Symbol K8 - is an abbreviation derived by replacing the 8 letters “ubernete” with “8”

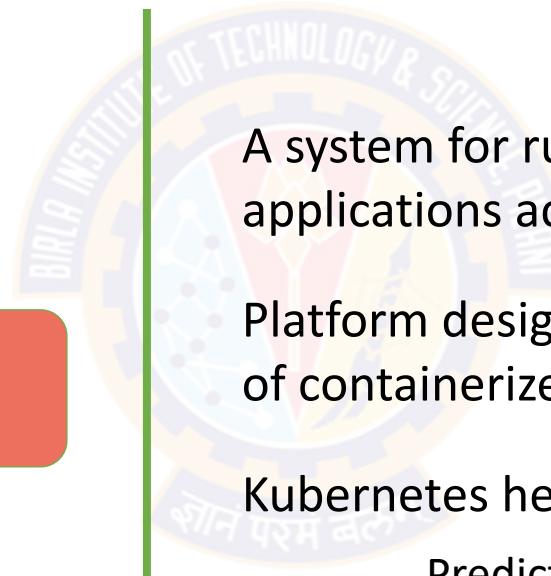


# Kubernetes

## What?



**Kubernetes**



A system for running and coordinating containerized applications across a cluster of machines

Platform designed to completely manage the life cycle of containerized applications and services

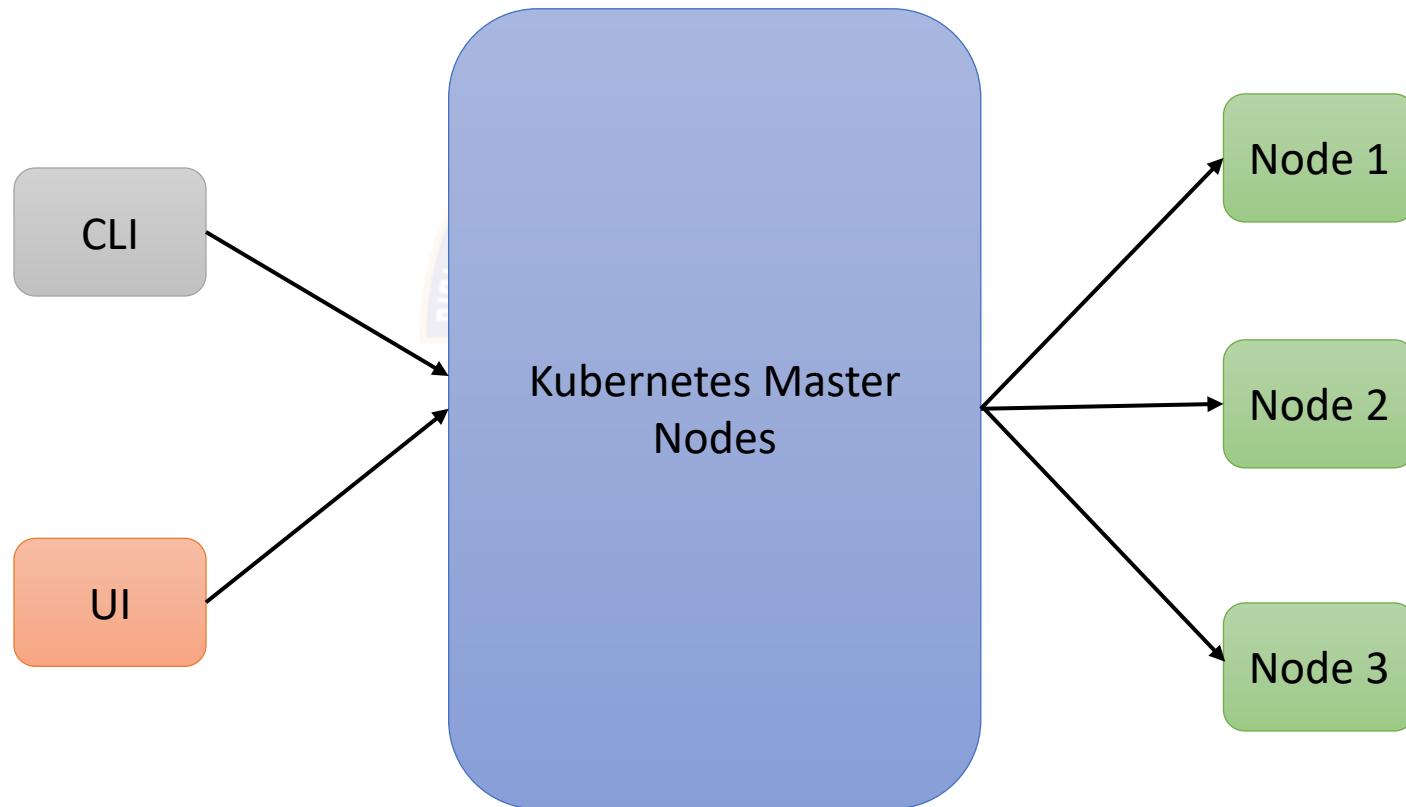
Kubernetes helps with methods that provide

- Predictability
- Scalability
- High availability



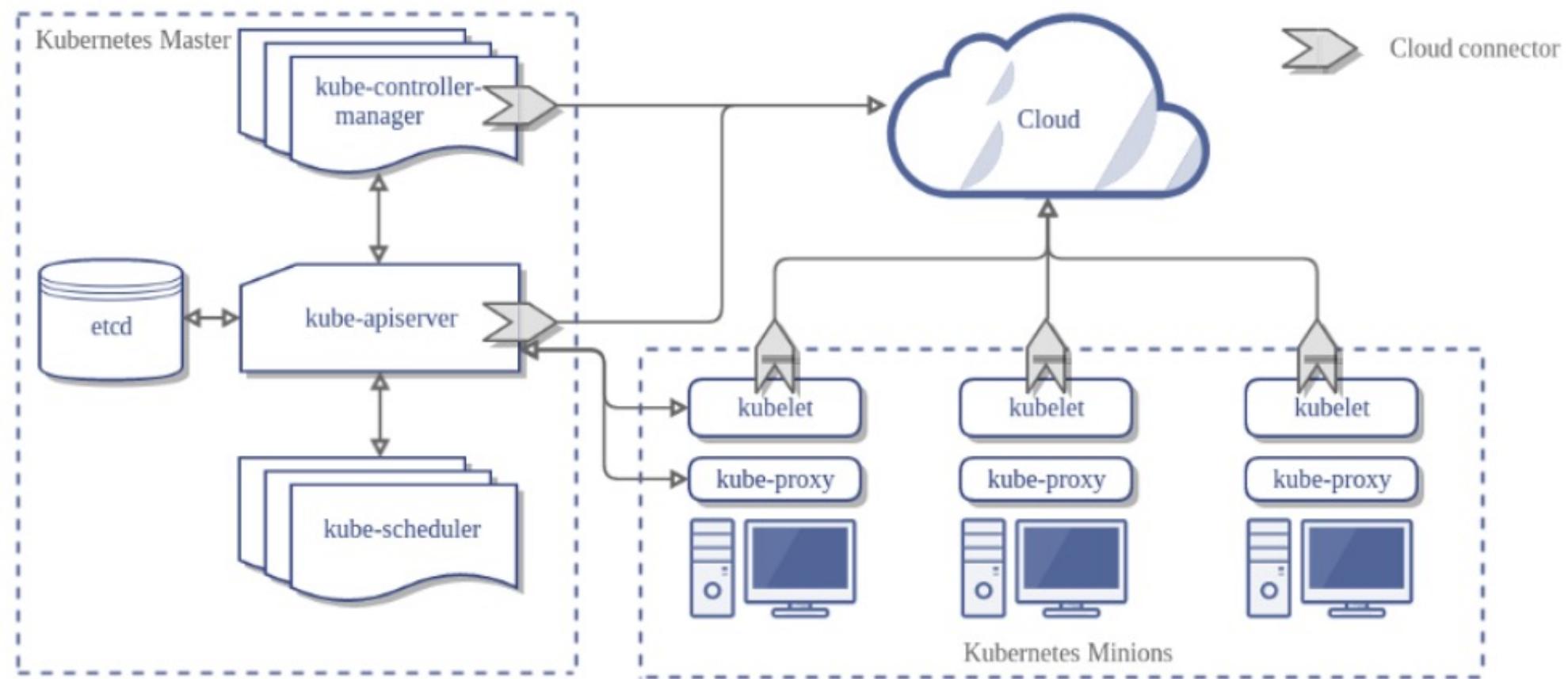
# Kubernetes

## Components



# Kubernetes

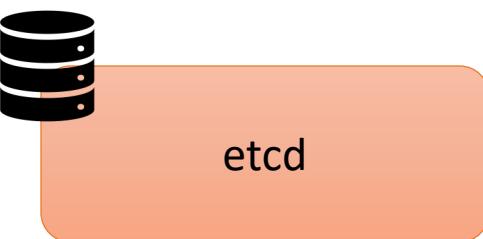
## Architecture



# Kubernetes Components

## Kubernetes Master Server Components

- Kubernetes master server acts as the primary control plane for Kubernetes clusters
- Overall the components on the master server work together to accept user requests, determine the best ways to schedule workload containers, authenticate clients and nodes, adjust cluster-wide networking, and manage scaling and health checking responsibilities
- These components can be installed on a single machine or distributed across multiple servers



Globally available configuration store

Distributed key value store

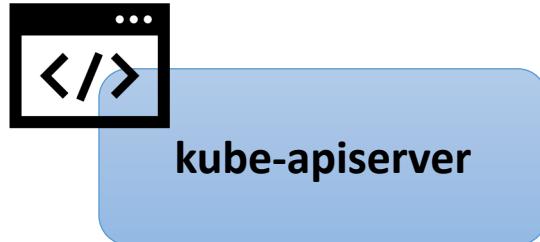
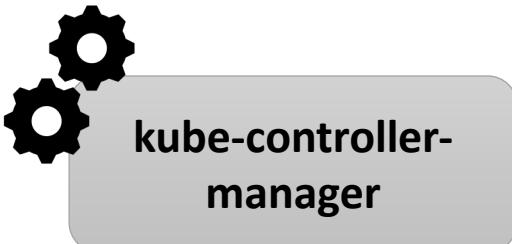
Configured to span across multiple nodes

etcd can be configured on a single master server or, in production scenarios, distributed among a number of machines



# Kubernetes Components

## Kubernetes Master Server Components



### Main Management Point

It allows a user to configure Kubernetes' workloads and organizational units

It is also responsible for making sure that the etcd store and the service details of deployed containers are in agreement

The API server implements a RESTful interface

It manages different controllers that regulate the state of the cluster, manage workload life cycles, and perform routine tasks

A replication controller ensures that the number of replicas defines for a pod machine

When a change is seen, the controller reads the new information and implements the procedure that fulfills the desired state

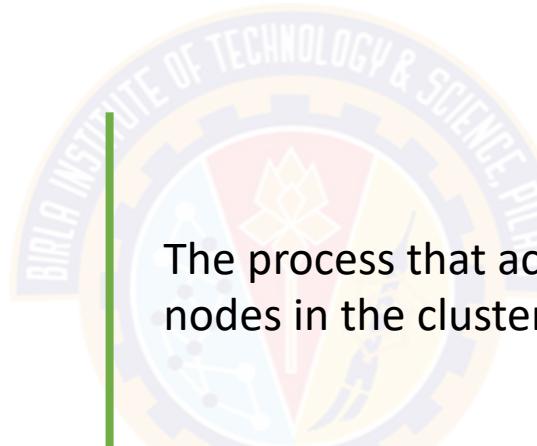


# Kubernetes Components

## Kubernetes Master Server Components



**kube-scheduler**



The process that actually assigns workloads to specific nodes in the cluster is the scheduler

The scheduler is responsible for tracking available capacity on each host to make sure that workloads are not scheduled in excess of the available resources



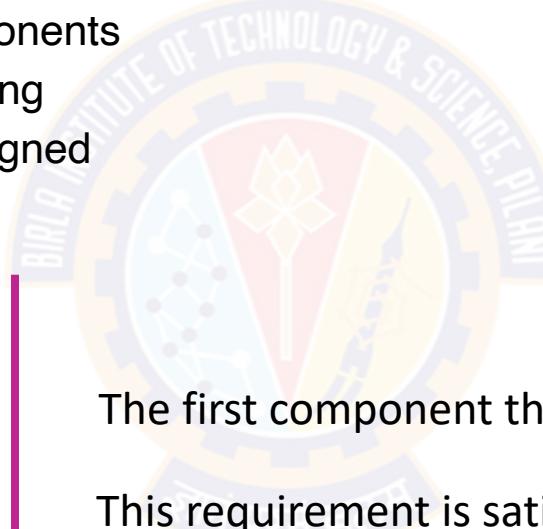
# Kubernetes Components

## Kubernetes Node Server Components

- In Kubernetes, servers that perform work by running containers are known as nodes
- Node servers have a few requirements that are necessary for :
  - Communicating with master components
  - Configuring the container networking
  - Running the actual workloads assigned



A Container  
Runtime



The first component that each node must have

This requirement is satisfied by installing and running Docker

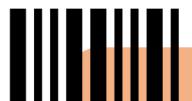
Responsible for starting and managing containers

Runs the containers defined in the workloads submitted to the cluster



# Kubernetes Components

## Kubernetes Node Server Components



**kube-proxy**



**kubelet**

To manage individual host subnetting and make services available to other components

This process forwards requests to the correct containers

It can do primitive load balancing

The main contact point for each node with the cluster group

Responsible for relaying information to and from the control plane services

Interact with the etcd store to read configuration details or write new values

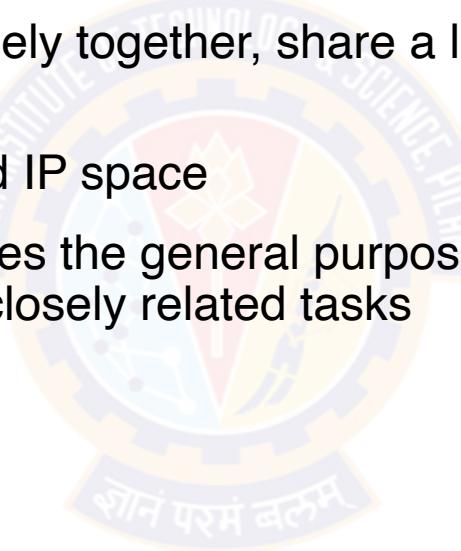
The kubelet service communicates with the master components to authenticate to the cluster and receive commands and work



# Kubernetes Objects

## Kubernetes Pods

- One or more tightly coupled containers are encapsulated in an object called a pod
- Generally represents one or more containers that should be controlled as a single application
- Consist of containers that operate closely together, share a life cycle, and should always be scheduled on the same node
- Share their environment, volumes, and IP space
- Consist of a main container that satisfies the general purpose of the workload and optionally some helper containers that facilitate closely related tasks

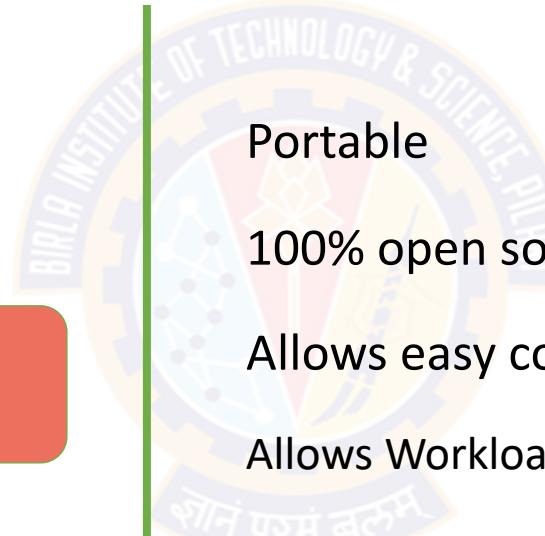


# Kubernetes

## Benefits



### Benefits



Portable

100% open source

Allows easy container management

Allows Workload Scalability

Supports High Availability

Efficient



# Kubernetes || Docker Swarm

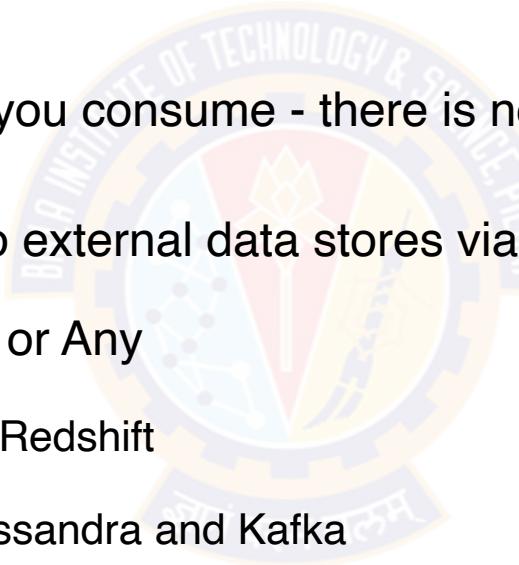
## Lets compare

| Features               | Kubernetes  | Docker Swarm  |
|------------------------|---|---|
| Installation           | Complex Installation but a strong resultant cluster once set up   | Simple installation but the resultant cluster is not comparatively strong                                     |
| GUI                    | Comes with an inbuilt Dashboard   | There is no Dashboard which makes management complex  |
| Scalability            | Highly scalable service that can scale with the requirements. 5000 node clusters with 150,000 pods      | Very high scalability. Up to 5 times more scalable than Kubernetes. 1000 node clusters with 30,000 containers |
| Load Balancing         | Manual load balancing is often needed to balance traffic between different containers in different pods | Capability to execute auto load balancing of traffic between containers in the same cluster                   |
| Rollbacks              | Automatic rollbacks with the ability to deploy rolling updates  | Automatic rollback facility available only in Docker 17.04 and higher if a service update fails to deploy     |
| Logging and Monitoring | Inbuilt tools available for logging and monitoring  | Lack of inbuilt tools. Needs 3rd party tools for the purpose  |
| Node Support           | Supports up to 5000 nodes   | Supports 2000+ nodes  |
| Optimization Target    | Optimized for one single large cluster  | Optimized for multiple smaller clusters   |
| Updates                | The in-place cluster updates have been constantly maturing  | Cluster can be upgraded in place  |
| Networking             | An overlay network is used which lets pods communicate across multiple nodes                            | The Docker Daemons is connected by overlay networks and the overlay network driver is used                    |
| Availability           | High availability. Health checks are performed directly on the pods                                     | High availability. Containers are restarted on a new host if a host failure is encountered                    |

# AWS Lambda

## Function as a Service [FaaS]

- AWS Lambda is a compute service that lets you run code without provisioning or managing servers
- “serverless” computing
- You pay only for the compute time you consume - there is no charge when your code is not running
- Lambda functions can write state to external data stores via web requests
- External Data store can be of AWS or Any
  - AWS Solutions: S3, Dynamo, and Redshift
  - Other Solutions: PostgreSQL, Cassandra and Kafka



# AWS Lambda : Success Story The Seattle Times

## Challenges

After maintaining on-premises hardware and custom publishing software for nearly two decades, The Seattle Times sought to migrate its website publishing to a contemporary content management platform

To avoid the costs of acquiring and configuring new hardware infrastructure and the required staff to maintain it, the company initially chose a fully managed hosting vendor

But after several months, The Times' software engineering team found it had sacrificed flexibility and agility in exchange for less maintenance responsibility

As the hosted platform struggled with managing traffic under a vastly fluctuating load, The Seattle Times team was constrained in its ability to scale up to meet customer demand



# AWS Lambda : Success Story The Seattle Times

## Why Amazon Web Services?

To address these core scalability concerns, The Seattle Times engineering team considered several alternative hosting options, including self-hosting on premises, more flexible managed hosting options, and various cloud providers

The team concluded that the available cloud options provided the needed flexibility, appropriate architecture, and desired cost savings. The company ultimately chose Amazon Web Services (AWS), in part because of the maturity of the product offering and, most significantly, the auto-scaling capabilities built into the service

The Seattle Times deployed its new system in just six hours. The website moved to the AWS platform between 11 p.m. and 3 a.m. and final testing was completed by 5 a.m. — in time for the next news day



# AWS Lambda : Success Story The Seattle Times

## End Result

With AWS, The Seattle Times can now automatically scale up very rapidly to accommodate spikes in website traffic when big stories break, and scale down during slower traffic periods to reduce costs

“Auto-scaling is really the clincher to this,” Seattle Times says. “With AWS, we can now serve our online readers with speed and efficiency, scaling to meet demand and delivering a better reader experience.”

**“AWS Lambda provides us with extremely fast image resizing,”** Seattle times says.

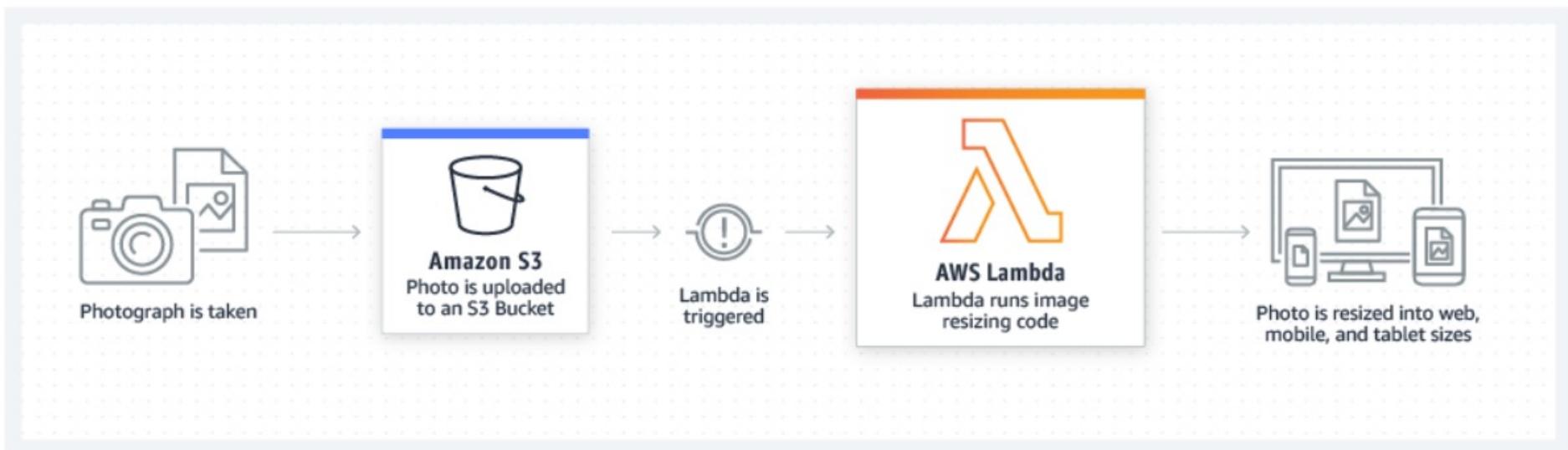
**“Before, if we needed an image resized in 10 different sizes, it would happen serially. With AWS Lambda, all 10 images get created at the same time, so it’s quite a bit faster and it involves no server maintenance.”**



# AWS Lambda : Success Story The Seattle Times

## How it worked

Reference Architecture: [Sample Code](#)



# References

## External

- **Docker** : <https://docs.docker.com/>
- **Microservices** :
- <https://www.mendix.com/blog/the-microservices-you-need-for-devops/>
- <https://www.mulesoft.com/resources/api/what-are-microservices>
- **Kubernetes** : <https://kubernetes.io/docs/concepts/>
- **Kubernetes in 5 mins by VMware**: <https://www.youtube.com/watch?v=PH-2FfFD2PU>
- **AWS Lambda**: [https://www.youtube.com/watch?time\\_continue=1&v=eOBq\\_h4OJ4](https://www.youtube.com/watch?time_continue=1&v=eOBq_h4OJ4)

# SRE & DevOps

## Two sides of same coins

|                | <b>SRE</b>   | <b>DevOps</b>   |
|----------------|--|---|
| Essence        | Set of practices & metrics                                 | Mindset and culture of collaboration                                    |
| Coined         | 2003, by Ben Treynor, @ Google                             | 2009, by Patrick Debois   |
| Goal           | Bridge the gap between Dev & Operation                     | Bridge the gap between Dev & Operation                                  |
| Focus          | Site availability & Reliability                            | Continuity & Speed, Early time to market, stability                     |
| Team Structure | Site reliability engineers with ops and development skills | Wide range of role Product owners, developers, QA engineers, SREs etc., |

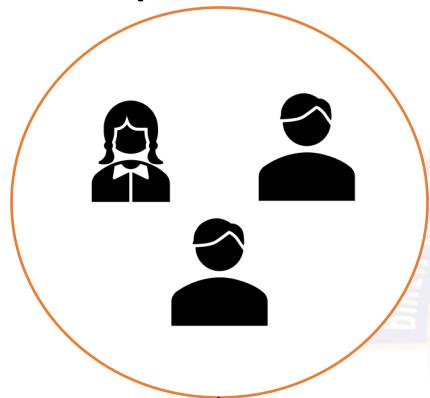
# DevOps & SRE

## Big Picture

Development Team



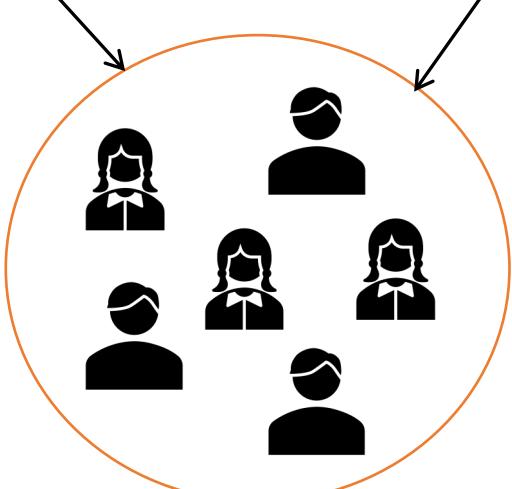
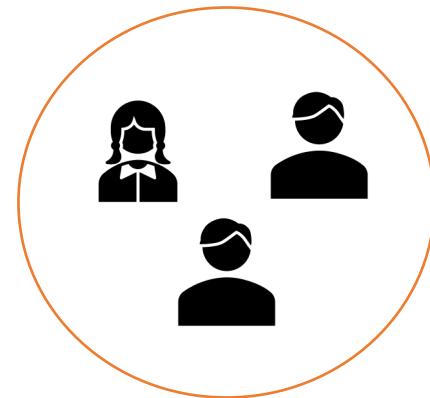
Ops Team



Development Team



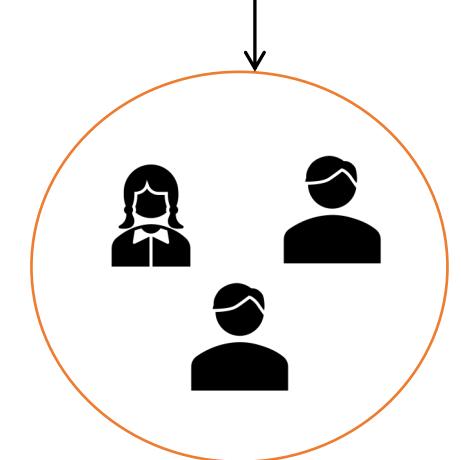
Ops Team



DevOps Team



Development Team



SRE Team



# Q&A



# Thank You!

In our next session: