

BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



Course Name - DevOps for Cloud

Course Number - CC ZG507

Units - 5

Instructor Profile

Dr. Shreyas Rao

- 18+ Years of Experience in IT, Teaching and Research
- Working as Associate Professor (Off Campus), Dept. of CSIS, BITS-Pilani, WILP
- B.E from VTU, M.S in Software Systems from BITS (WILP) and PhD from MAHE
- Worked as Business Analyst and Team Lead at SLK Software Services for 7+ years
- Previously worked in Presidency University and Sahyadri College, Mangaluru as R&D Head, CSE
- Executed 10+ Consultancy projects
- COE member in AI&ML and COE member in Data Science (Govt. Sponsored for 1.2 Cr)

Instructor Profile

Consultancy

- ISRO-SAC (Ahmedabad) funded research project titled “Ontology Enabled Disaster Management Web Service using Data Integration” as Technical Consultant. Deployed in ISRO.

Collaboration with Dept. of Health Innovation, Kasturba Hospital, MAHE

- Telemedicine effectiveness during Covid Wave-I at Kasturba Hospital, Manipal
- Study on psychological implications of COVID-19 on Nursing professionals
- Covid prediction using Patient Discharge Data

*Published papers can be viewed at <https://scholar.google.com.tw/citations?user=MFNrIcAAAAJ&hl=en&oi=ao>

Instructor Profile

Application Development

- Design and Development of AI enabled tool for juvenile self-transformation (Mental Health domain, App Development, Deep Learning & NLP) for Dept. of Psychology, Montfort College.
- SEEC application for MAHE University (Applied patent)
- Grievance Management Portal
- Designed and Developed ‘Dhriti’, a mental health resource Chabot that caters to mental health needs of people during Covid, from the COE in AI&ML, SCEM. Bot is released in Dakshina Kannada region of Karnataka which answers user queries in English, Kannada and Hindi languages. Deployed on the Web and Facebook Messenger channels, used by 5000+ users.

Student Profile

-
- Name
 - Role in Organization (Ex: Developer, Tester, Architect etc.)
 - Exposure to DevOps? Yes / No (If yes, mention Tool used)
 - Experience in any Cloud platform: Yes / No (If Yes, mention Cloud Provider)
 - Did you attend “Scalable Services” course last semester?



CC ZG507 – DevOps for Cloud Lecture No. 1



Walkthrough of Course Handout

Course Objectives

No	Objective
CO1	Understand the DevOps landscape with dimensions such as people, process, and tools
CO2	Explore the principles of Cloud Native DevOps
CO3	Gain proficiency in GitOps practices such as Continuous integration, Continuous Delivery, and Continuous Deployment
CO4	Familiarize with the tools and technologies employed in implementing Cloud DevOps for effective application deployment and management
CO5	Understand the principles and practices involved in DataOps and MLOps



Module Structure

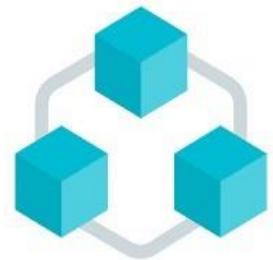
[14 modules]

Module	Description
0	Foundational Concepts
1	Introduction to DevOps
2	Cloud Native Application
3	Source Code Management
4	Continuous Integration
5	Continuous Integration and Continuous Delivery using GitOps
6	Kubernetes
7	Continuous Deployment



Module Structure (contd...)

Module	Description
8	IaC and Serverless CI/CD
9	Security in the DevOps lifecycle
10	Observability and Continuous Monitoring
11	MLOps
12	DataOps
13	Future trends in Cloud DevOps and Course Review



Contact Session wise Coverage

Session	Coverage
CS01	Foundational Concepts and Introduction to DevOps
CS02	Introduction to DevOps (contd...)
CS03	Cloud Native Application
CS04	Source Code Management
CS05	Continuous Integration (Traditional approach)
CS06	Continuous Integration and Continuous Delivery using GitOps
CS07	Kubernetes
CS08	Kubernetes (contd...)
CS09	Continuous Deployment (Traditional approach)
CS10	Continuous Deployment using GitOps
CS11	IaC and Serverless CI/CD
CS12	Security in the DevOps lifecycle
CS13	Observability and Continuous Monitoring
CS14	MLOps
CS15	DataOps
CS16	Future trends in Cloud DevOps & Course Review

Tools

Tool name	Purpose
GitHub	Source Code Repository
GitHub Actions	Continuous Integration with GitOps
ArgoCD	Continuous Deployment with GitOps
AWS DevOps	CodeBuild, CodeDeploy, CodePipeline,
AWS SAM	Serverless Application Model to demonstrate IaC and Serverless continuous deployment
AWS CloudWatch	Continuous Monitoring
Docker Desktop	Docker Engine
MiniKube	Creating Kubernetes clusters
AWS SageMaker	MLOps demonstration
Docker Desktop	Public registries for deploying docker images
ECR (Elastic Container Registry)	

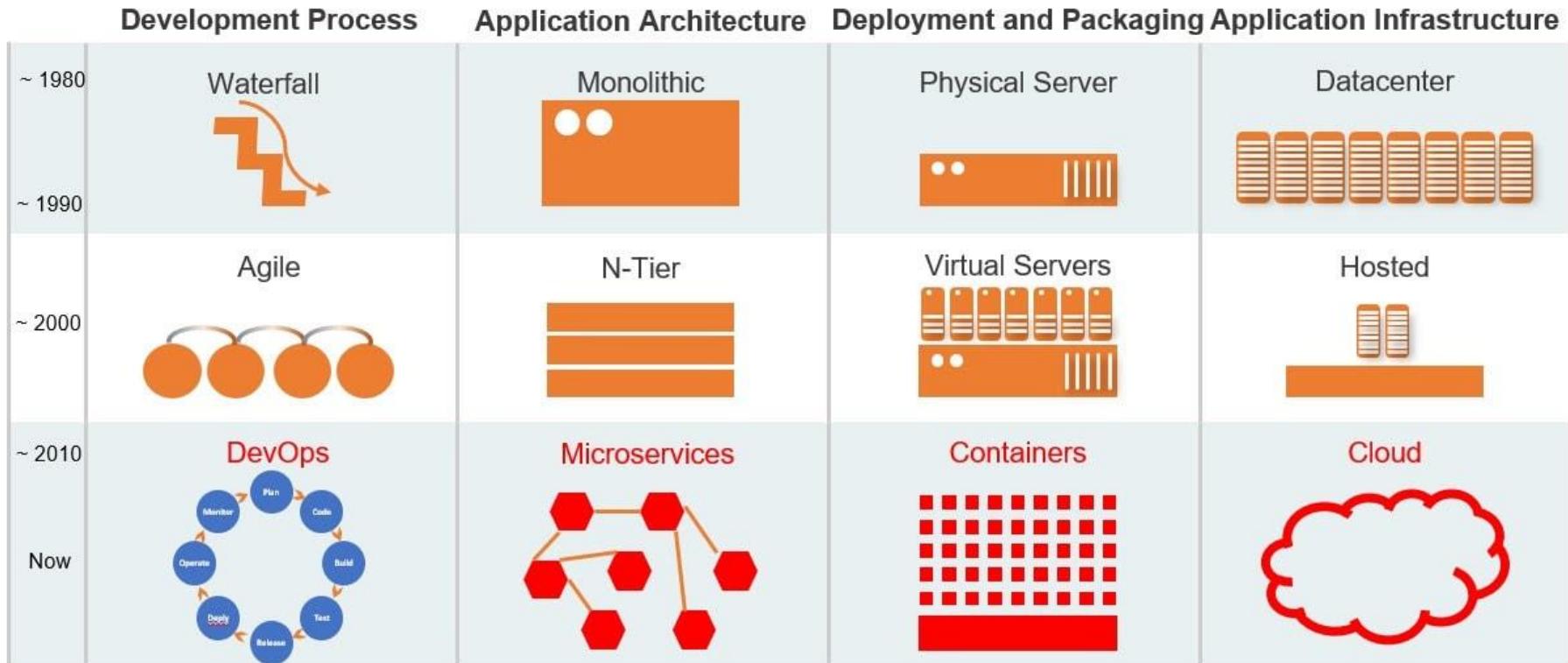
Lab Details (Demo / Hands-On)

1. Demonstrate source code management using GIT
 2. Demonstrate Continuous Integration using GitHub Actions for a Microservice
 3. Demonstrate Continuous Deployment using ArgoCD and Minikube
 4. Demonstrate Continuous Deployment using AWS Serverless Application Model (SAM)
 5. Demonstrate Cloud Monitoring (Metrics, Logs, Alerts, and Traces) using AWS CloudWatch
 6. Demonstrate setting up an Alert for AWS API Gateway using AWS CloudWatch Alarm
 7. Experiments using Docker Desktop, DockerHub and ECR (Elastic Container Registry)
 8. Experiments on Kubernetes using Minikube
 9. Setting up Cluster and CI/CD on ECS
-

Evaluation

Evaluation Component	Name (Quiz, Lab, Project, Midterm exam, End semester exam, etc)	Type (Open book, Closed book, Online, etc.)	Weight	Duration	Day, Date, Session, Time
EC – 1	Quiz 1		5%		February 19-28, 2024
	Quiz 2		5%		March 19-28, 2024
	Assignment I		10%		April 19-28, 2024
	Assignment II		10%		To be announced
EC – 2	Mid-term Exam	Closed book	30%	2 hours	15/03/2024 (FN)
EC – 3	End Semester Exam	Open book	40%	2 ½ hours	17/05/2024 (FN)

Importance of the Course



Shift in the last 6-7 years

1. Shift from Monolith **to Microservices Architecture**
2. Shift from On-premise **to Cloud (AWS, Azure, GCP)** for hosting applications
3. Shift from Virtual Machine based deployment **to Docker/ Kubernetes based deployment**
4. Shift from Traditional CI/CD approaches (Monolith, Cloud-based or Cloud enabled apps, VM based deployment) **To Cloud-native CI/CD [Microservices, Docker/ Kubernetes]**
5. Imperative **to Declarative mode**



Foundational Concepts

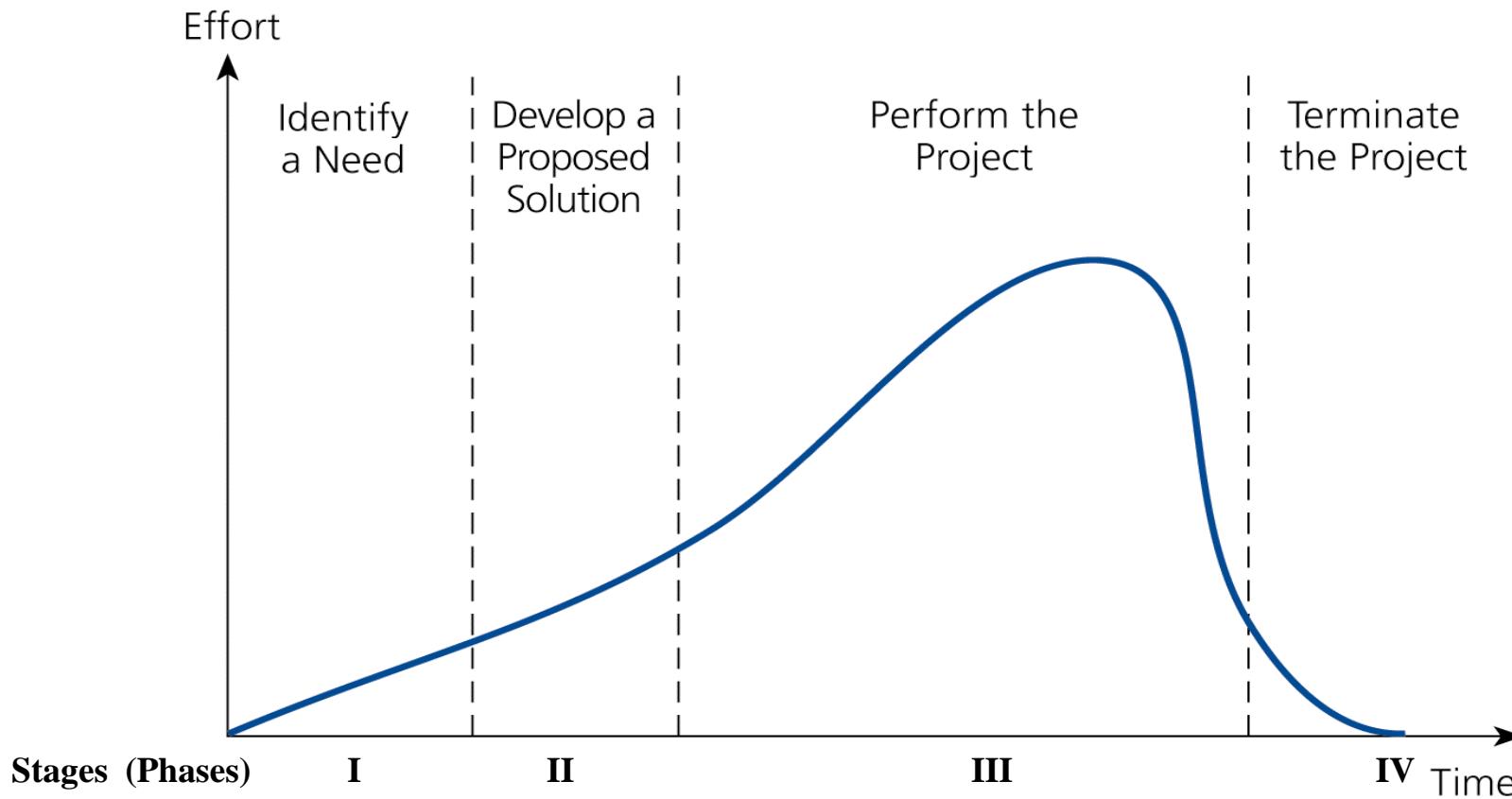
Agenda [Recap/ Review]

1. Software Development Life Cycle
 2. Process Models (before Agile)
 3. Agile Process Model
 4. Agile Methodologies – Scrum
 5. TDD and FDD in Agile context
-

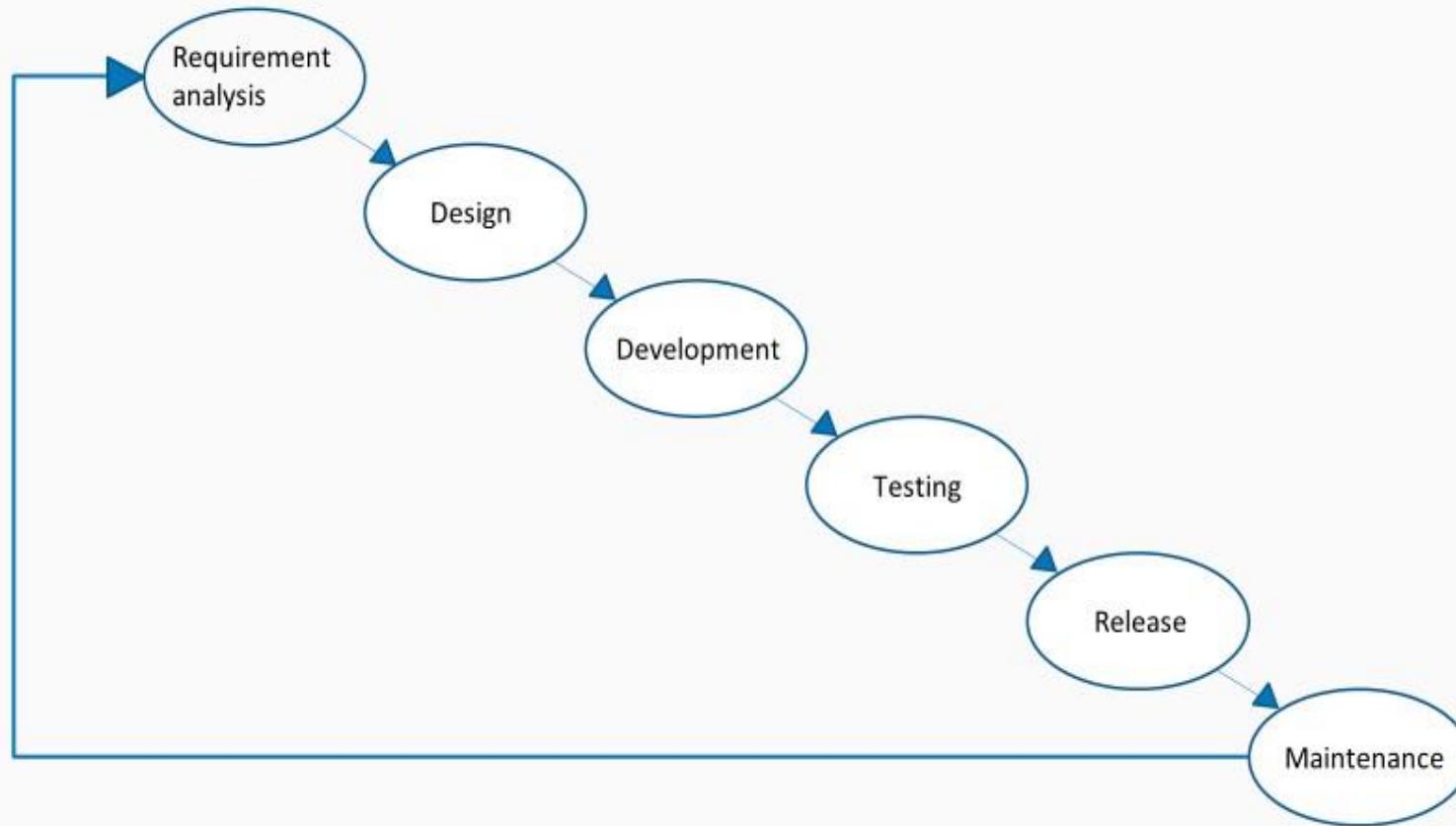


Software Development Life Cycle (SDLC)

Typical Project Life-Cycle (4 Phases)



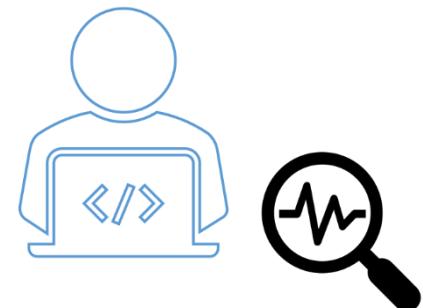
Software Development Life Cycle (SDLC)



Requirement Analysis



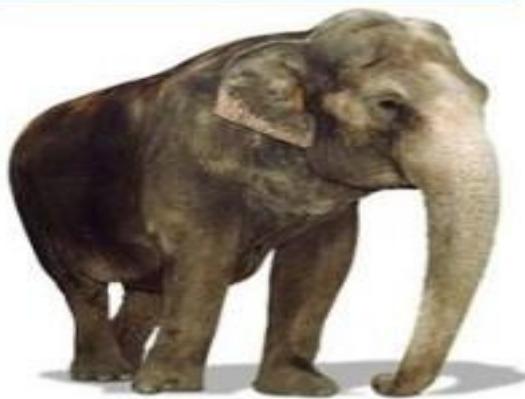
- Capture the requirements / features of the application
- Encounter majority of problems
- Find common language between people outside of IT and people in IT (through SRS - Software Requirement Specification)
- Leads to different problems around terminology (Domain terminologies)
- Phase to capture the Business flow
- Iterative approach



Requirement Analysis



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 😊

Design



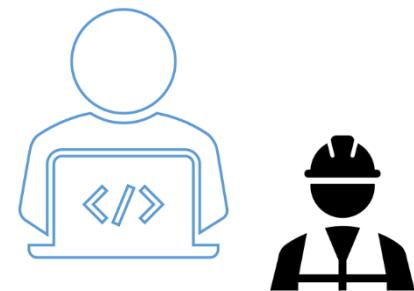
- Design our flows in language that IT crowd can understand straight away
[UML - Visio]
- Overlaps with requirement analysis
- Desirable as **diagrams** are perfect middle language that we are looking for
- Minimal Value Product
- May include Storytelling via Prototypes



Development



- Software is built
- Build technical artifacts according to specification and design
- “Deliver early and deliver often” is mantra followed to minimize impact of wrong specification
- No matter what we do, our software has to be modular so we can plug and play modules in order to accommodate new requirements



Testing

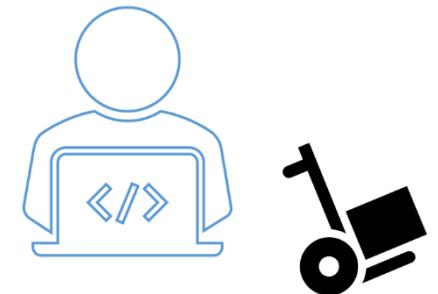
- Finding defects in the application
- Unit testing, Integration testing, System testing, User Acceptance testing
- Manual or Automated testing



Release / Deploy



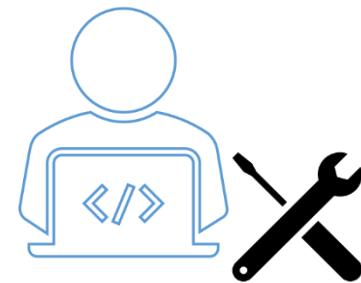
- Deliver software to production environment
- Enables developers to execute build-test-deploy cycle very quickly
- Deploy = Release



Maintenance



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





Process Models (before Agile)

Software Process Models

Software Process (SP) is a **framework** for the activities, actions, and tasks that are required to build high-quality software.

Typical activities include: Communication, Planning, Modelling, Construction and Deployment

Prescriptive models are used as guidelines to organize and structure how software developmental activities should be performed, and in what order.

The name 'prescriptive' is given because the model prescribes a set of activities, actions, tasks, quality assurance and change the mechanism for every project.

The Waterfall Model

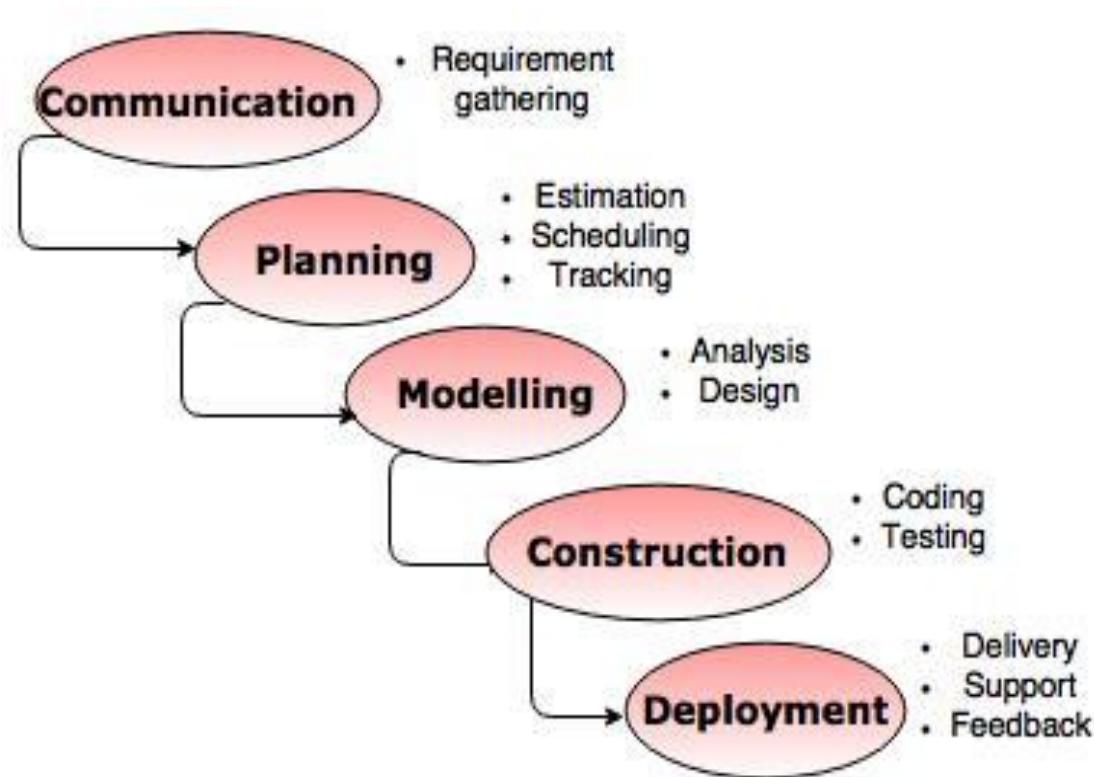
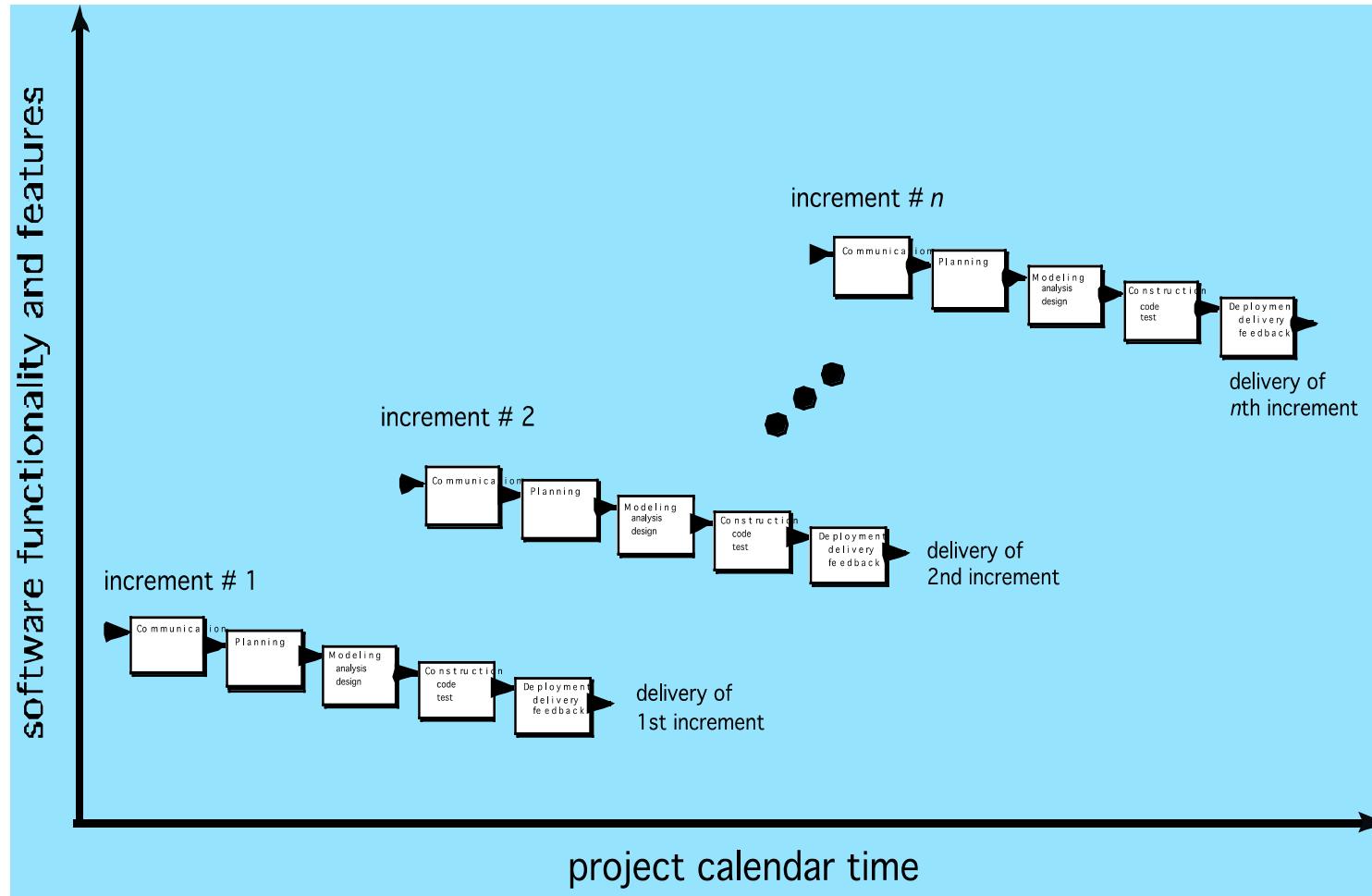


Fig. - The Waterfall model

The Waterfall Model

- The waterfall model is also called as '**Linear sequential model**'.
- In this model, *each phase is fully completed* before the beginning of the next phase.
- This model is used for small projects.
- Assumes that requirements are defined and reasonably stable.
- Testing part starts only after the development is complete.

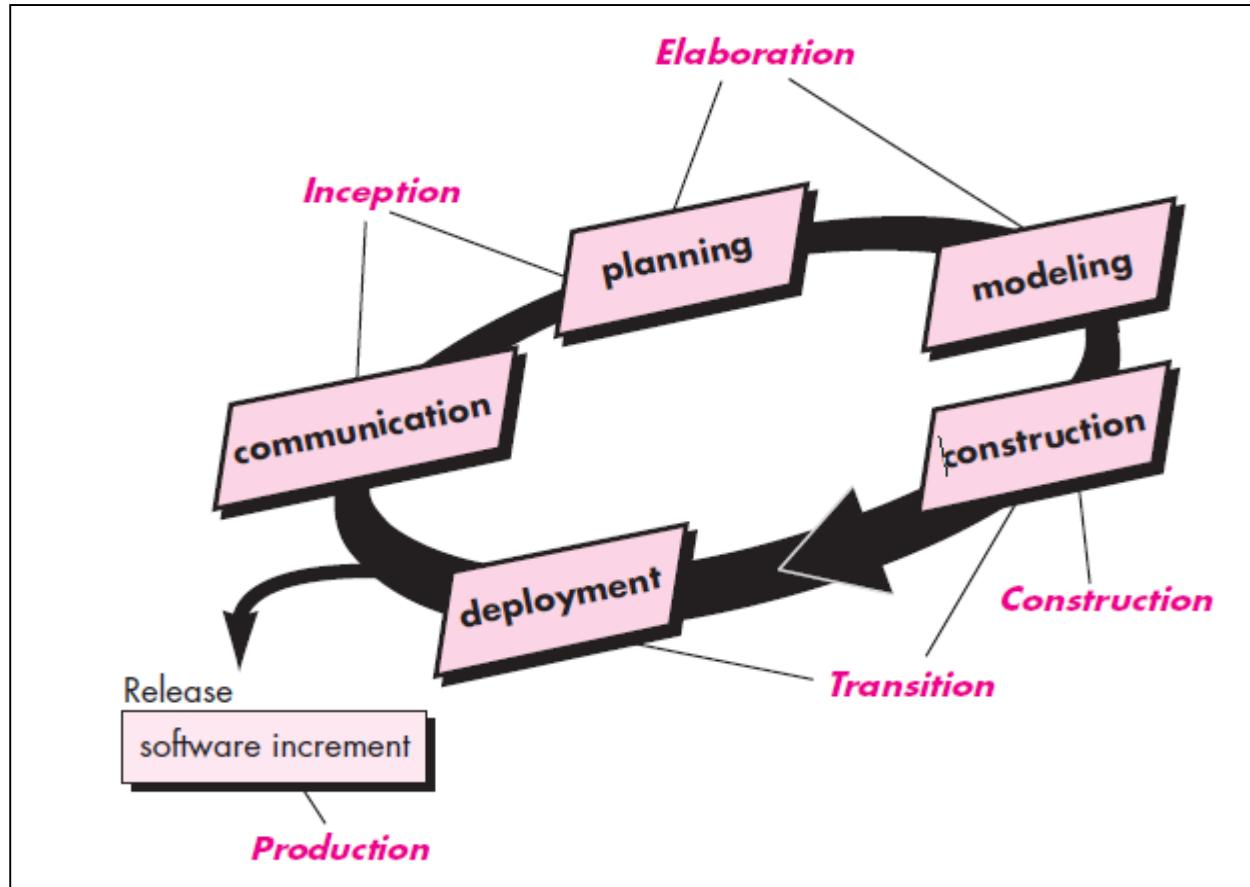
Incremental Model



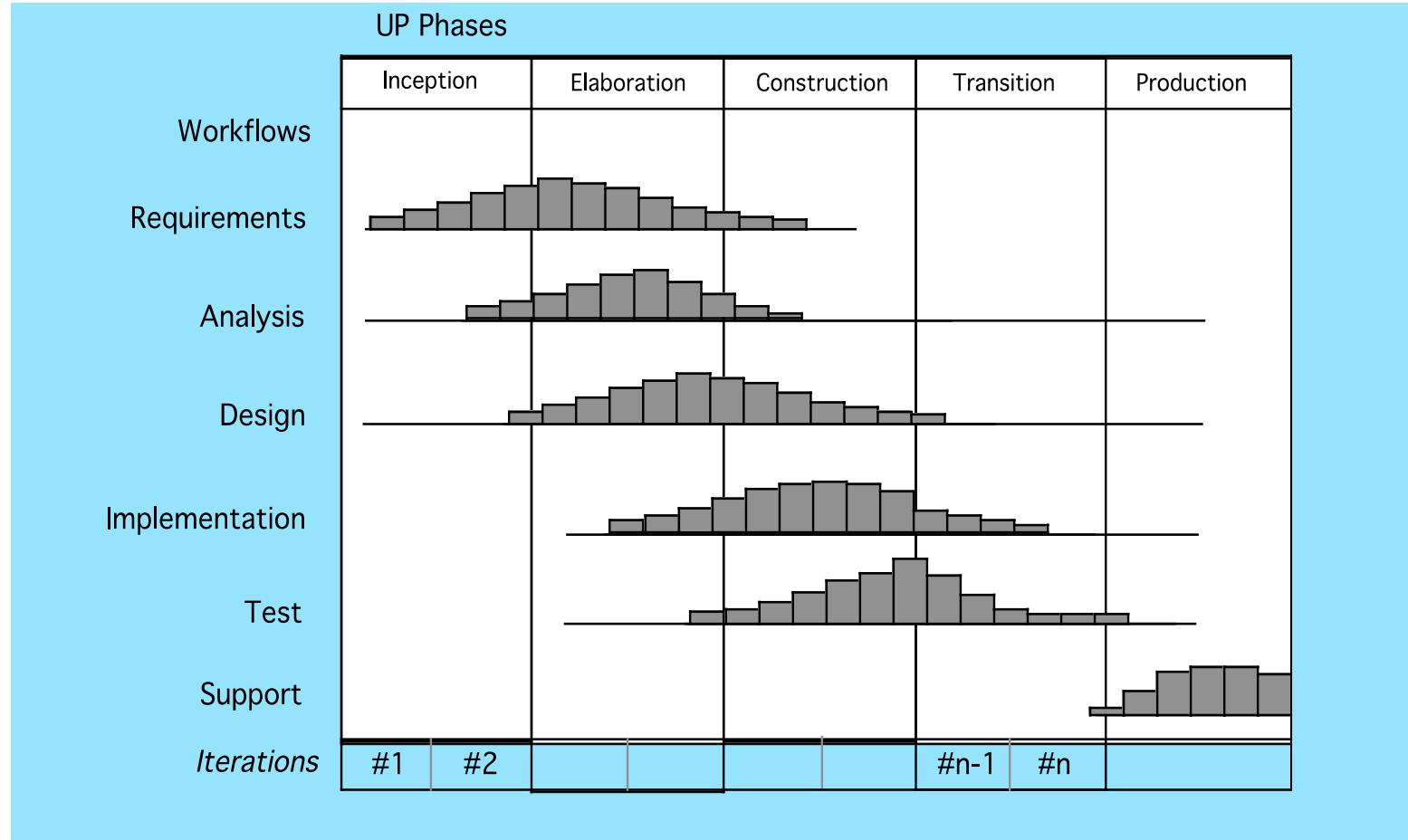
Unified Process Model

- The **Unified Approach (UA)** is a methodology for software development that is proposed by the author Ali Bahrami (1999).
- The UA, based on methodologies by Booch, Rumbaugh, and Jacobson, tries to **combine the best practices**, processes, and guidelines along with the Object Management Group's **Unified Modeling Language (UML)**.

Unified Process Model



UP phases





Agile Process Model

Why Agile?

- The project will produce the **wrong product** [after long months / year effort]
- The project will produce a product of **inferior quality** [not meeting expectations]
- The project will be **late**
- We'll have to work 80 hour weeks
- We'll have to break commitments

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

Agile Background

- Agile was formally launched in 2001
- 17 technologists drafted the Agile Manifesto





The Manifesto for Agile Software Development

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions* over processes and tools
- *Working software* over comprehensive documentation
- *Customer collaboration* over contract negotiation
- *Responding to change* over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al

12 Principles of Agile Methodology

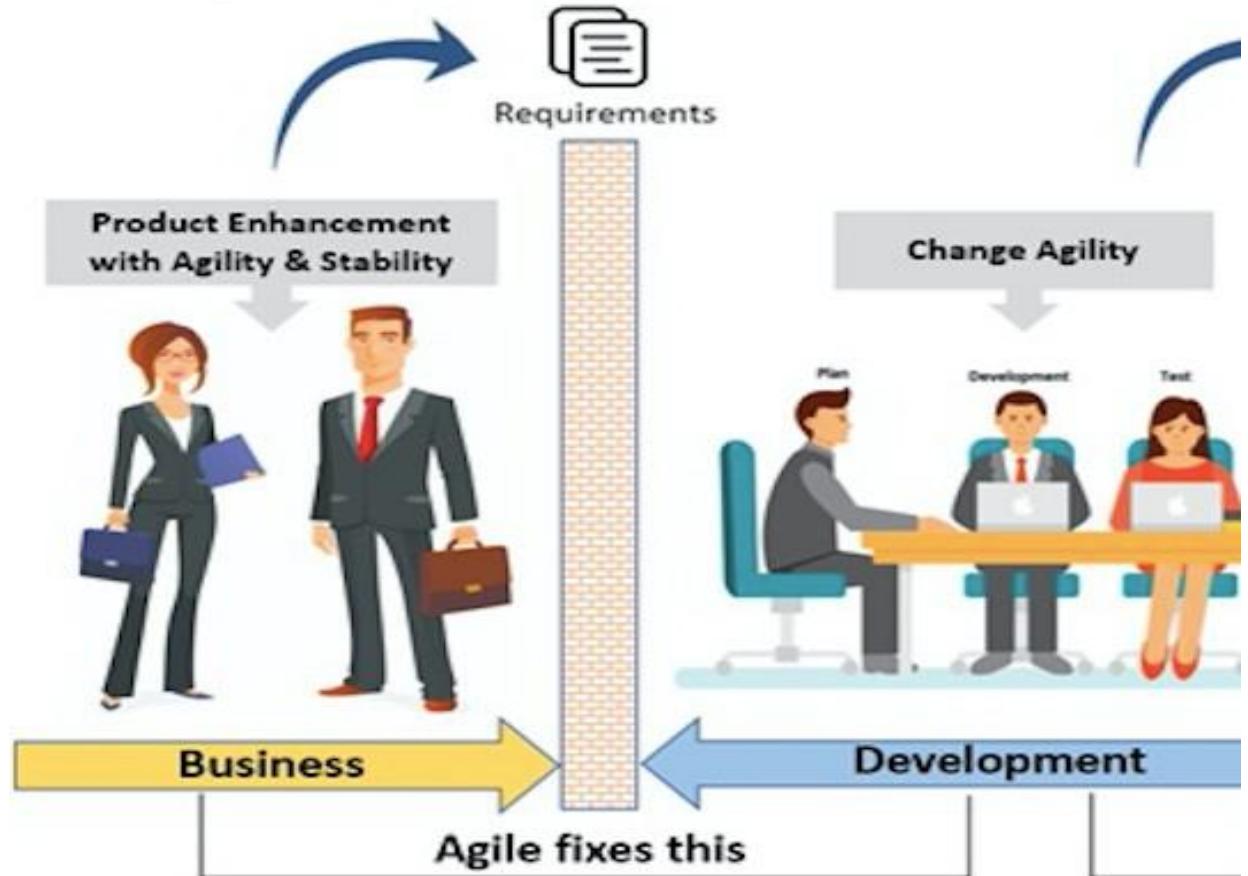
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage
3. Deliver working software frequently, from a couple of weeks 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

*self-organizing team -> cross-functional team, collaboration (dev, test, ops), ownership, focus on continuous delivery

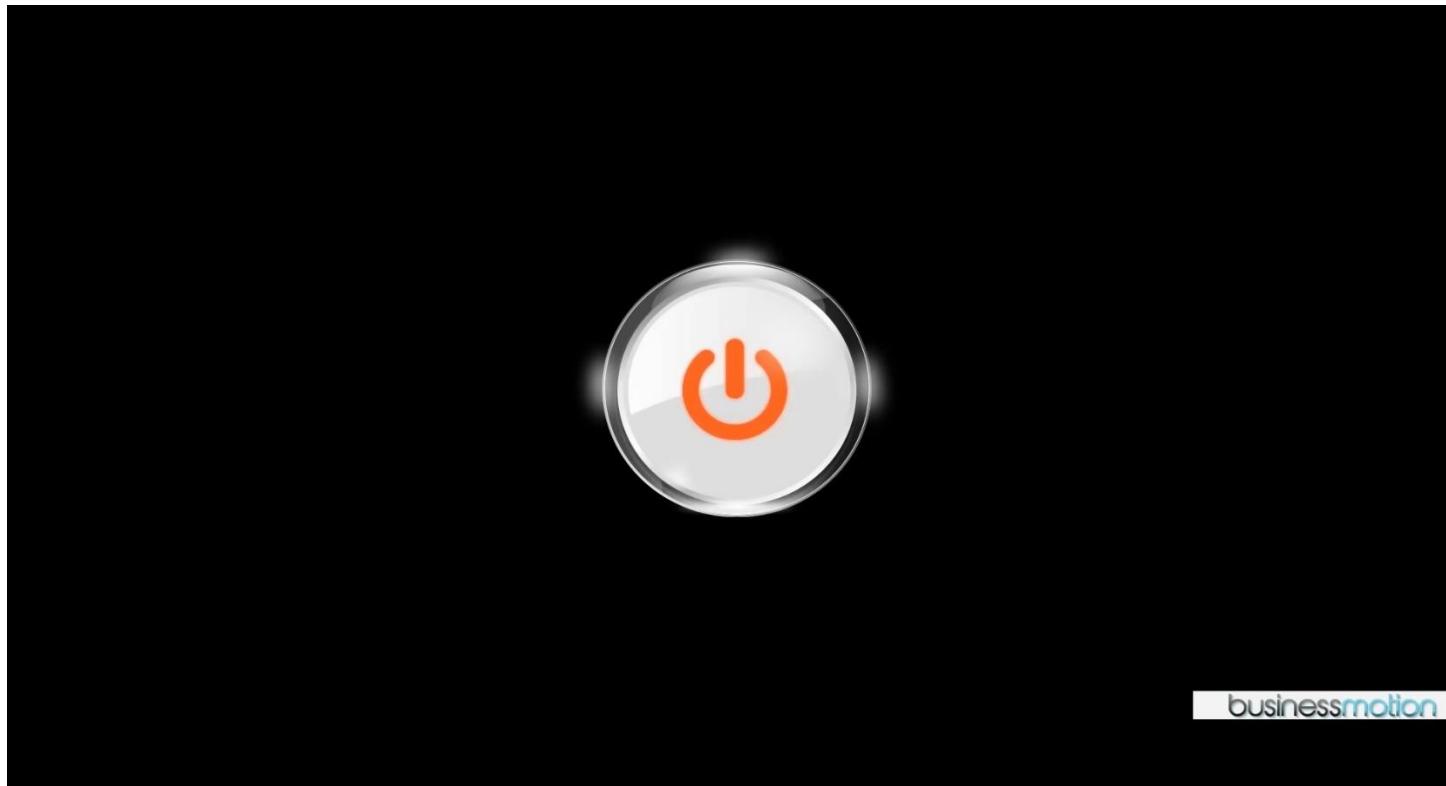
Importance of Agile

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



How Agile actually works



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



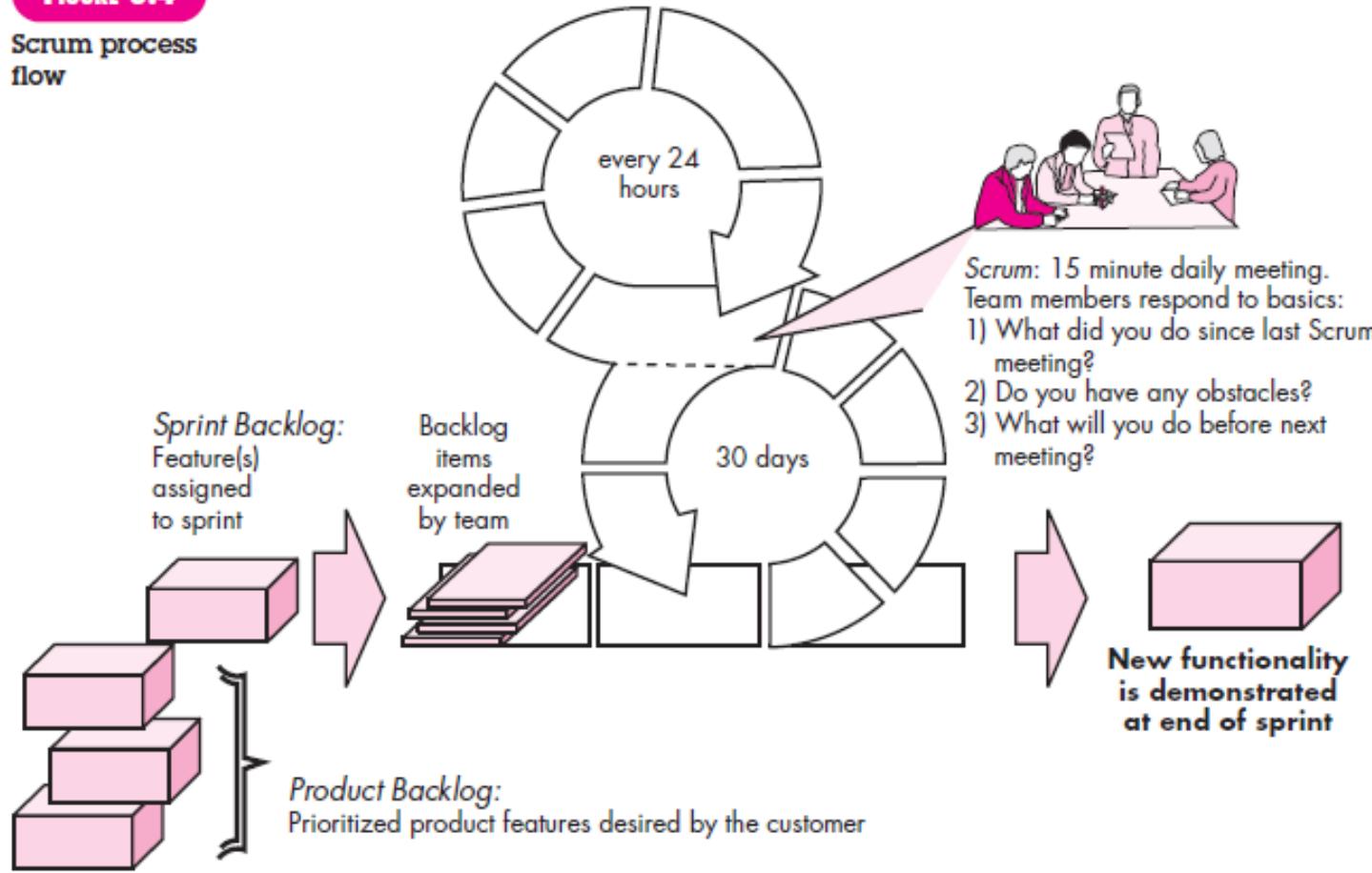
Agile Methodologies

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

SCRUM

FIGURE 3.4

Scrum process flow



Scrum Terminologies

1. Product Backlog

- Prioritized list of features - enhancements or bug fixes

2. Sprint Planning

- Product Owner and Scrum Master select subset of items from product backlog to work for the sprint (~2-4 weeks).
- Team commits to deliver the potentially shippable product increment

3. Sprint Backlog

- Tasks / Timelines for the current sprint are estimated and team commits

4. Daily Stand-ups

- Discuss progress, challenges and plans for the day
- Answer three qns -> What did I do yesterday, What will I do today. Any impediments for my progress?

5. Sprint Review

- At the end of sprint, team will showcase work to stakeholders and feedback is gathered

Scrum Team

1. Product Owner

- Prioritizes product backlog
- Represents interest of stakeholders
- Decides what features are developed in Sprint

2. Scrum Master

- Facilitates the scrum process
- Helps team remove impediments
- Ensures that scrum practices are followed

3. Development Team

- Cross-functional and self-organizing
- Usually consists of developers, testers, designers
- Commits to completing tasks in the Sprint Backlog

4. Stakeholders

- Anyone with an interest in the project, including customers, end-users, and managers
- Provides feedback during Sprint Review

Scrum - Team



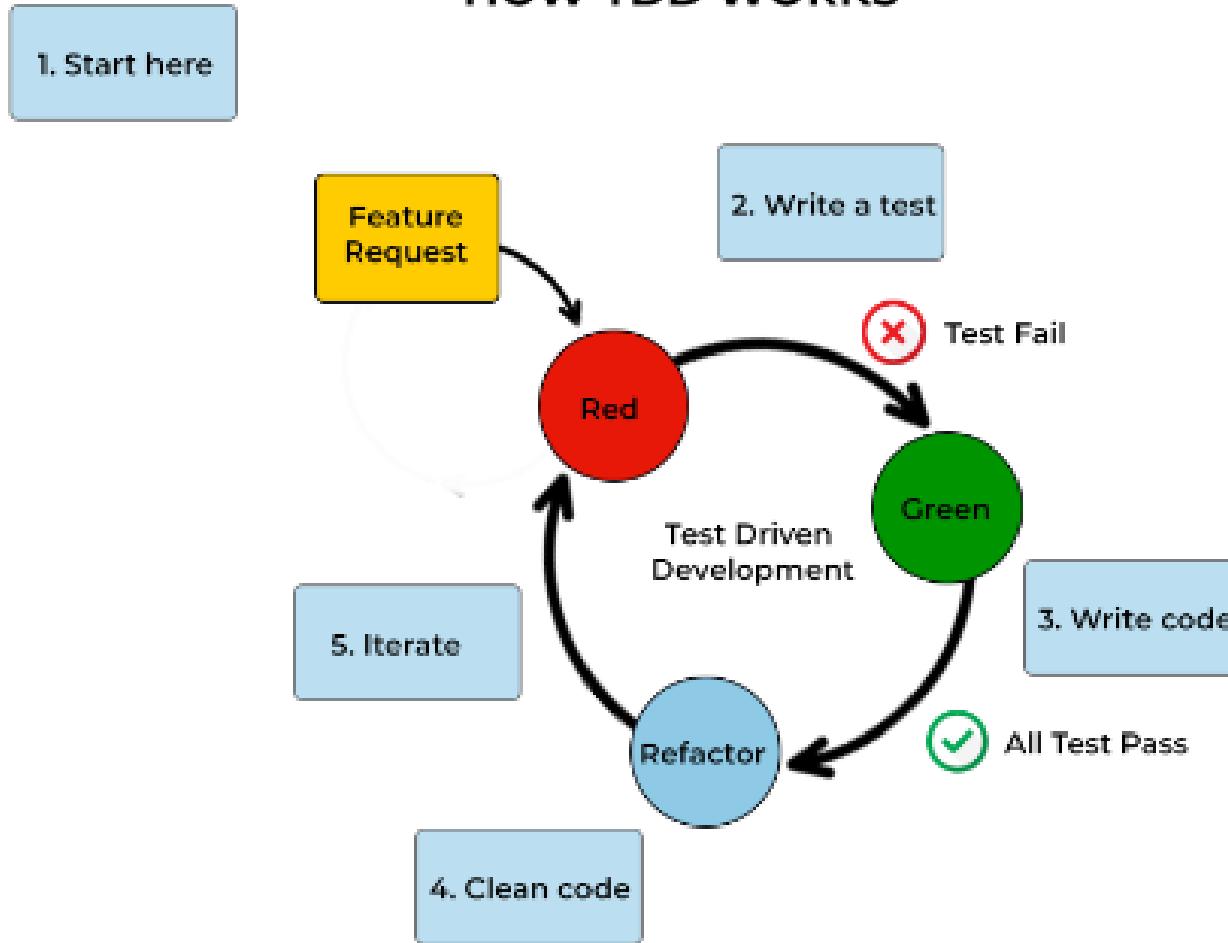
<https://jessefewell.com/wp-content/uploads/2014/07/scrum-responsibilities-raci.png>

Test Driven Development

- Test-Driven Development (TDD) is a software development approach where tests are written before the actual code.
- The process involves a cycle of writing a test, writing the minimum amount of code to pass the test, and then refactoring the code to improve its structure without changing its behavior.

Test Driven Development

HOW TDD WORKS



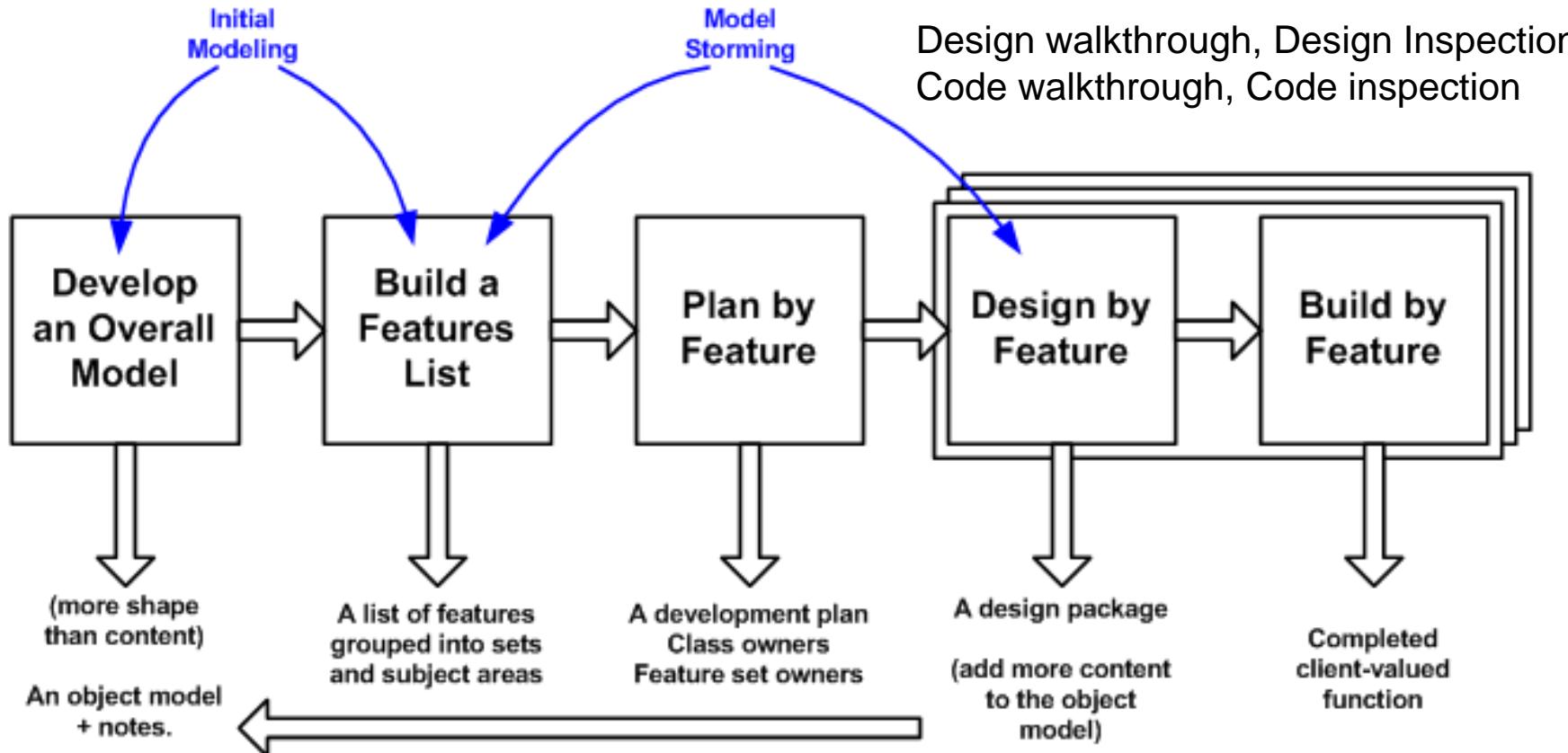
Feature Driven Development

- FDD shares some common principles with Agile methodologies, such as
 - Iterative development
 - Incremental delivery
 - Adaptability to change
- FDD can be considered as a framework that is compatible with Agile principles.

Feature Driven Development

- Originally proposed by Peter Coad et al
- FDD - distinguishing features
 - Emphasis is on defining “**features**”
 - a **feature** “is a client-valued function that can be implemented in two weeks or less.”
 - Feature is a *small block of deliverable functionality*
 - Uses a **feature template**
`<action> the <result> <by | for | of | to> a(n) <object>`
 - Ex 1: Add the Product to shopping cart
 - Ex 2: Display the technical specifications of the product
 - Ex 3: Store the shipping information for the customer

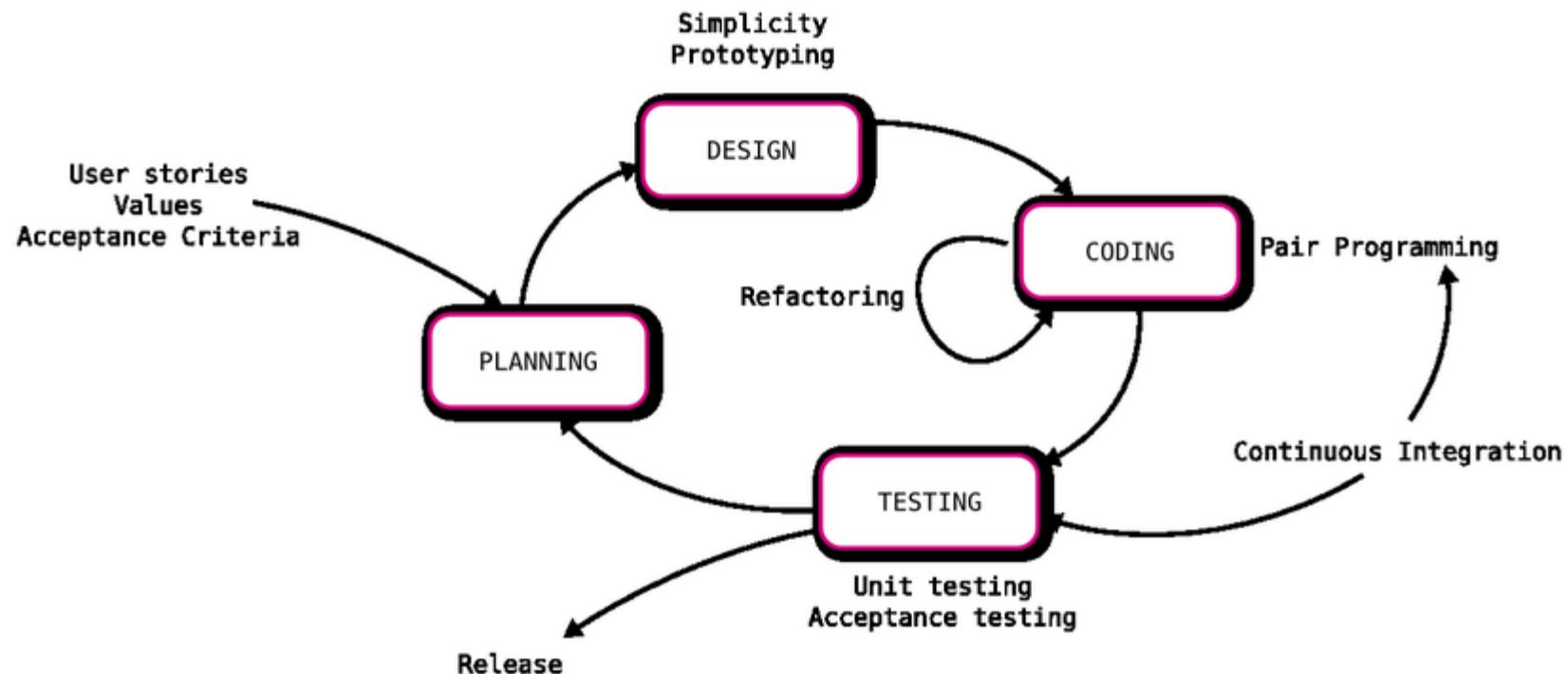
Feature Driven Development



Copyright 2002-2005 Scott W. Ambler
 Original Copyright S. R. Palmer & J.M. Felsing

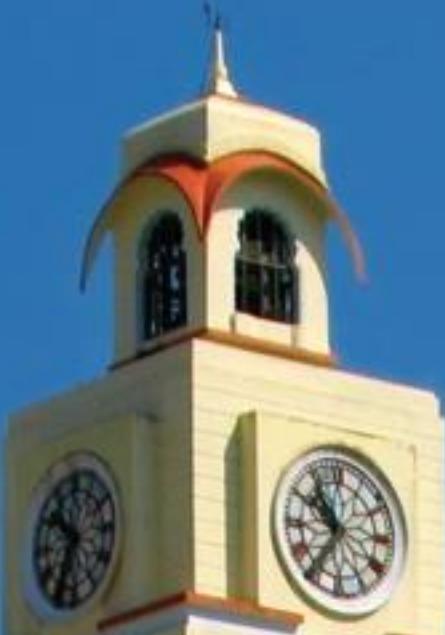
Extreme Programming

- XP emphasizes customer satisfaction, continuous feedback, and the ability to accommodate changing requirements. Encourages TDD.
- It was introduced by Kent Beck in the late 1990s.





Thank You!



BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lecture No. 2

Agenda

-
- Need for DevOps
 - What is DevOps
 - Evolution of DevOps
 - DevOps Lifecycle
 - Three Dimensions of DevOps - People, Process, Tools
 - Key DevOps Practices - CI, CT, CD, CM
 - Agile and DevOps
 - Overview of Xops
 - Benefits of DevOps
-



Introduction to DevOps

Need, Definition and Evolution

Need for DevOps

Achieve Faster Deployment Cycles

(Average 500+ micro deployments per day)

COMPANY	DEPLOY FREQUENCY	DEPLOY LEAD TIME	RELIABILITY	CUSTOMER FEEDBACK
AMAZON	23.000/day	minutes	high	high
GOOGLE	5.500/day	minutes	high	high
NETFLIX	500/day	minutes	high	high
FACEBOOK	1/day	hours	high	high
TWITTER	3/week	hours	high	high
TYPICAL ENTERPRISE	once every 9 months	months or more	low/medium	low/medium

From "The Phoenix Project"

Need for DevOps

- Development and Deployment Needs of Latest Software Apps
- Enable Concurrent Development Activities (Across the Globe)

Facebook Live Video Center (20k+ developers work on FB)



Need for DevOps

Accommodate diverse Developer Languages, Tools and Technologies

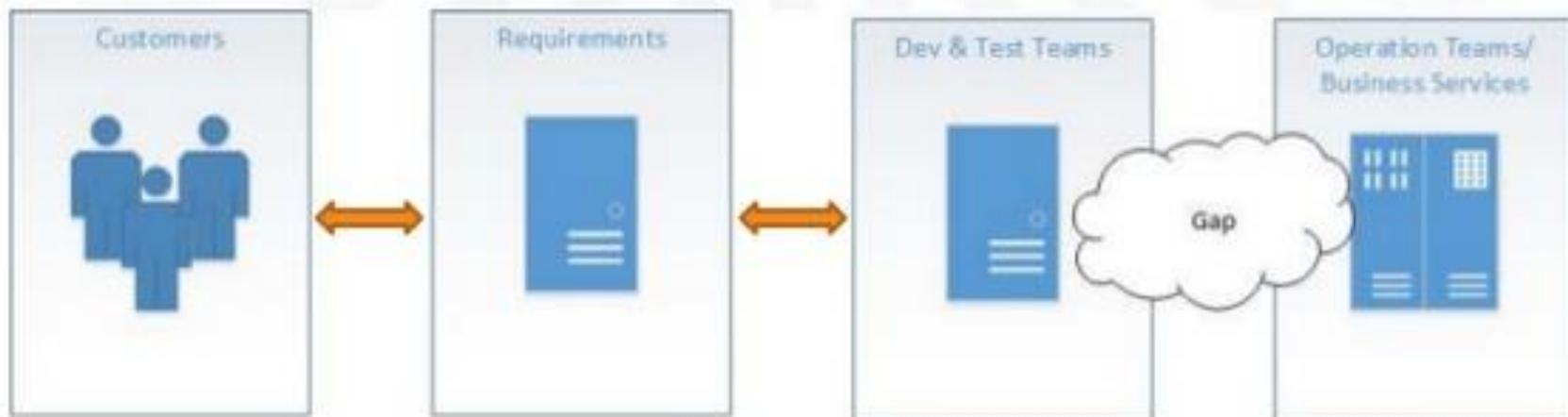
(Ex: Instagram is built using Python, React Native, JavaScript, Django, PostGreSQL, Objective-C, Nginx Web Server, Redis in-memory database etc.)



Fig. Agile Tools (Vikash Karuna)

Scenario before DevOps

Why DevOps? – Delivery Challenges



Dev and Ops Dialogue

Dev:

- Put the current release live, NOW!
- It works on my machine
- You are using the wrong version



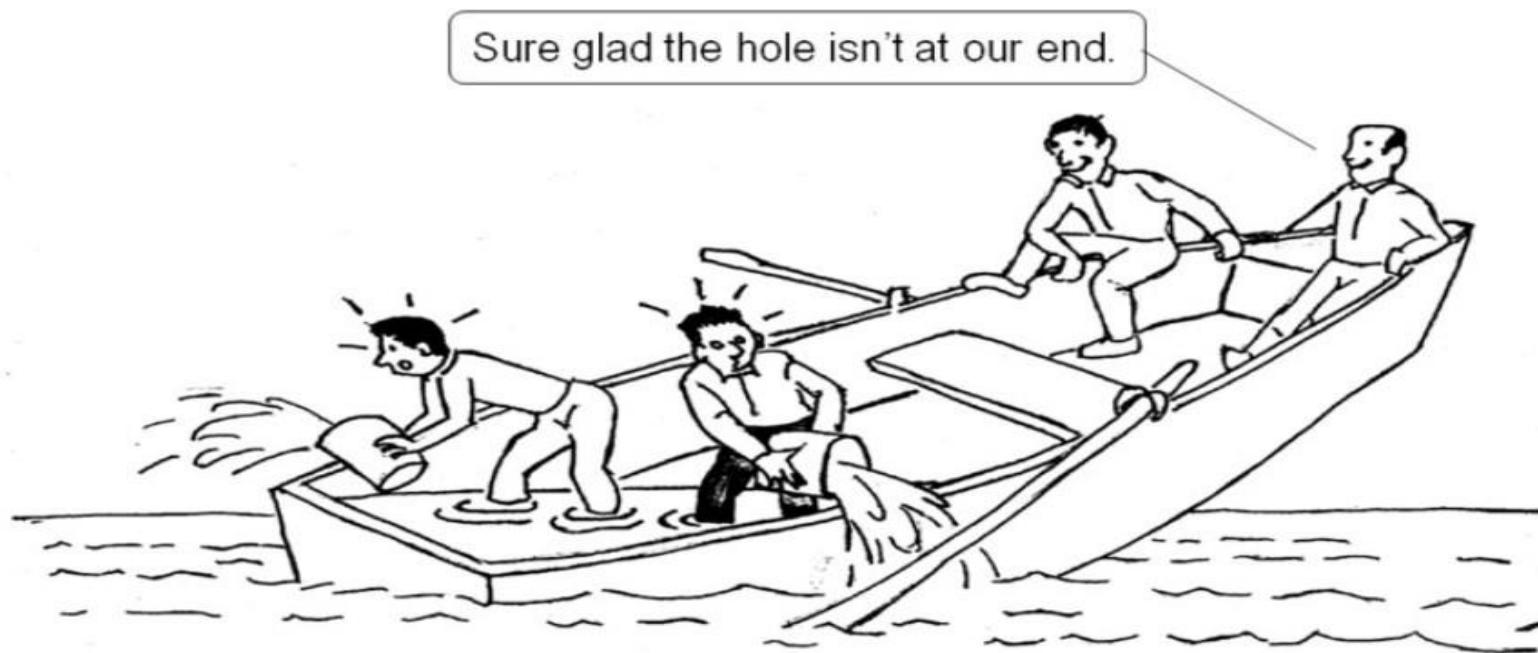
Ops:

- What are the dependencies?
- No machines available...
- Which DB?
- High Availability?
- Scalability?



Scenario before DevOps

Blame Game



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

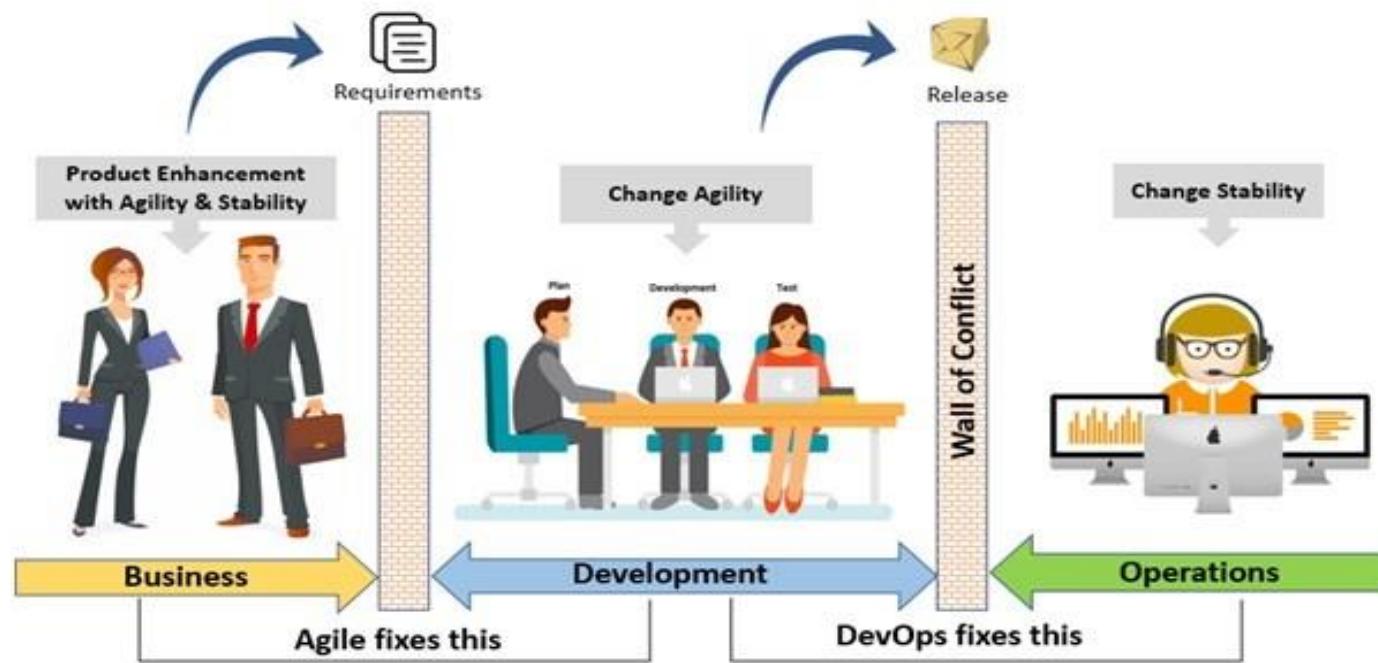
Manual Configuration Management of Production Environments

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

What does DevOps Fix?

Agile fixed the communication and collaboration between the Business and Development teams

DevOps fixes the communication and collaboration between the Development and Operations teams



DevOps - Definition

1. DevOps is the process of alignment of IT Development and Maintenance Operations with better and improved communication.

2. Microsoft defines DevOps as “Union of People, Process and Products to enable continuous delivery of value to the customers”

People -> Culture

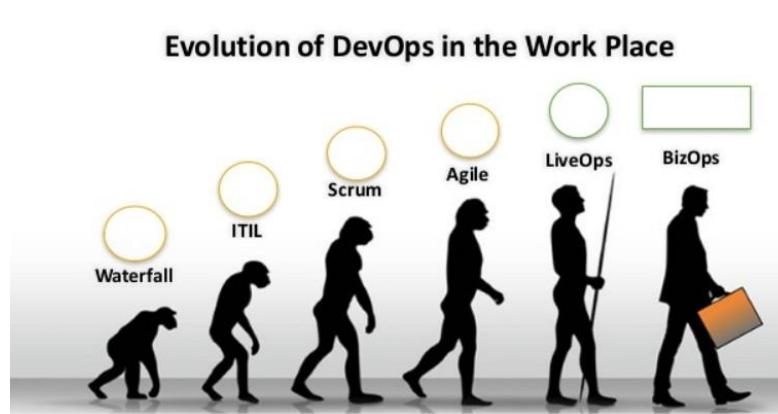
Process -> DevOps Methodology

Products -> Tools

Evolution of DevOps

History

- Back in 2007
- Patrick Debois [Belgian Engineer] through “DevOpsDays” conference
- Initially it was “Agile Infrastructure” but later coined the phrase DevOps
- Many tangential DevOps Initiative
 - WinOps (implement DevOps for Windows centric ecosystem)
 - DevSecOps (integrates security practices in DevOps pipeline)
 - BizDevOps (Automation of business processes + Dev + Ops integration)
-





Introduction to DevOps

Dimensions and Practices

DevOps Dimensions

1. People (Culture)

2. Process

3. Tools



DevOps Dimensions



1. People -> DevOps Culture



Waterfall, Agile, Devops

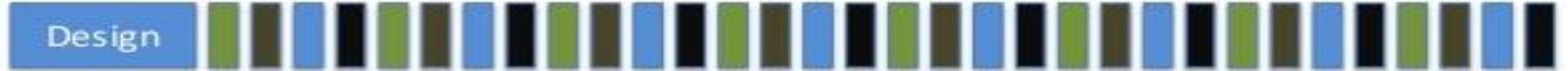
- Waterfall



- Agile

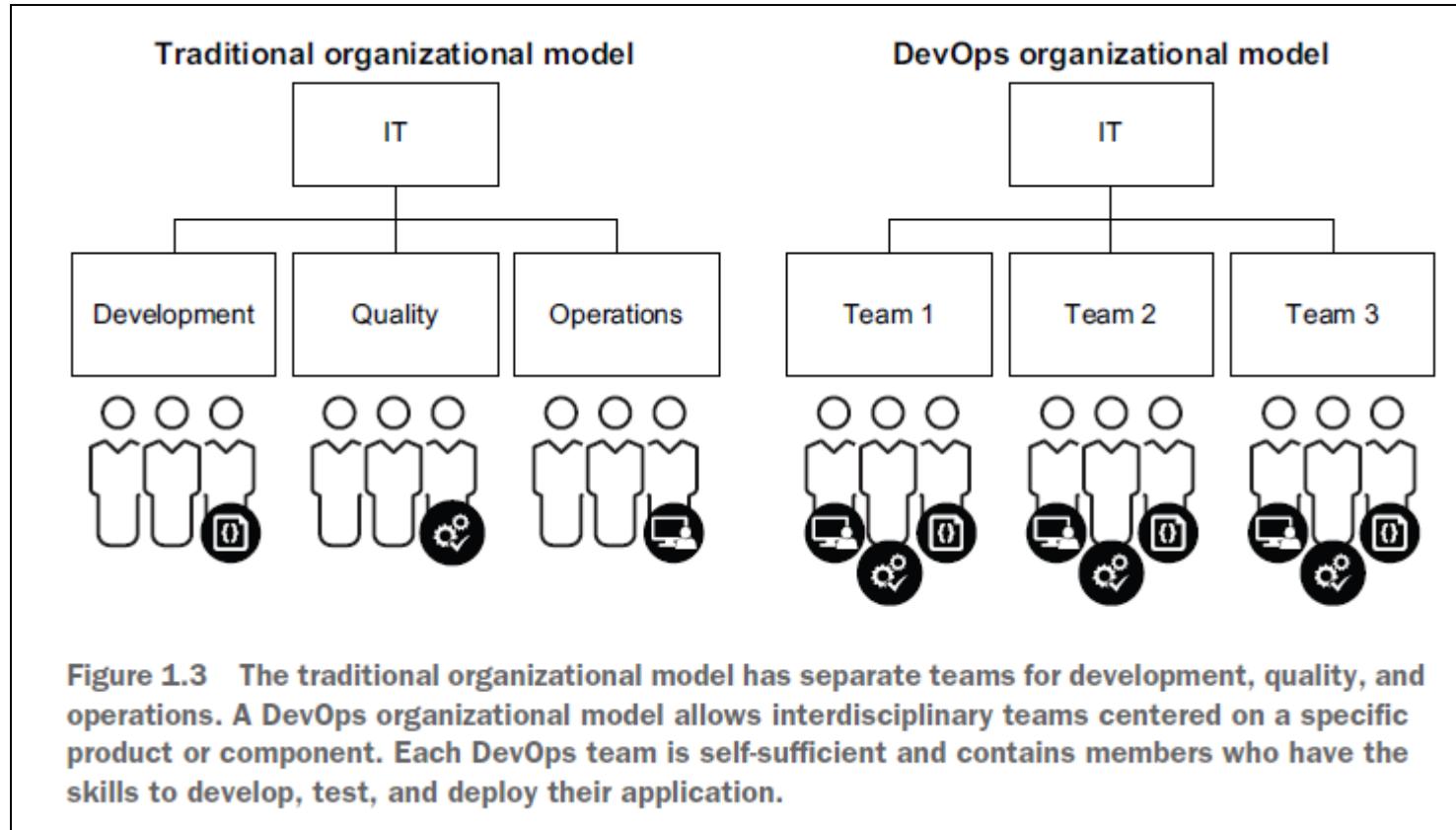


- Agile with Continuous Deploy



Continuous Deploy requires DevOps

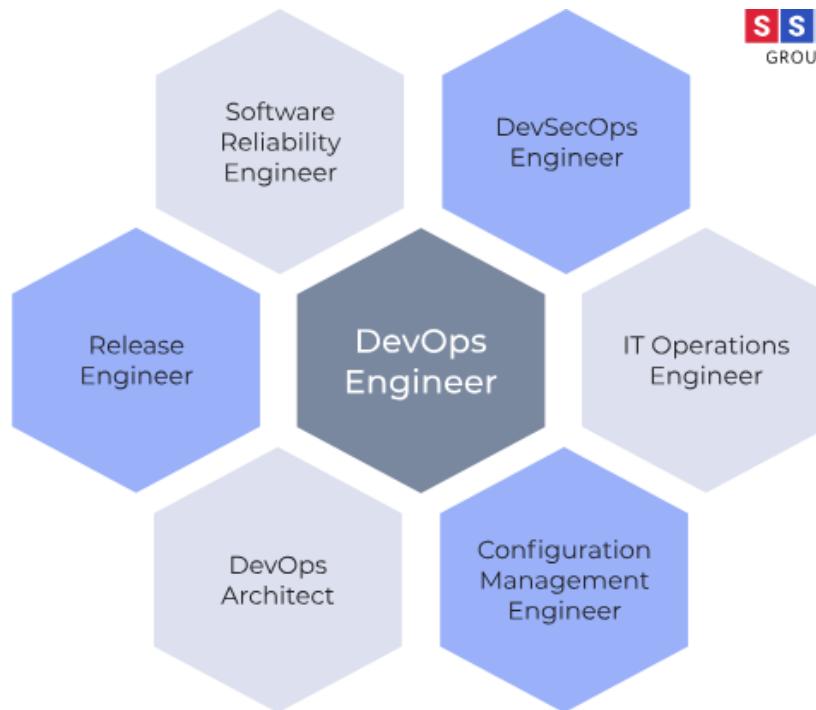
1. People -> Team Formation



1. People -> Key Roles

1. DevOps Engineer

- Manage CI/CD process
- Assessing and monitor performance
- Configuration and maintaining infrastructure
- Security



2. DevOps Architect

- Design and Implement DevOps Strategy
- Tool selection
- Architecting CI/CD pipelines
- Consults on environment for Deployment

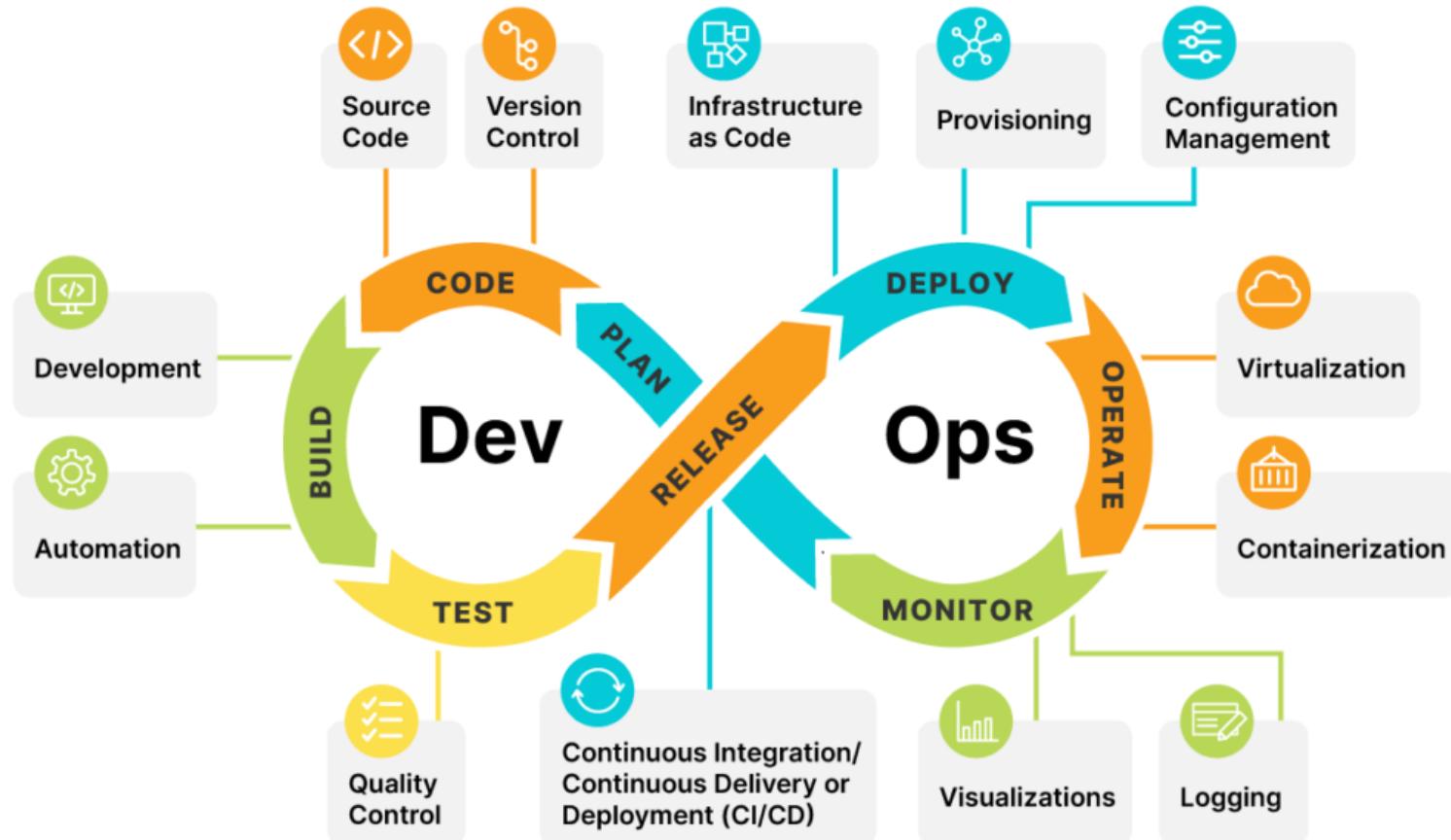
3. SRE

- Responsible for reliability, scalability of services
- Monitoring & Alerting
- Incident response
- Disaster Recovery

4. Release Engineer

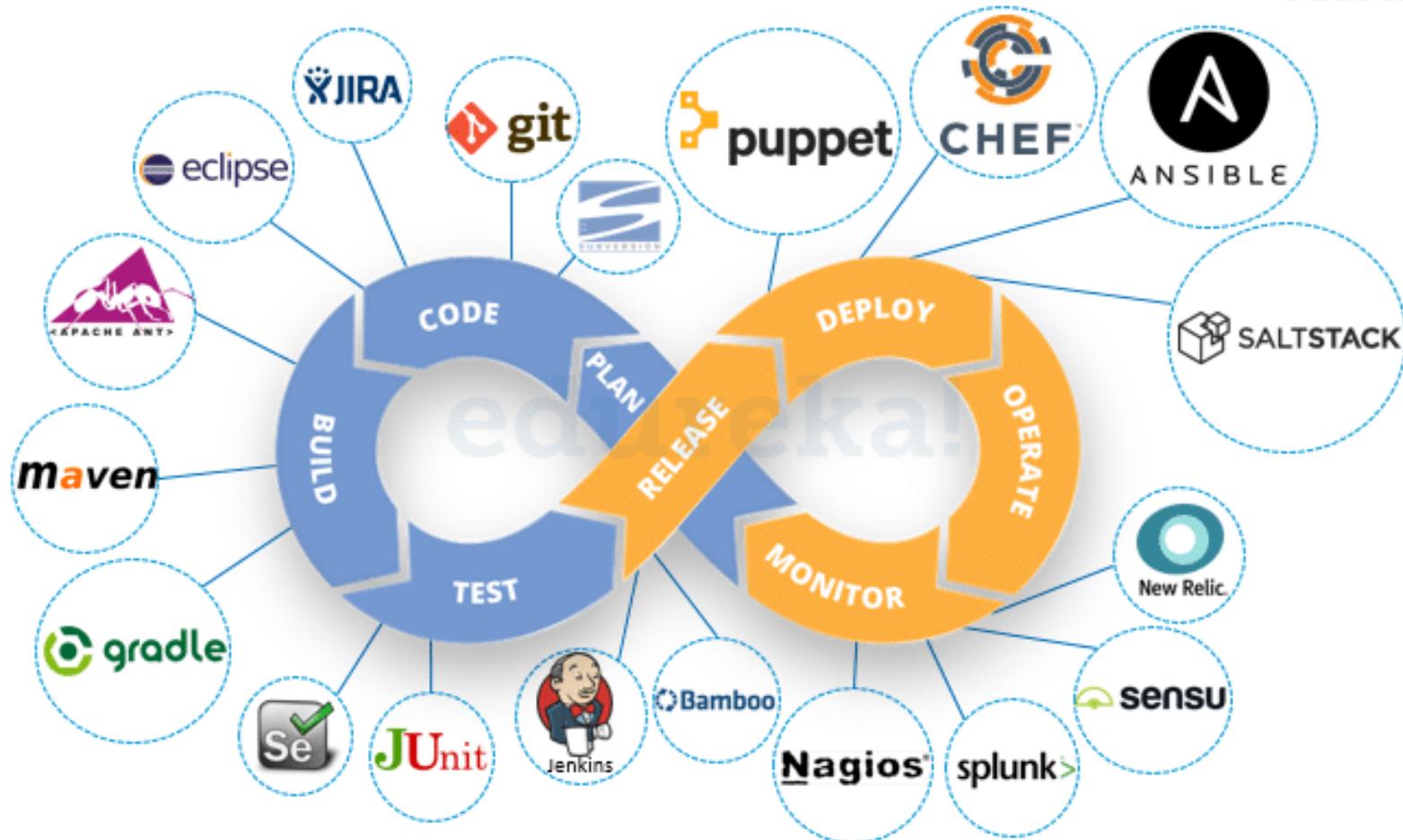
- Plan release schedule and milestones
- Responsible for Build Automation (CI part)
- Continuous Improvement

2. Process -> DevOps Lifecycle

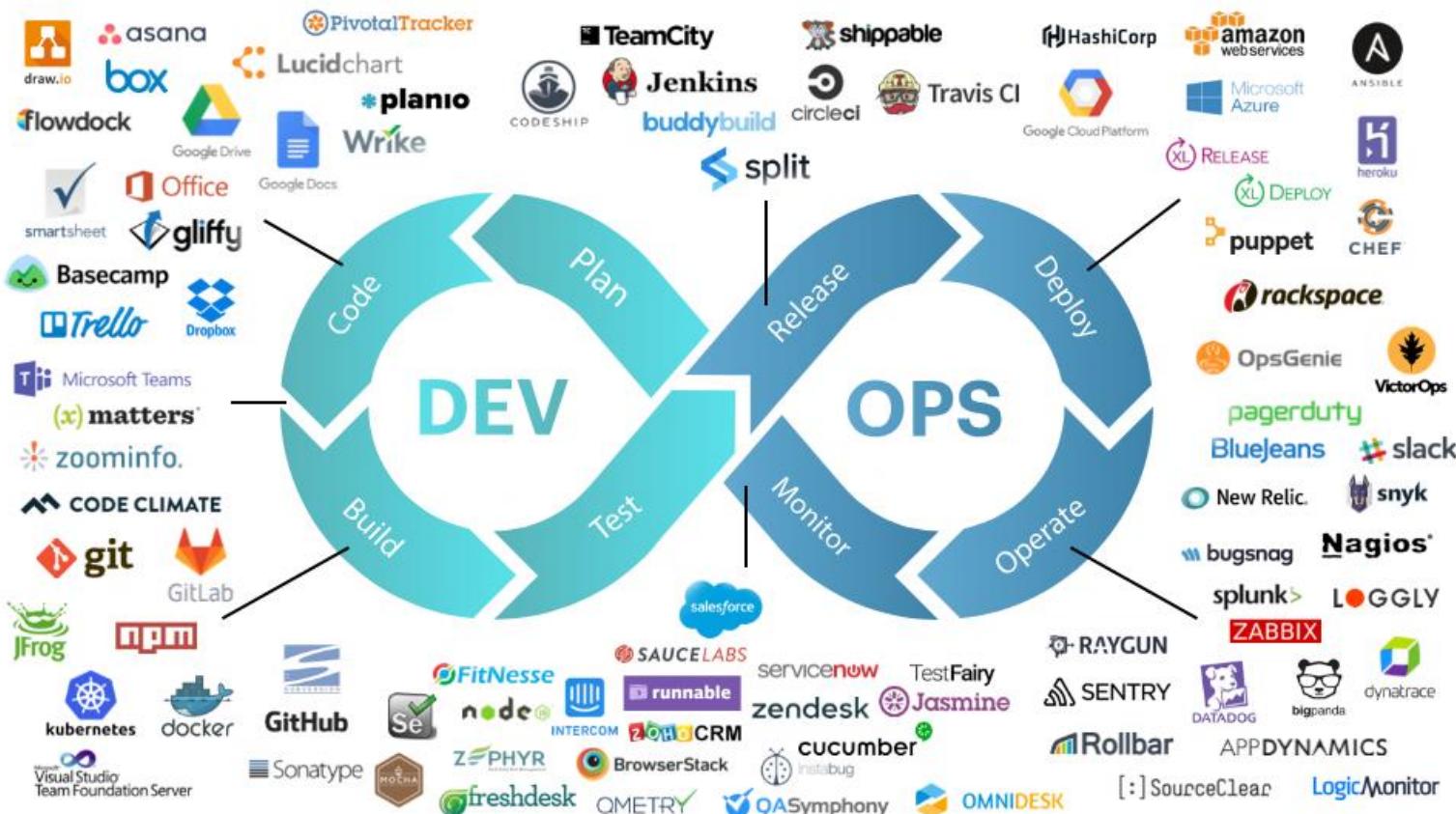


<https://basanagouda.medium.com/day-1-getting-started-with-devops-b78ccca0af8>

3. Tools (Brief)



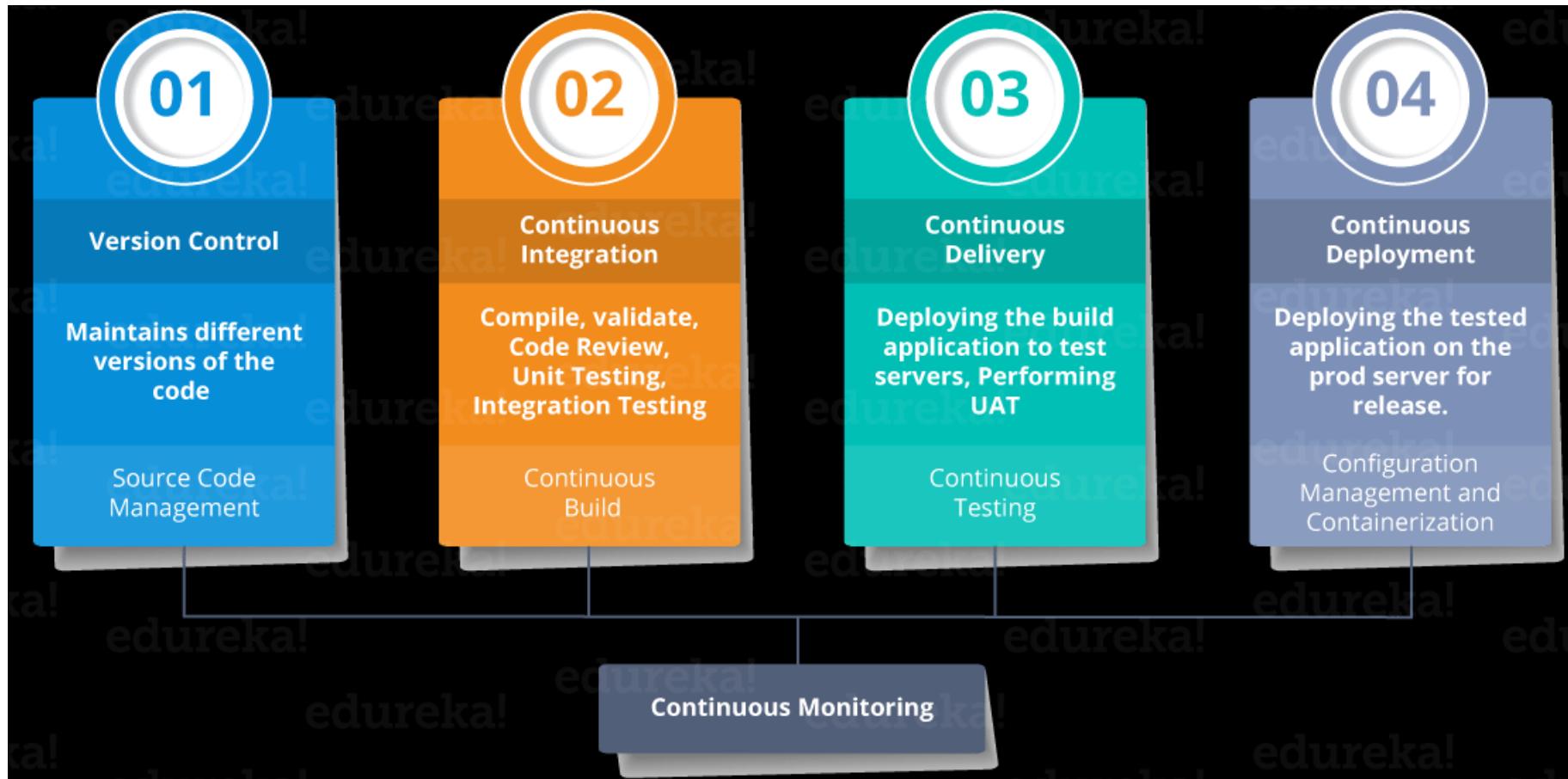
3. Tools (Enhanced)



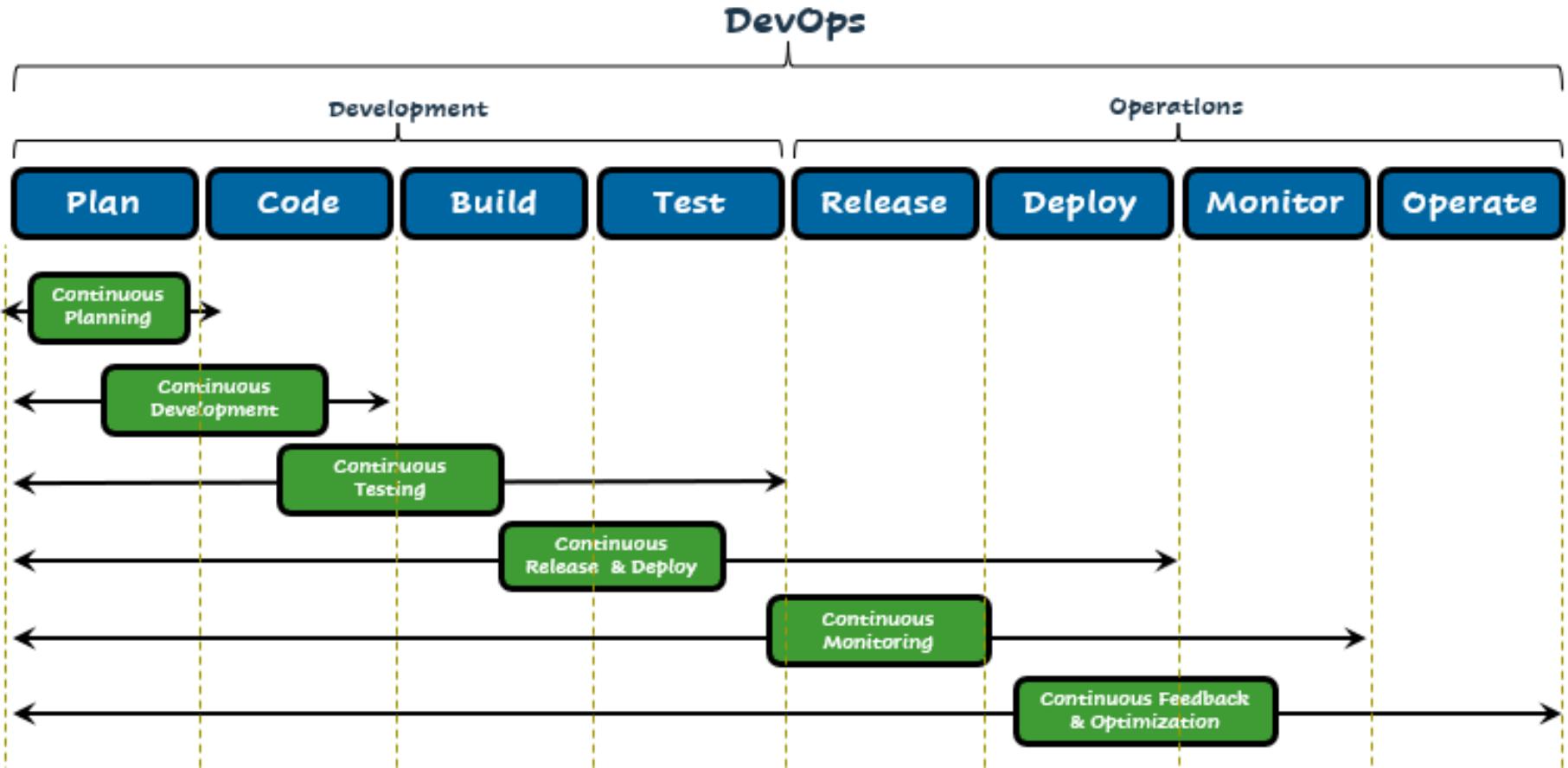


DevOps Practices

Key Practices in DevOps



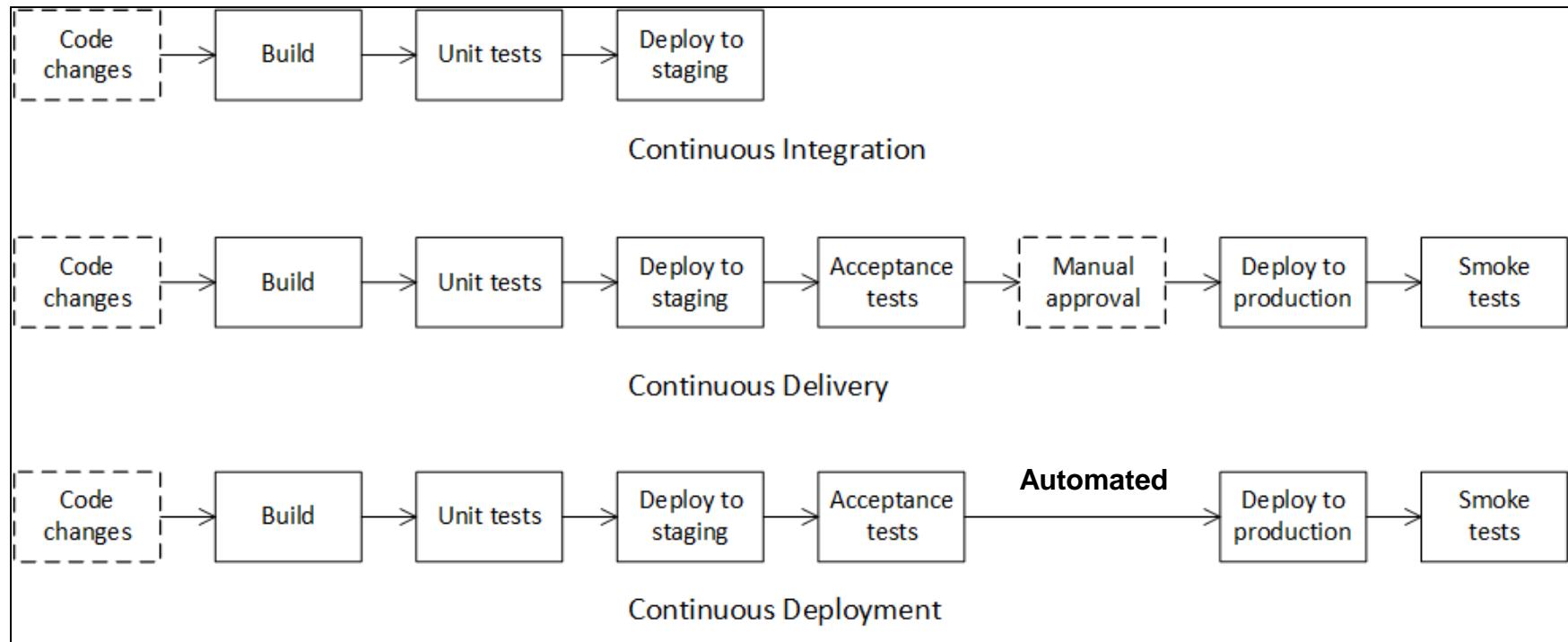
Key Practices tied to Lifecycle



CI/CD practices in DevOps

- Continuous Integration, Continuous Delivery, Continuous Deployment (CI/CD) are **DevOps practices** for producing software in short cycles between merging source code changes and updating applications.
- The ultimate goal of these practices is to:
 - Reduce the costs
 - Save time
 - Mitigate risks by delivering software in small pieces

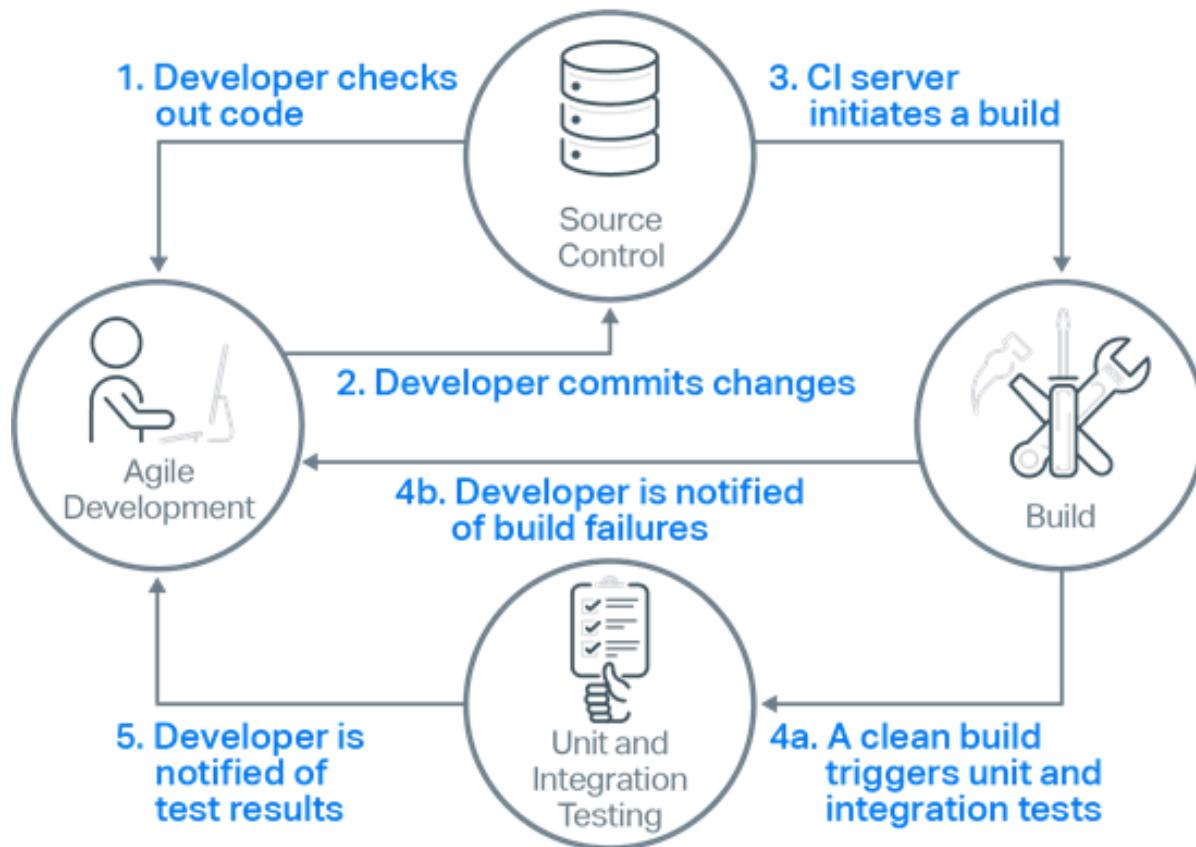
CI/CD practices in DevOps



Continuous Integration (CI)

- Continuous Integration is a software development practice in which developers regularly commit and push their local changes back to the shared repository (such as GIT, usually several times a day).
- Before each commit, developers can run unit tests locally on their source code as an additional check before integrating.
- A continuous integration service automatically builds and runs unit tests on the new source code changes to catch any errors immediately.

CI Workflow



Continuous Delivery (CD)

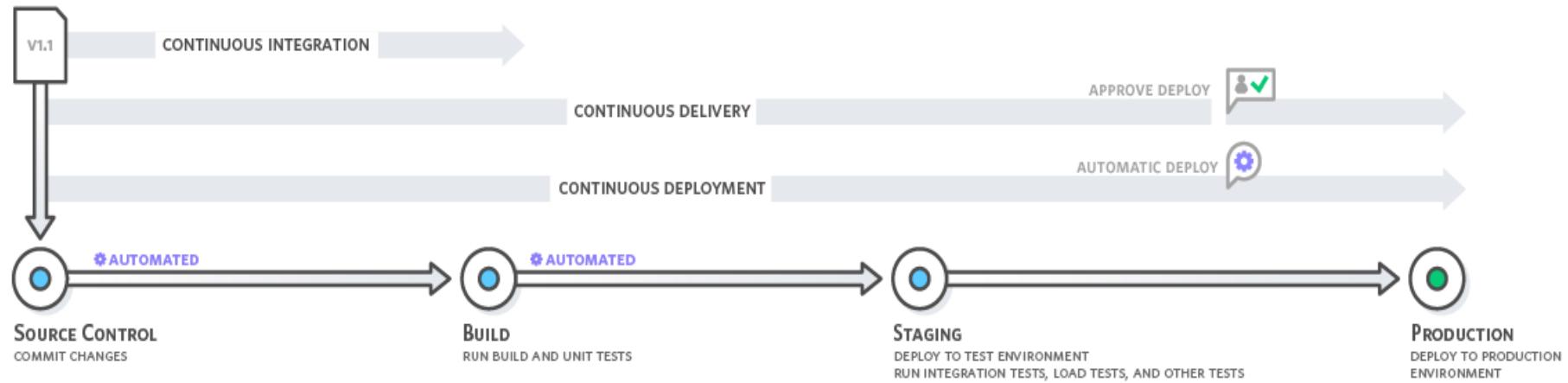
- Source code changes are automatically prepared for deployment to a production instance.
- After a build, the build artifact with new changes is deployed to a staging instance where advanced (integration, acceptance, load, end-to-end, etc.) tests are run.
- If needed, the build artifact is deployed to the production instance after **manual approval**.

Continuous Deployment (CD)

- Extends Continuous Delivery in which source code changes are automatically deployed to a production instance.
- The difference between Continuous Delivery and Continuous Deployment is the presence of manual approval.
- With Continuous Delivery, deployment to production occurs automatically after manual approval.
- With Continuous Deployment, deployment to production occurs automatically without manual approval.

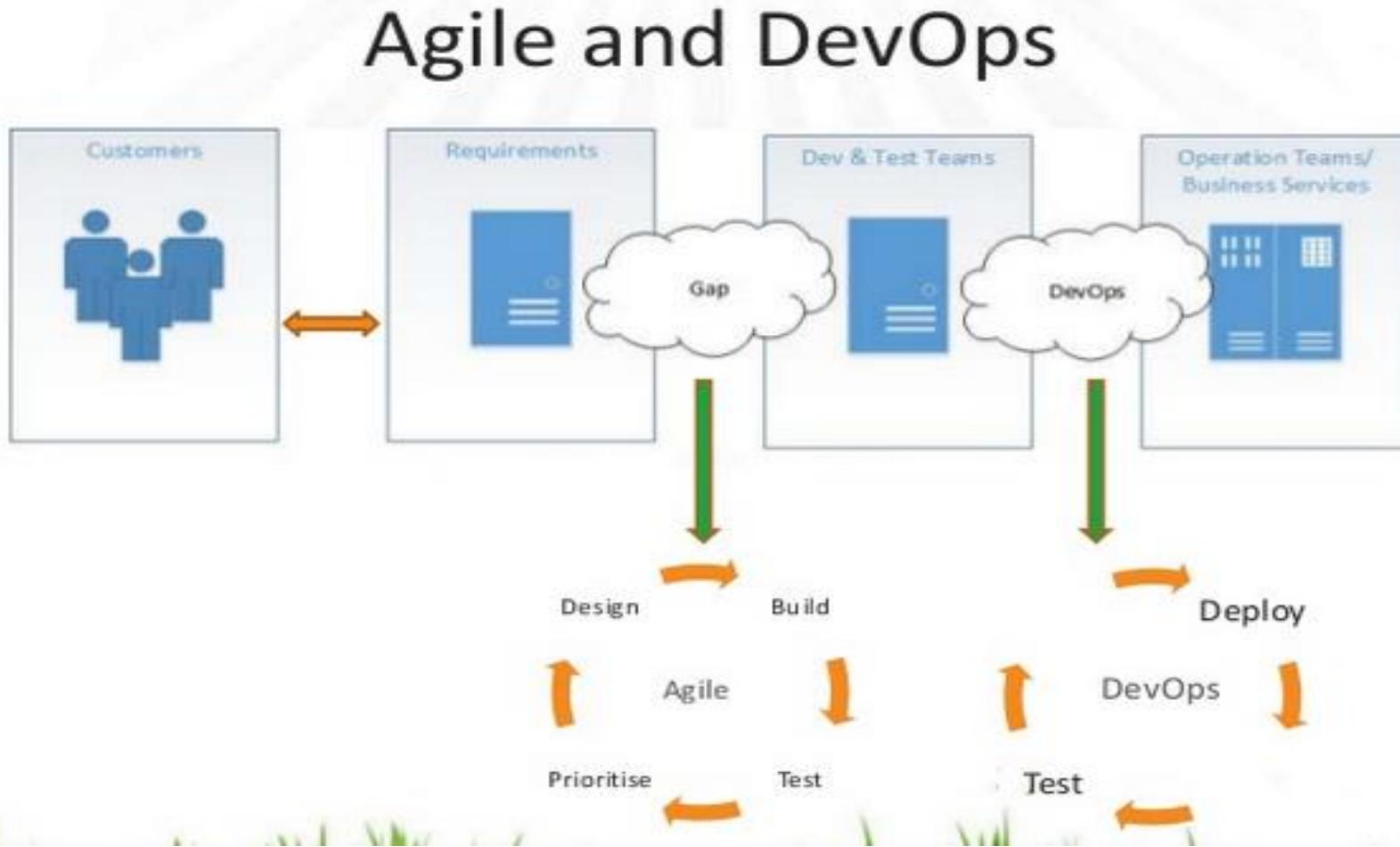
CI / CD Pipeline

A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of software



<https://aws.amazon.com/devops/continuous-integration/>

Agile and DevOps



Agile and DevOps



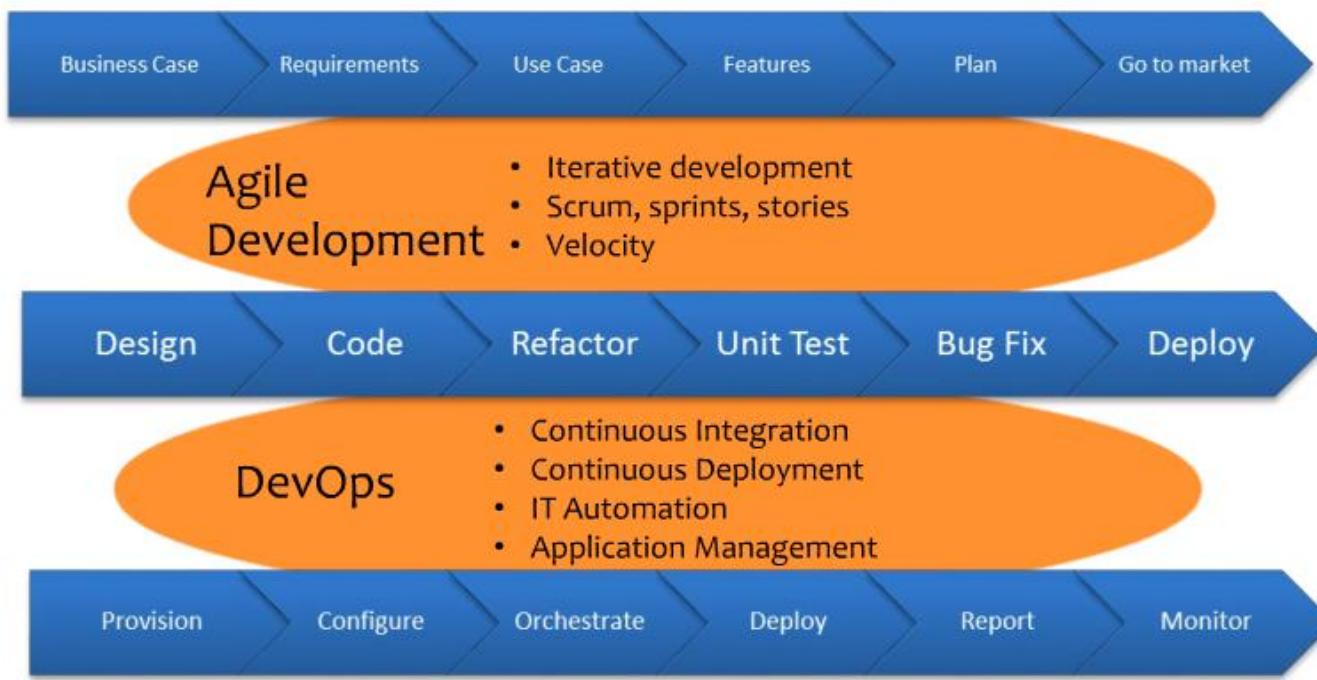
Business



Developers
(application)



IT Operations



Business
Agility

IT
Agility

Overview of XOps

"XOps" is a broad term that encompasses various practices, philosophies, and methodologies related to extending the principles of DevOps beyond traditional boundaries.

The "X" in "XOps" is a placeholder that can be replaced with various terms, indicating different areas or aspects where DevOps principles are applied.

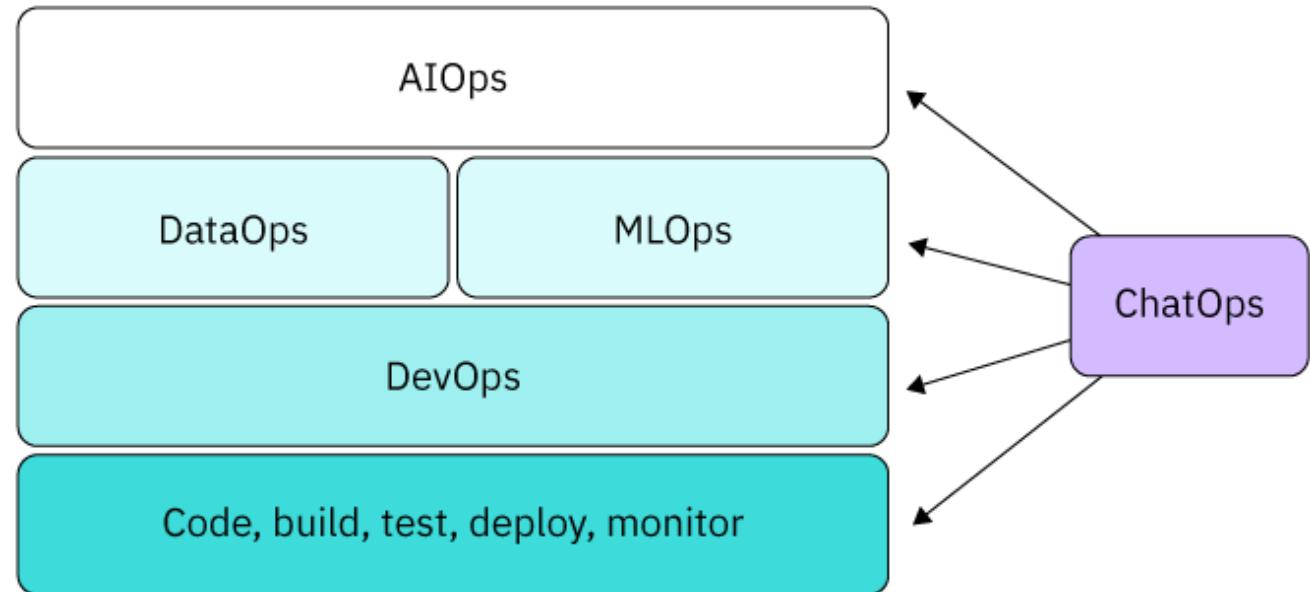
X + Ops [IBM]

Optimize and make better decisions.
Enhanced observability

Applying DevOps to data

Automation + CI/CD pipeline

Basic flow in any project



<https://developer.ibm.com/articles/all-the-ops-devops-dataops-mlops-and-aiops/>

DataOps vs MLOps

- DataOps is applying DevOps to data instead of an application or traditional code.
- DataOps is managing the whole data lifecycle, from creation to deployment to monitoring that data.
- MLOps applies DevOps principles to machine learning models, which are like applications but requiring training, running interference, and verifying that the results are accurate.
- MLOps is meant to standardize and streamline the lifecycle of machine learning models in production by orchestrating the movement of machine learning models, data, and outcomes among the systems.
- Both DataOps and MLOps are DevOps-driven.

<https://developer.ibm.com/articles/all-the-ops-devops-dataops-mlops-and-aiops/>

ChatOps

- ChatOps is a collaborative approach that integrates chat platforms, such as Slack, Microsoft Teams, or Mattermost, with DevOps practices, tools, and workflows.
- It enables teams to perform various operations, automate tasks, and manage infrastructure and deployments directly from within their chat environment.

Ex 1. Deployment Notifications

Whenever a new version of the web application is deployed to production, a deployment notification is automatically posted in the team's Slack channel. This notification includes details such as the version number, timestamp, and a link to the deployment dashboard.

Ex 2. Monitoring and Alerts

When a critical alert is triggered, such as high CPU usage or increased error rates, an alert message is sent to the Slack channel. Team members can immediately see the alert, investigate the issue, and take necessary actions.

Benefits of DevOps



21% Reduction
in time spent fixing &
maintaining
applications



33% Increase
in time for
infrastructure
improvements



19% Improvement
in application quality
and performance



15% Increase
in time for
self-improvement



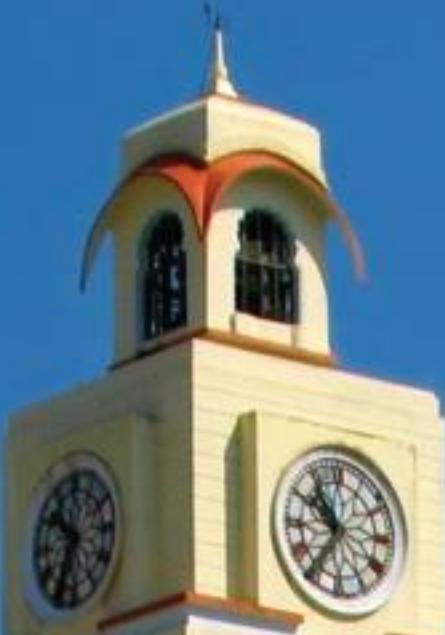
18% Increase
in revenue



60% Decrease
in application release
time



Thank You!



BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lecture No. 3

Agenda

-
- Modern application requirements
 - Cloud-native evolution
 - Introducing Cloud-native software
 - Cloud-enabled vs Cloud-based vs Cloud-native apps
 - Obstacles & enablers for Cloud-native apps
 - CNCF Landscape
 - Overview of Cloud-native ecosystem
 - Cloud DevOps
 - Microservices
 - Containers
 - Serverless Computing
-



Modern Application Requirements

R4: “Cloud Native Patterns”, by Cornelia Davis. Publisher: Manning, 2019

Modern Application Requirements

Streaming / OTT



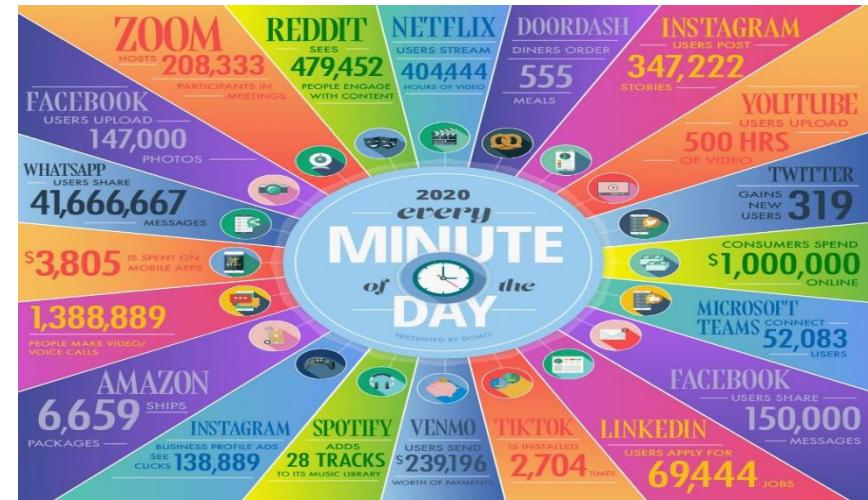
Social Media Apps



E-Commerce Apps



Data Deluge (Infographic)



Modern Application Requirements

General Requirements

1. Zero Downtime
2. Shortened Feedback Cycles
3. Mobile and Multi-device support
4. Connected Devices (IoT)
5. Data driven

Software Quality Attributes

- Scalability
- Performance
- Availability
- Reliability
- Interoperability
- Testability
- Usability
- Modifiability
- Security
- Portability
- Maintainability

Principles of Software Delivery

1. Create a Repeatable, Reliable Process for Releasing Software
 2. Automate Almost Everything
 3. Keep Everything in Version Control
 4. Build Quality In Every Aspect of The Process
 - “The Earlier you catch the defects, the cheaper they are to fix”
 5. Everybody Is Responsible for the Delivery Process
 6. Continuous Improvement
-



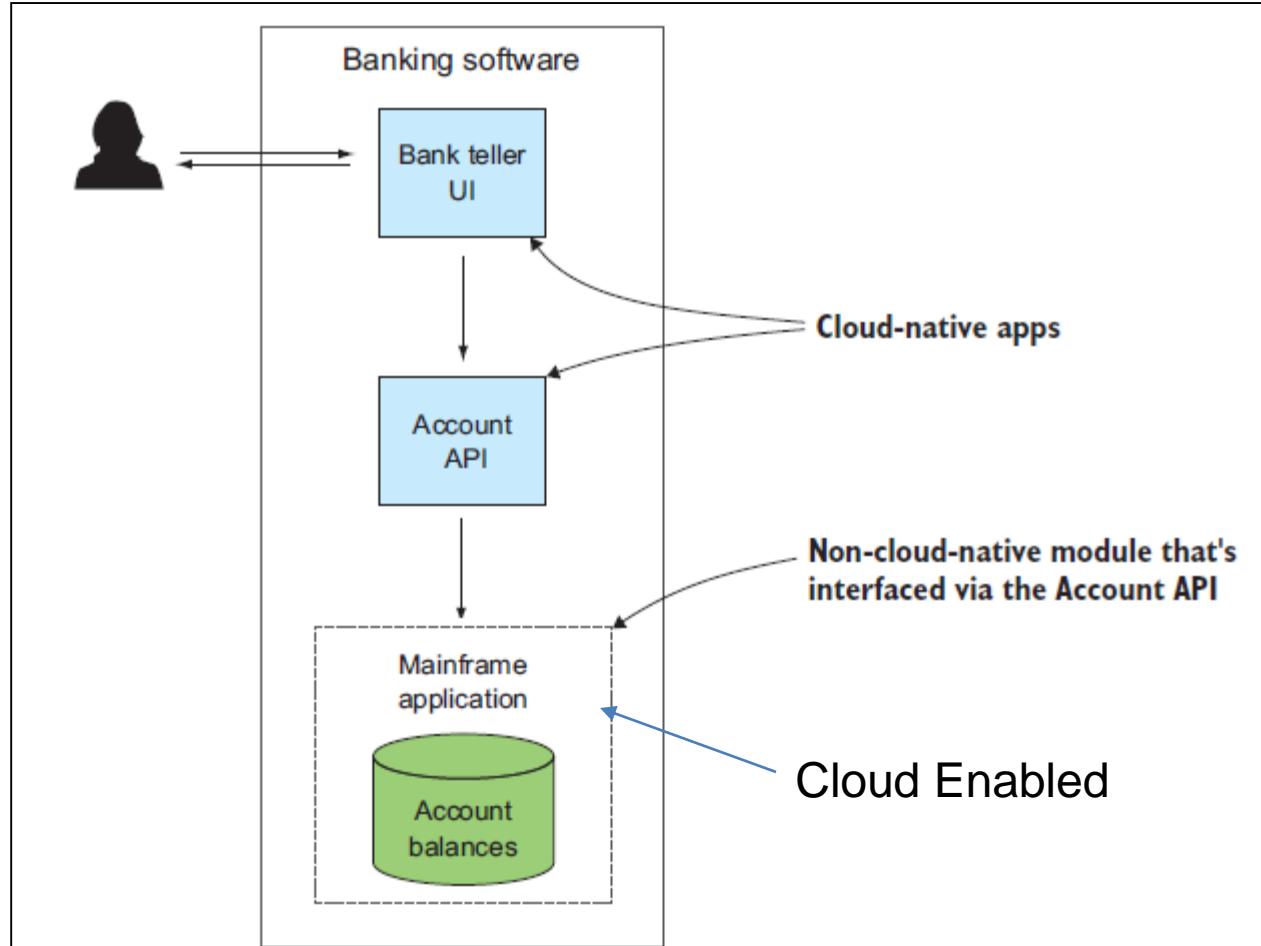
Cloud-enabled vs Cloud-based vs Cloud-native apps

Evolution of Cloud Native apps

Cloud Enabled

- Legacy applications
- Applications were built traditionally in a monolithic fashion
- They depend on local resources and hardware (on-premises)
- Integrate with services hosted on the cloud
- **The application cannot take the advantage of factors like scalability as the underlying architecture remains monolithic**

Cloud Enabled



Cloud Based

- Applications moved to cloud to leverage capabilities of cloud
 - Scalability
 - Higher Availability
- No need of redesign of applications to migrate to cloud
- Ex: In-house web application moved to AWS or Azure
- Do not have to worry about
 - Management of resources
 - Maintaining the servers
 - Backup
- Advantages
 - Pay for what is used
 - Scaling up/down
 - Zero downtime

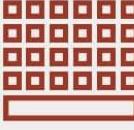
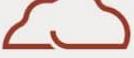
Cloud Native

- Redhat - “Cloud-native applications are a collection of small, independent, and loosely coupled services.”
 - VMWare - “Cloud native is an approach to building and running applications that exploits the advantages of the cloud computing delivery model.”
 - IBM - “Cloud native refers less to where an application resides and more to how it is built and deployed.”
 - Cloud-native is born in the cloud
 - Cloud is about *where* we are computing; Cloud-native is about *how*
-

Cloud Native

- Cloud Native applications are:
 - Architected to run in a Cloud environment (Public clouds - AWS, GCP, Azure)
 - Built using cloud based technologies
 - Accessible and Scalable
- Allow developers to continuously deliver new services more quickly and easily
- Cloud-native technologies
 - Continuous integration
 - Orchestrators
 - Container engines

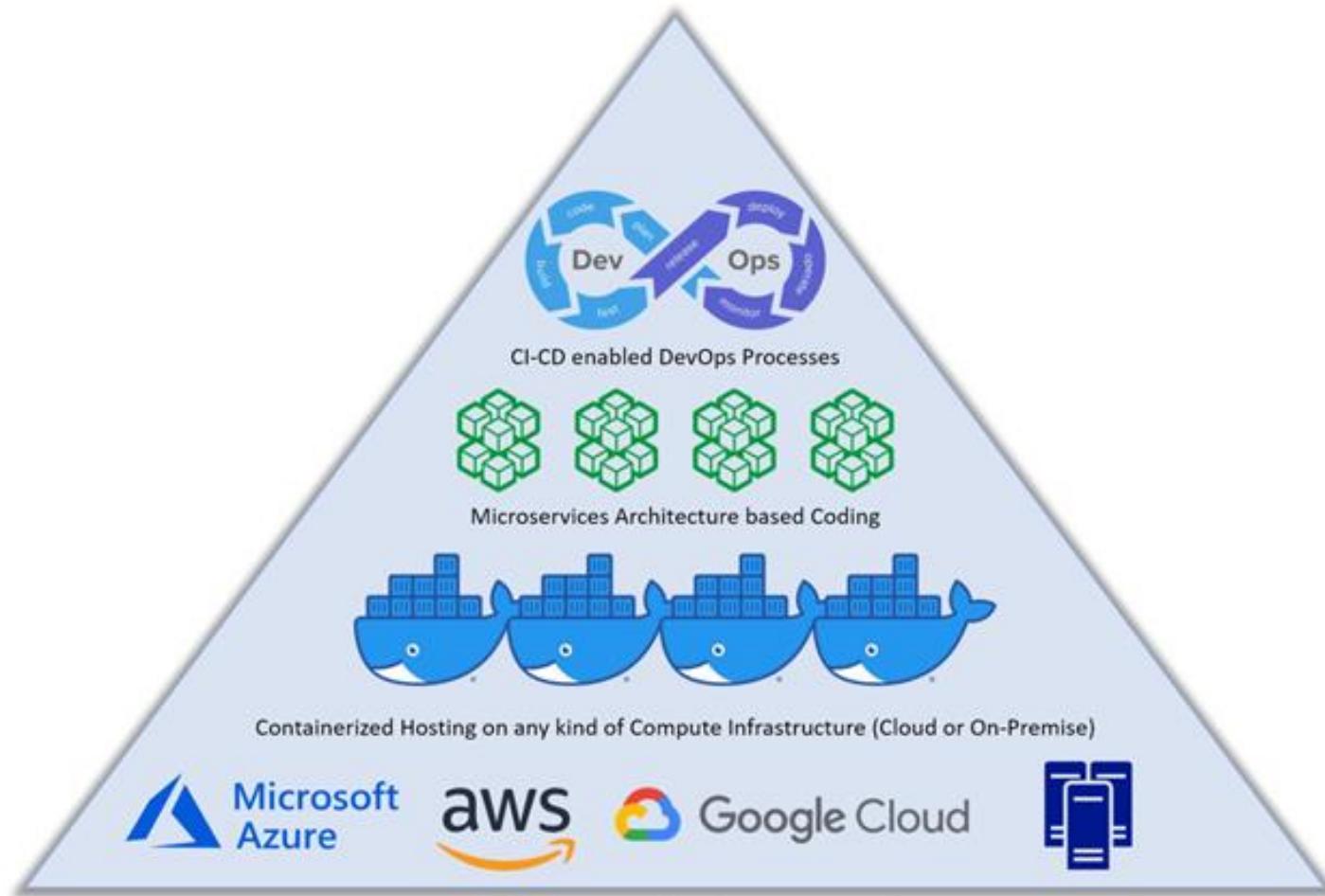
Cloud Native - Evolution

	Development Process	Application Architecture	Deployment & Packaging	Application Infrastructure
~ 1980	Waterfall 	Monolithic 	Physical Server 	Datacenter 
~ 1990				
~ 2000	Agile 	N-Tie 	Virtual Servers 	Hosted 
~ 2010	DevOps 	Microservices 	Containers 	Cloud 

Cloud Native App -> DevOps + Microservices + Containers + Cloud

<https://www.oracle.com/in/cloud/cloud-native/what-is-cloud-native/>

Cloud Native Technologies Pyramid



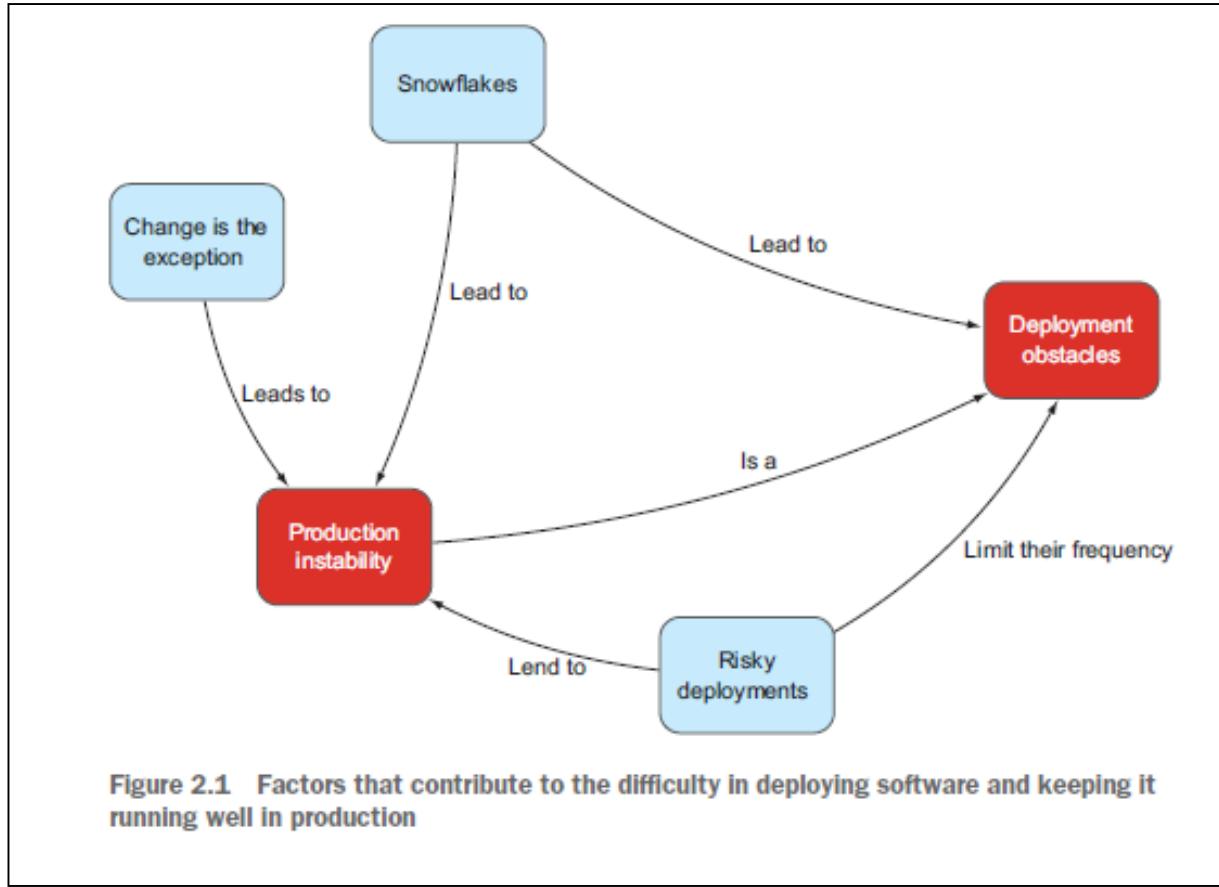
Cloud Native Technologies Pyramid

<https://www.ridge.co/blog/cloud-native-applications-explained/>

Cloud Native – Focus Areas

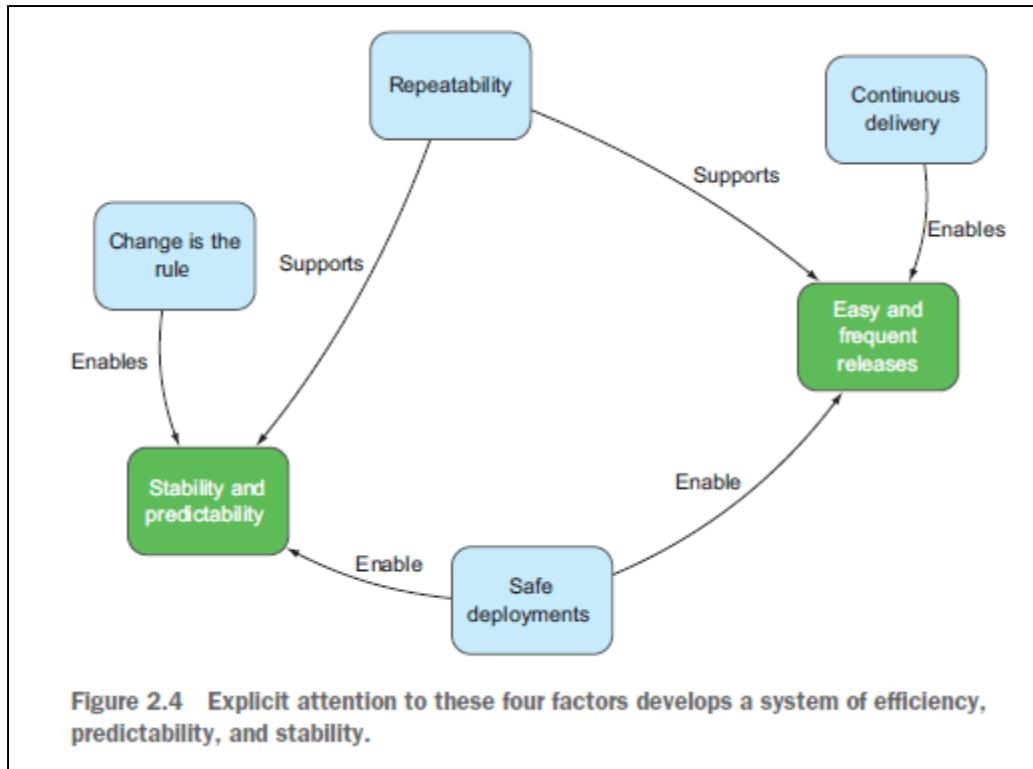
- **Design**
 - **Cloud-native:** Must think about failure. Use of microservices architecture.
 - **Cloud-based:** Was designed for availability.
- **Implementation**
 - **Cloud-native:** Faster to deploy because there is no hardware or software to deploy.(Container Images)
 - **Cloud-based:** Slower because of hardware provisioning or software setup.
- **Pricing**
 - **Cloud-native:** Consumption-based-pricing (pay for what you use)
 - **Cloud-based:** More expensive because you have to own the whole stack (compute – EC2, storage, monitoring & logging services) etc.

Obstacles [Ops team perspective]



- Snowflake (it works on my machine), difference in environments and artifacts across deployments
- Risky deployment (software instability, avoid downtime), deployment strategies? One service fails, ripple effects across system
- Change is the exception (infrastructure instability caused by an exception case)

Enablers for Cloud-native Apps



- Google running 2 Million Servers in Data Centers
- Repeatability replaces Snowflakes (same deployable across all environments)
- Continuous Delivery replaces Risky deployment (Canary, Blue-Green, Ramped etc)
- Change is the rule, instead of exception

Examples of Cloud Native Applications



Netflix is a famous streaming app for TV shows, movies, and documentaries.

In 2016, the organization decided to go cloud-native and shifted to microservices.



Uber is another popular app that uses a cloud-native approach. Uber has over 4,000 independent microservices, which the team monitors using the Prometheus platform. This allows Uber developers to quickly respond to market changes and update or scale a specific portion of the app.



Airbnb is a well-known American vacation rental company that operates in 65,000 cities around the world. As the company grew, it shifted to a microservices model, and as of 2017, Airbnb deploys 3,500 microservices per week, which allows the organization to keep providing services seamlessly to thousands of customers.

CNCF - Cloud Native Computing Foundation

- CNCF is the open source, vendor-neutral hub for cloud native computing
- Part of Linux foundation
- URL - <https://www.cncf.io/>
- 24 Graduated projects (stable in production environment)
 - ArgoCD, Kubernetes, envoy service proxy, Helm, Prometheus etc.
- 37 Incubating projects
 - gRPC, Thanos (monitoring), OpenTelemetry (tracing) etc.

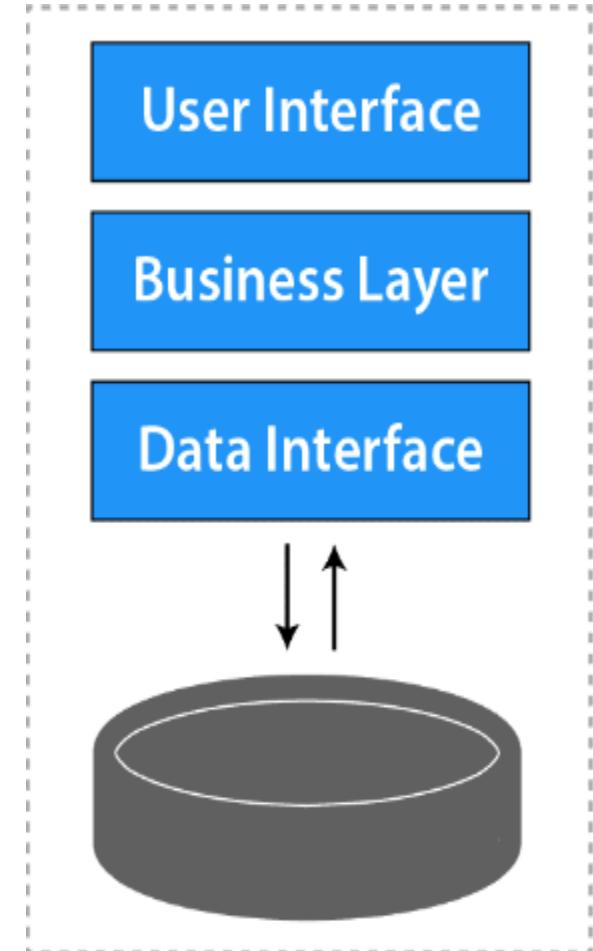


Microservices

(Overview of Cloud-native ecosystem)

What is Monolithic Architecture?

- Monolith means composed all in one piece.
- They're typically complex applications that encompass several tightly coupled functions.
- When all functionality in a system had to be deployed together, we consider it a monolith.



Monolith Application (Food Delivery)

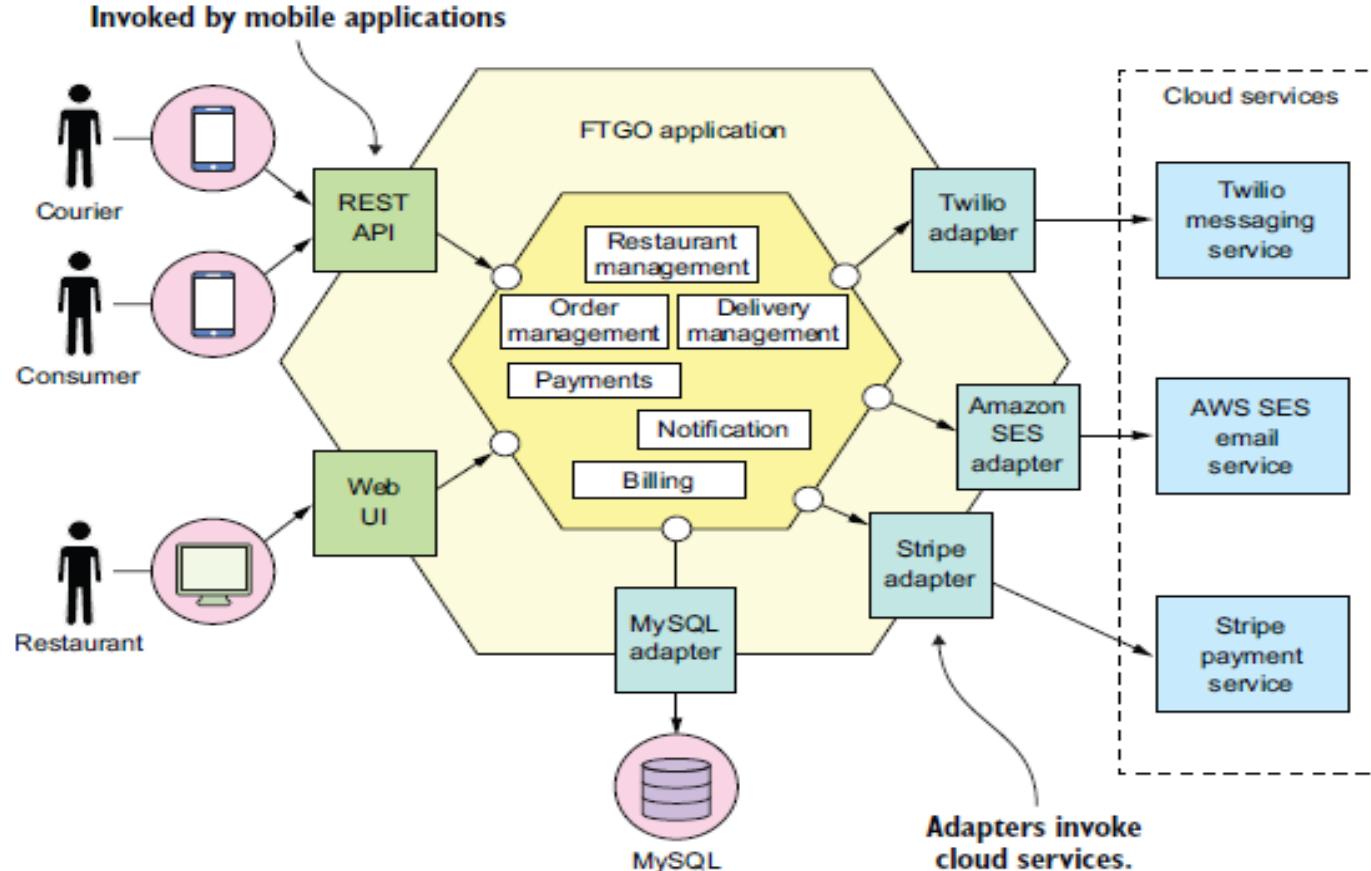
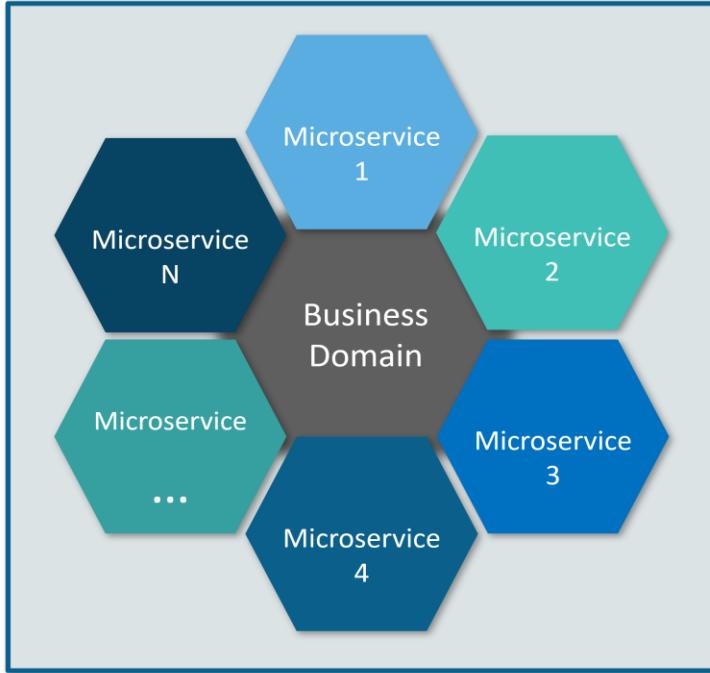


Figure 1.1 The FTGO application has a hexagonal architecture. It consists of business logic surrounded by adapters that implement UIs and interface with external systems, such as mobile applications and cloud services for payments, messaging, and email.

Monolithic Limitations

- *Technology Barrier* - embracing new technologies means application needs to be re-written
- *Scalability* - The only option is scaling the whole application, since we can't scale the components independently
- *Size* - As application size increases, complexity increases, may become unmanageable
- *Difficult to understand* - Every new joinee needs to understand the entire application, not just his/her modules, because of dependencies
[Maintenance project problems]

What are Microservices

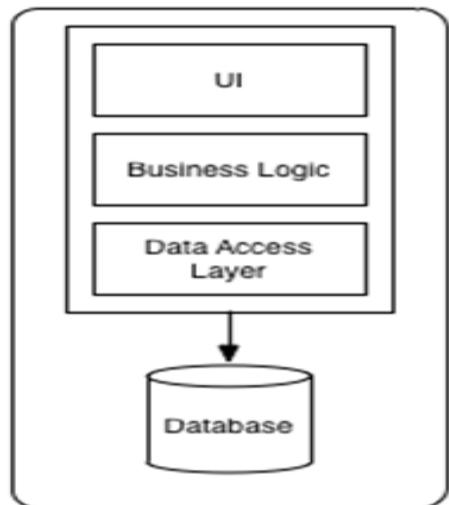


- An architectural style
- Microservices are autonomous, loosely coupled, and independently deployable services modeled around a business domain
- Each microservice can focus on a single business capability

Robert C. Martin's Single Responsibility Principle

*Gather together the things that change for the same reasons.
Separate those things that change for different reasons.*

Why Microservices are required



Monolithic Architecture

Company	Deployments
Amazon	23,000 / day
Google	5,500 / day
Netflix	500 / day
Twitter	3 / week

Use Case - Shopping Cart Application (Amazon, Flipkart etc.)

Problems with implementing Shopping Cart as Monolithic Architecture

Scenario 1: Agility

Immediate changes to the application needs entire code to be rebuilt for every small change (version management).

Scenario 2: Hybrid Technologies

Developers are from varying skillsets such as C#, Java, Python etc. They are forced to build the shopping cart application using a single technology.

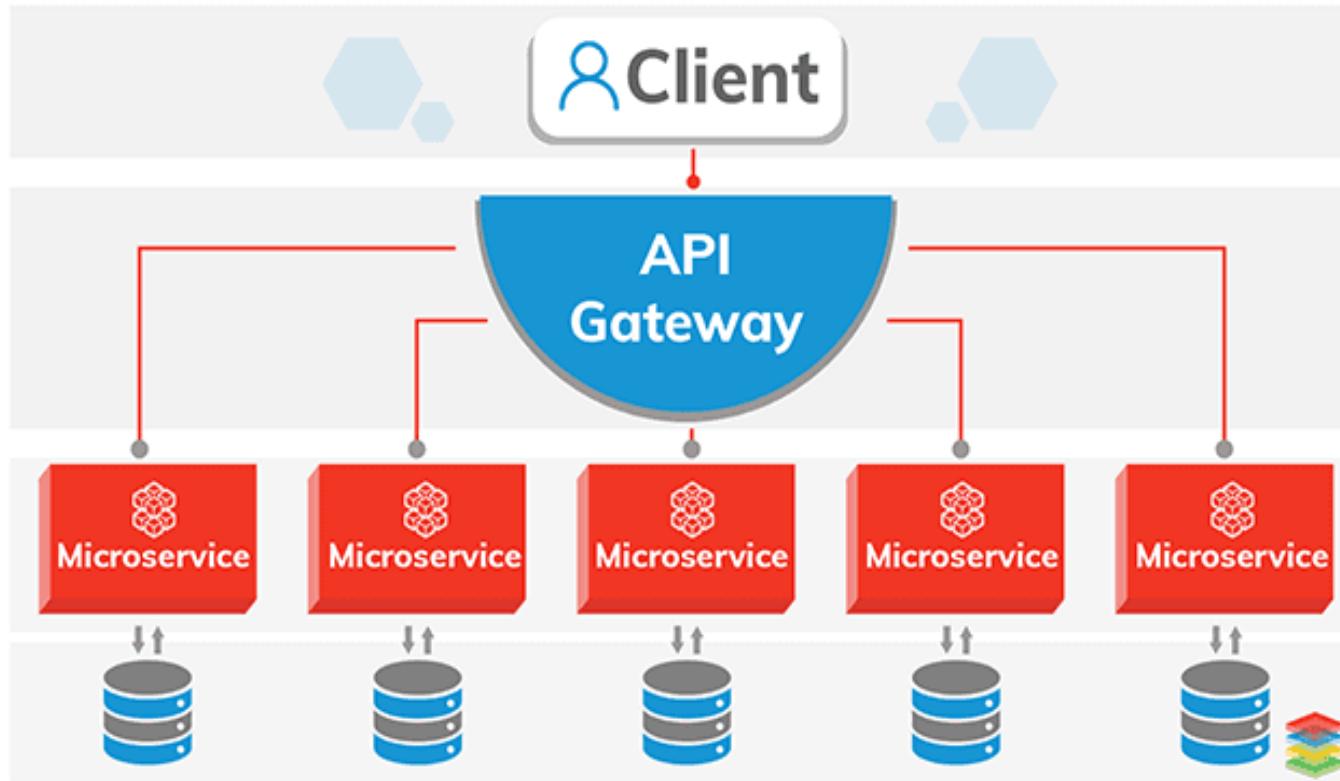
Scenario 3: Fault Tolerance

A feature (Ex: cart amount debit) is not working in the application. A fix implies that the entire application has to be re-built, re-tested and re-deployed.

Scenario 4: Scalability

Developers cannot scale the application simultaneously. New instances of the same application have to be created every time a new feature has to be developed or deployed.

Example Architecture



Cloud – Microservices Market

Cloud Microservices Market

Market Size in USD Billion

CAGR 22.88%



A bar chart titled 'Cloud Microservices Market' showing market size in USD Billion. The Y-axis represents market size, and the X-axis shows years 2023 and 2028. The 2023 bar is labeled 'USD 1.33 B' and the 2028 bar is labeled 'USD 3.72 B'. The chart has a dark background with light blue bars.

Year	Market Size (USD Billion)
2023	USD 1.33 B
2028	USD 3.72 B

Source : Mordor Intelligence



Study Period 2019-2027

Market Size (2023) USD 1.33 Billion

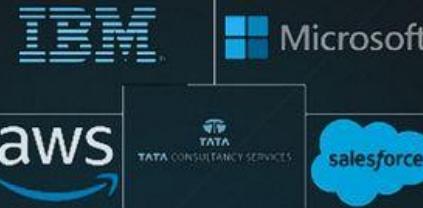
Market Size (2028) USD 3.72 Billion

CAGR (2023 - 2028) 22.88 %

Fastest Growing Market Asia Pacific

Largest Market North America

Major Players

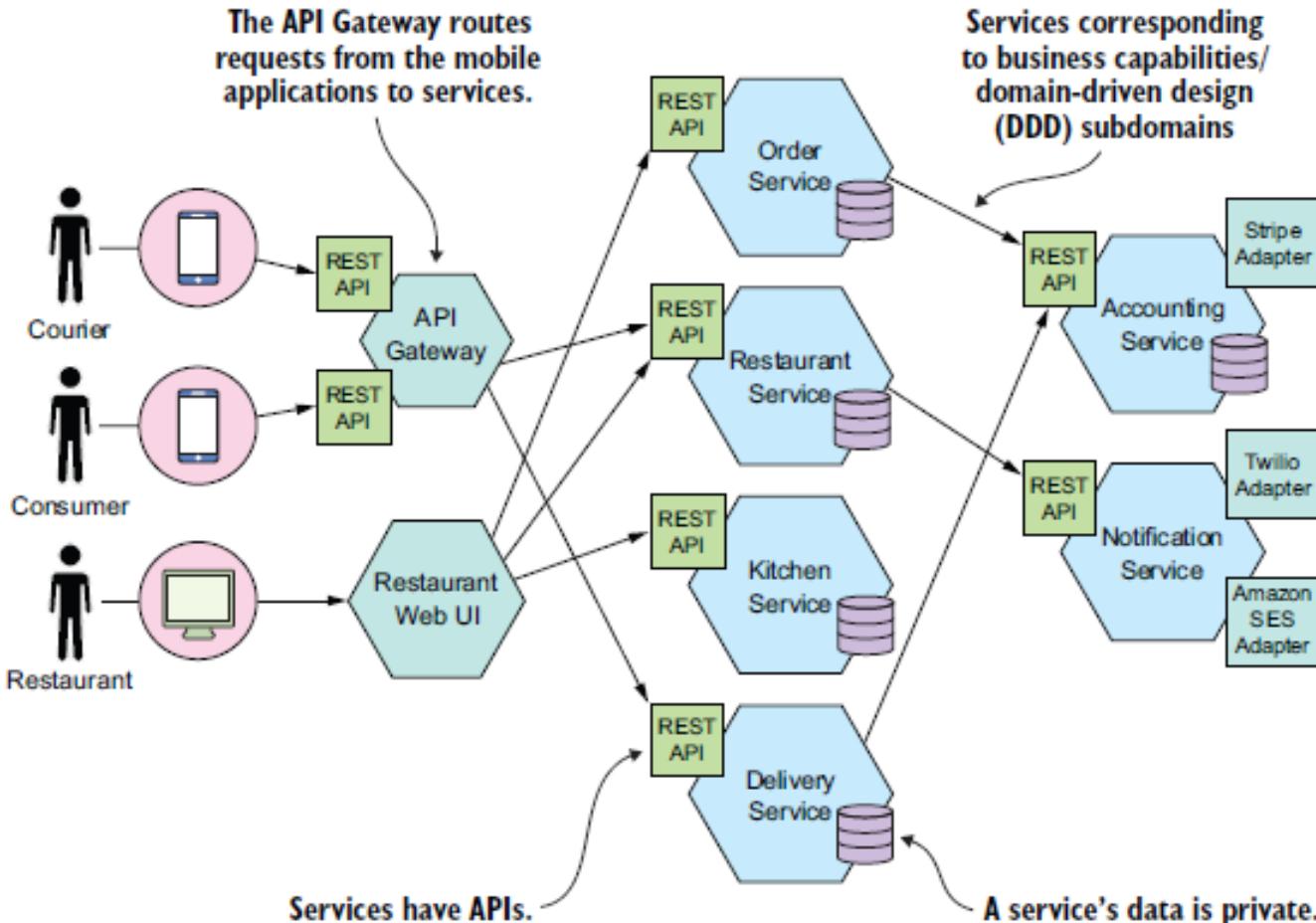


Logos of major players in the Cloud Microservices Market: IBM, Microsoft, AWS, TATA Consultancy Services, and salesforce. The logos are arranged in a cluster, with dashed lines connecting them to the 'Major Players' heading.

*Disclaimer: Major Players sorted in no particular order

BITS Pilani, Pilani Campus

Microservices Architecture (Food Delivery)



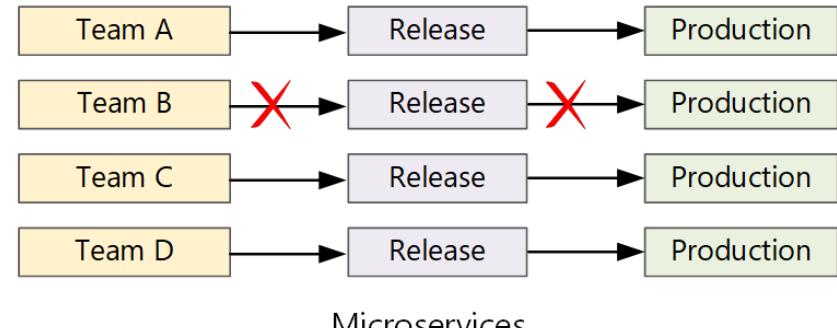
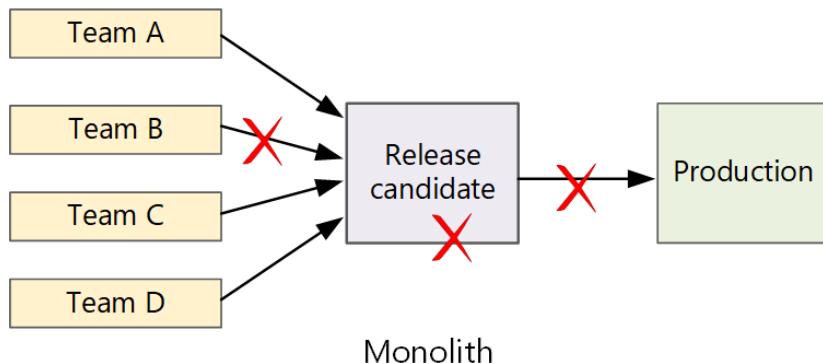
CI/CD pipeline for Monolith vs Microservices

Monolith

- Single Build Pipeline
- Bug fixing delays release of features

Microservice

- One service, one pipeline [pipeline-per-service pattern]
- High release velocity and reliability



<https://learn.microsoft.com/en-us/azure/architecture/microservices/ci-cd>

FTGO - Monolithic Hell - Pipeline

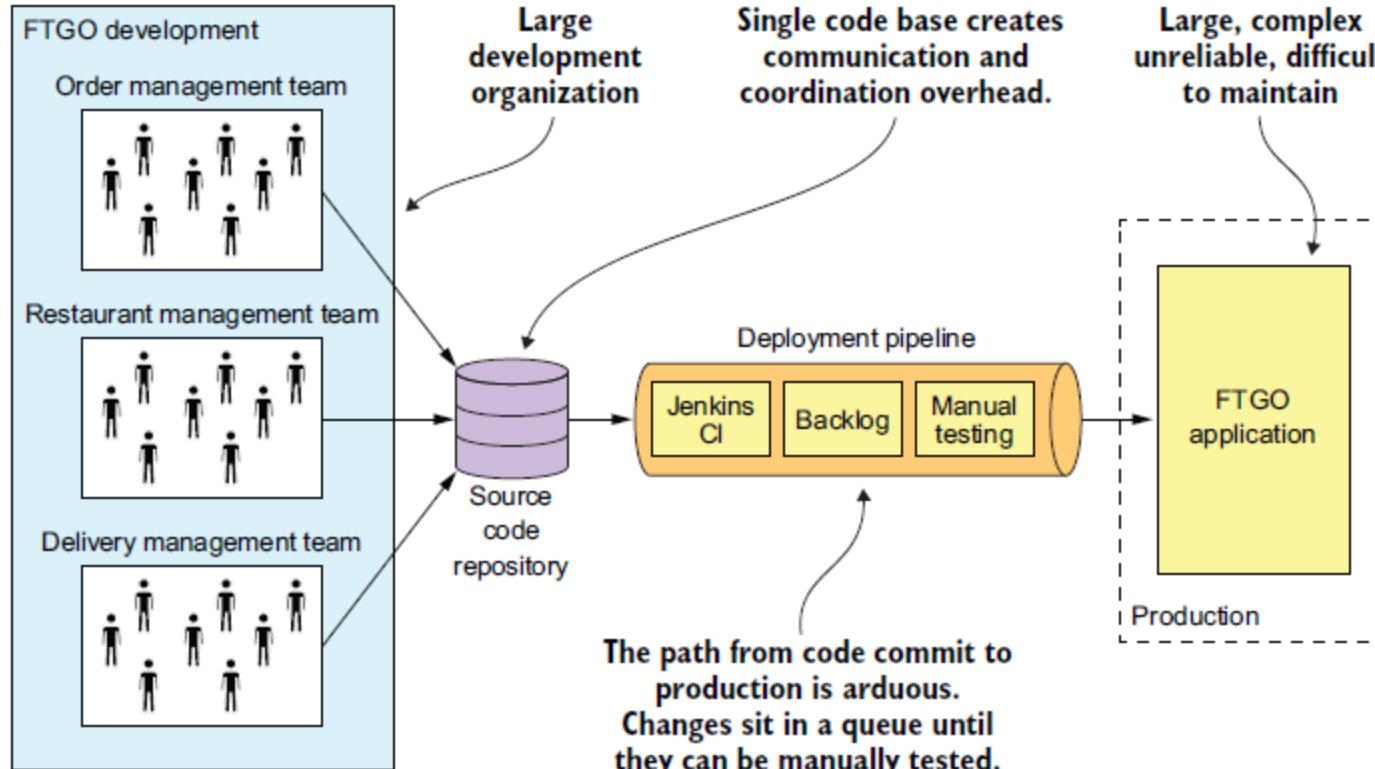
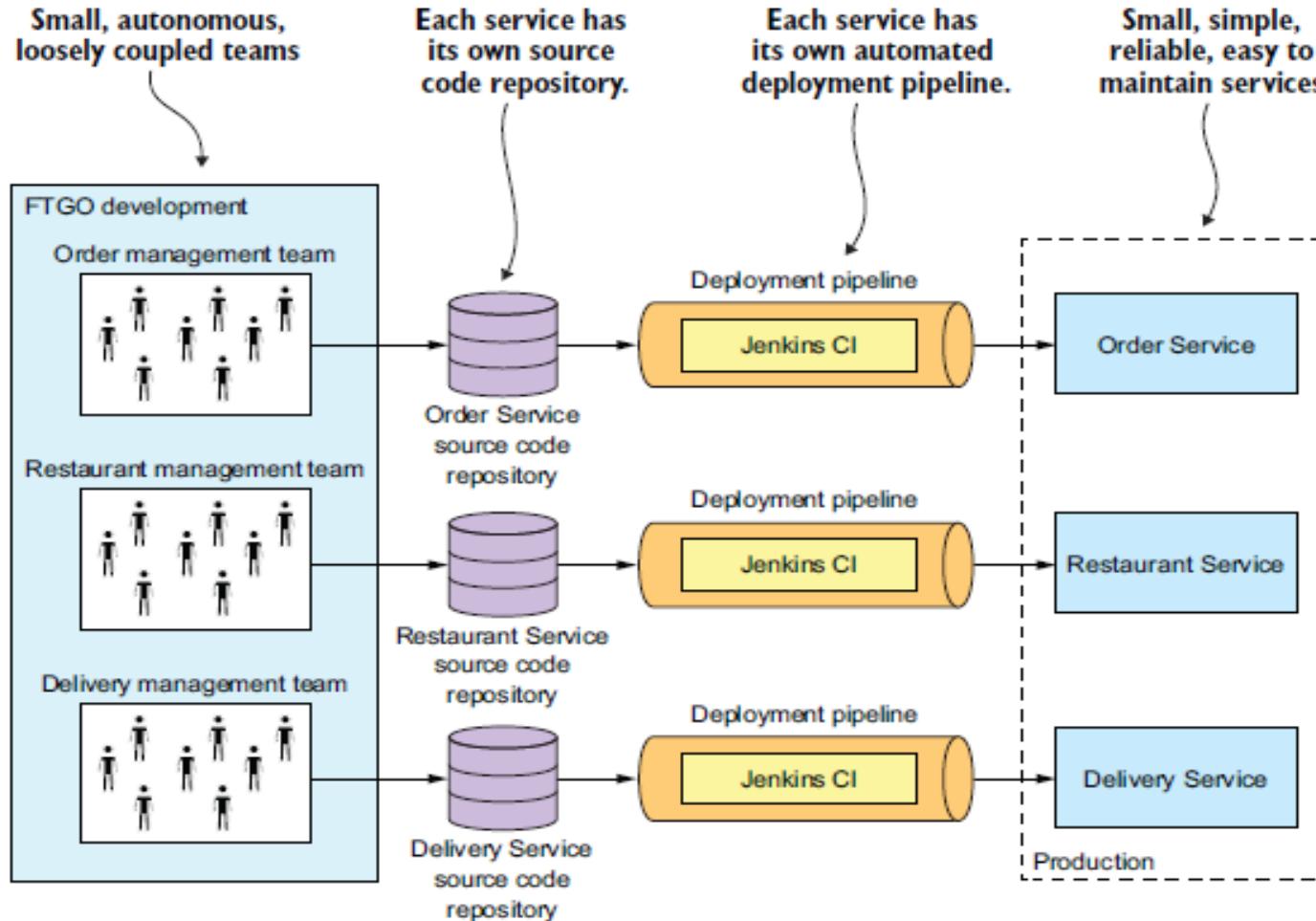


Figure 1.2 A case of monolithic hell. The large FTGO developer team commits their changes to a single source code repository. The path from code commit to production is long and arduous and involves manual testing. The FTGO application is large, complex, unreliable, and difficult to maintain.

Microservices Pipeline – FTGO Application

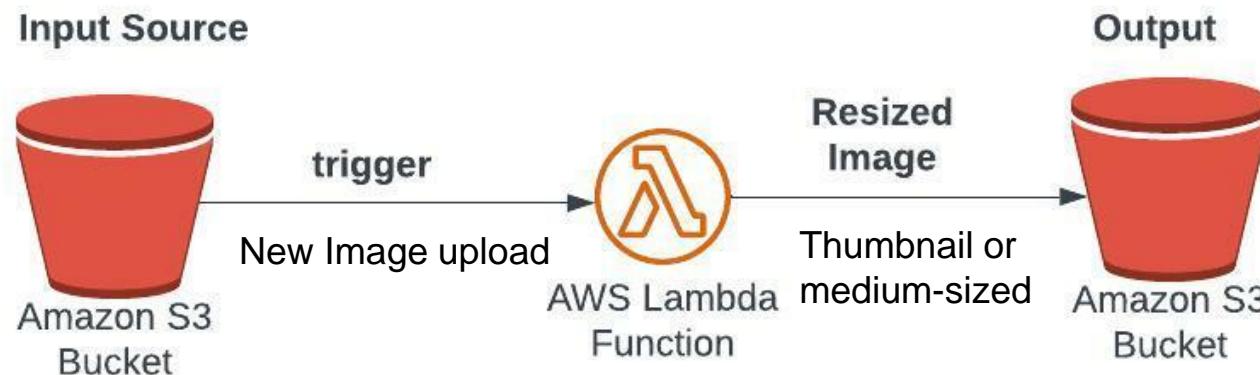




Serverless Computing

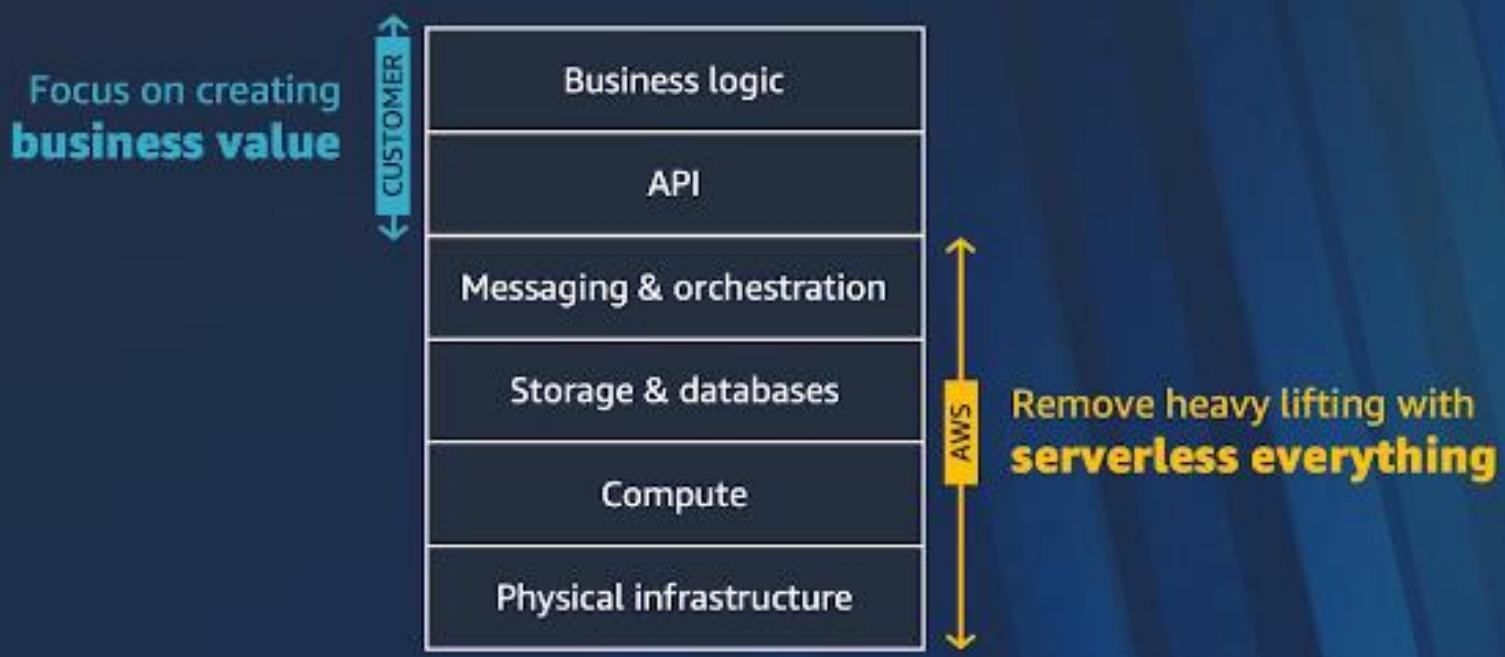
Serverless Computing

- Application deployment paradigm that allows applications to run on-demand, consuming only the resources required to execute them
- Build and run applications and services without managing infrastructure
- Developers can focus on their core product
- Ex: AWS Lambda, Google Cloud Functions (GCF), Azure Functions are examples of FaaS



Serverless Computing

Serverless help developers focus on building, not managing



AWS Serverless Services

Serverless is much more than compute

COMPUTE



DATA STORES

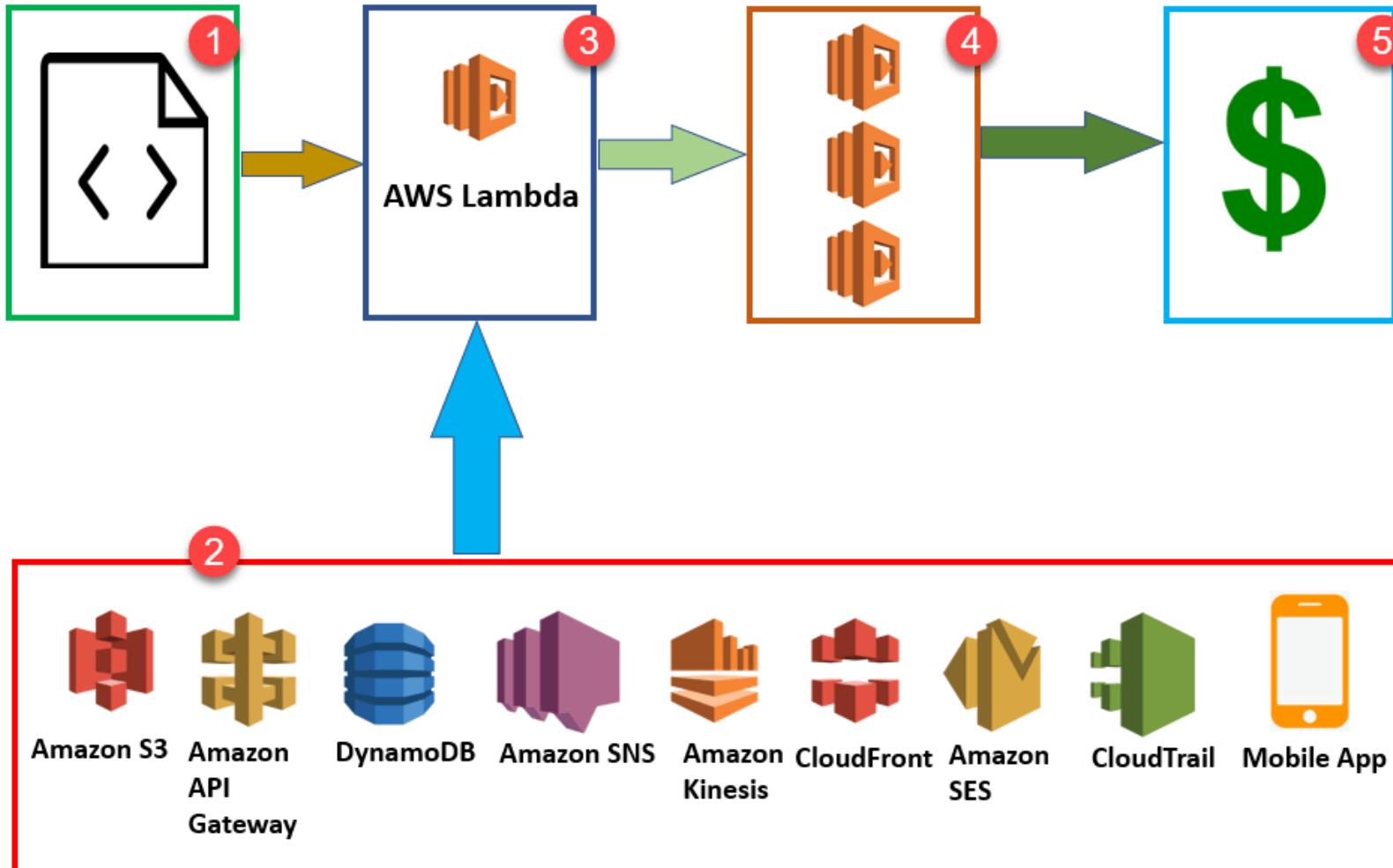


INTEGRATION



AWS Lambda

(FaaS - Function as a Service)

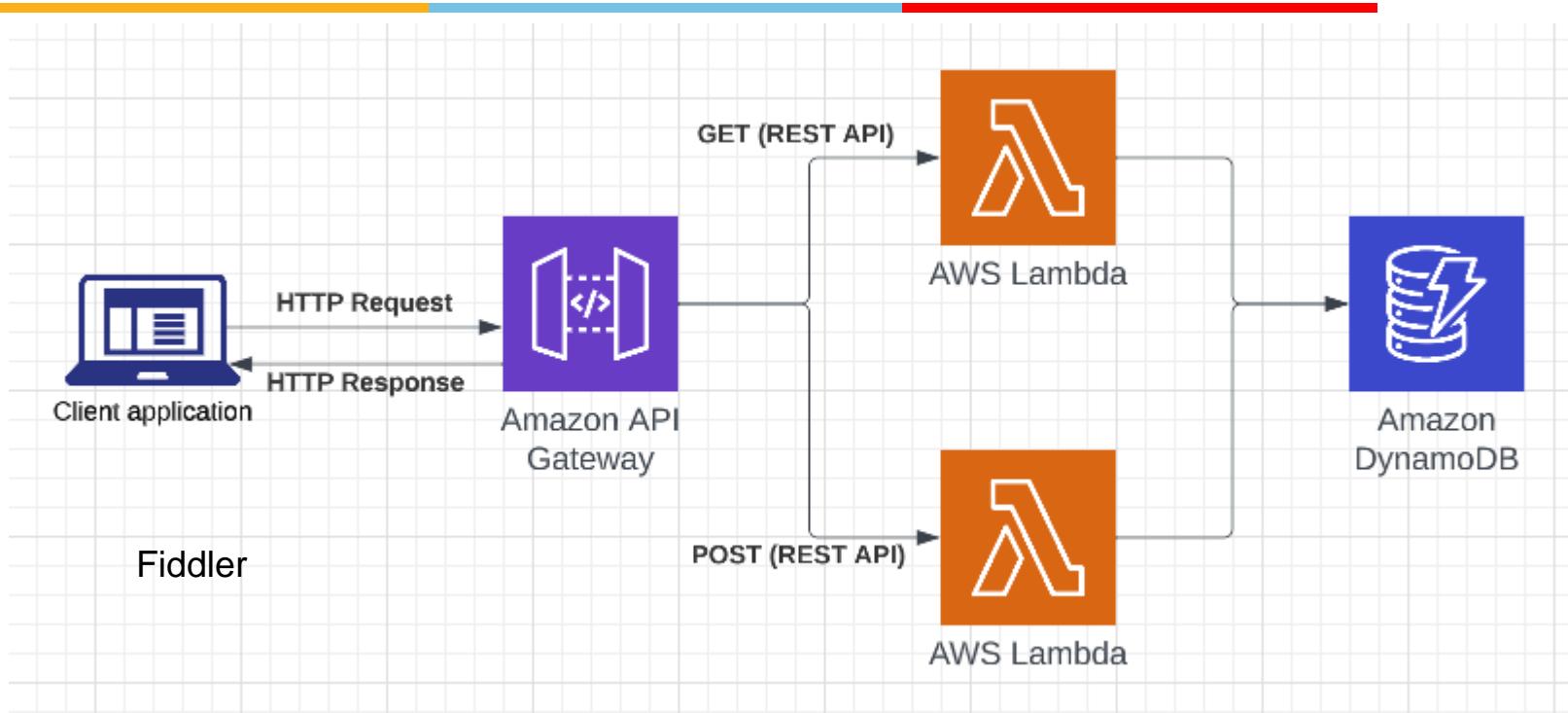


Serverless Stack

- When building Serverless applications, AWS Lambda is one of the main candidates for running the application code. Typically, to complete a Serverless stack you'll need:
 - A Compute service;
 - A Database service; and
 - A HTTP Gateway service.
- Lambda fills the primary role of the compute service on AWS.
- It also integrates with many other AWS services and, together with API Gateway, DynamoDB and RDS, forms the basis for Serverless solutions for those using AWS.
- Lambda supports many of the most popular languages and runtimes, so it's a good fit for a wide range of Serverless developers.



Demo of Serverless Application



- **AWS Lambda (Get and Post functions) are Serverless Compute**
- **Amazon DynamoDB is the Serverless Database**
- **Amazon API Gateway is the HTTP Gateway Service**

Drawbacks of Serverless Deployment

- It's not the best scenario for executing long-running applications [No control over the environment]
- Vendor lock-in
- Cold start [AWS Lambda typically keeps containers alive for 30-45 minutes. For a new request, if a function is not running in a warmed container, a new container will be created called 'cold start'. Latency (more time) to execute the request]



Thank You!



BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lecture No. 4

Agenda

Source Code Management

- Version Control System and its Types
- Introduction to GIT
- GIT Basics commands
- GIT workflows- Feature workflow, Master workflow, Feature branching
- GIT Pull requests
- Managing Conflicts
- Demo of GIT and GitHub



Version Control

Evolution of Version Control

What is Version Control

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	Example Tools (VCS)
First Generation	None	One file at a time	RCS, SCCS
Second Generation	Centralized	Multi-file	CVS, Subversion
Third Generation	Distributed	Changesets	GIT

Evolution of Version Control

Version Control System

- A category of software tools that help a software team manage changes to source code over time
- Version control software keeps track of every modification to the code
- If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake
- For almost all software projects, the source code is like the crown jewels - a precious asset whose value must be protected
- Great version control systems facilitate a smooth and continuous flow of changes to the code

Evolution of Version Control

Benefits of Version Control System

- A complete long-term change history of every file
- Restoring previous versions
- Branching and merging; having team members work concurrently
- Traceability; being able to trace each change made to the software
- Backup



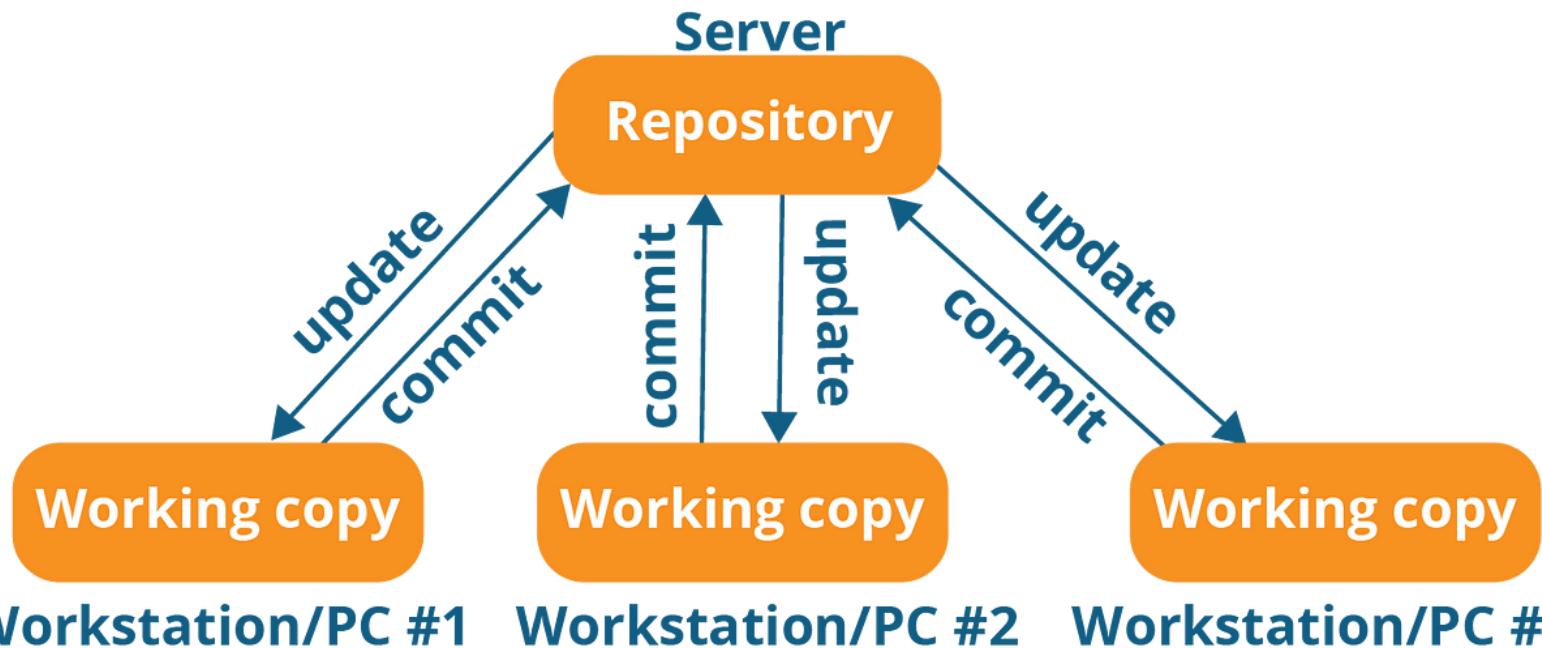
Version Control Systems and Types

Version Control System Types

- **Centralized Version Control System:[CVCS]**
 - With centralized version control systems, you have a single “central” copy of your project on a server and commit your changes to this central copy
 - You pull the files that you need, but you never have a full copy of your project locally
 - Some of the most common version control systems are:
 - Subversion (SVN) by Apache
 - Perforce by Perforce Software
- **Distributed Version Control System: [DVCS]**
 - With distributed version control systems (DVCS), you don't rely on a central server to store all the versions of a project's files
 - Instead, you clone a copy of a repository locally so that you have the full history of the project
 - Some of most common distributed version control systems are:
 - Git
 - Mercurial

Centralized VCS

Centralized version control system

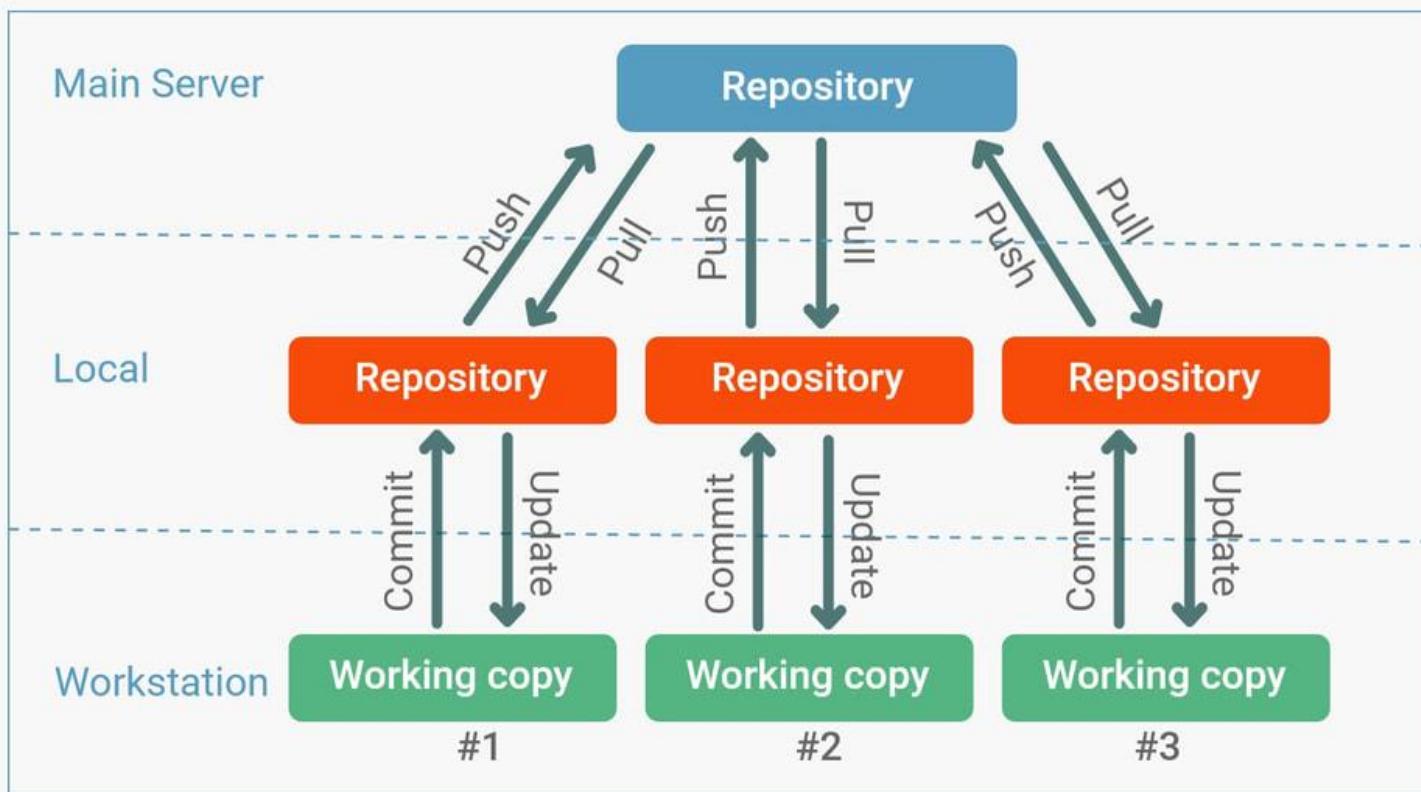


Version Control System

Centralized Version Control System:[CVCS]

- Centralized Version Control Systems were developed to record changes in a central system and enable developers to collaborate on other systems
- Advantages of Centralized Version Control Systems:
 - Relatively easy to set up
 - Provides transparency
 - Enable admins control the workflow
- Disadvantages of Centralized Version Control Systems:
 - If the main server goes down, developers can't save versioned changes (Single point of failure)
 - Remote commits may be slow
 - If the central database is corrupted, the entire history could be lost (security issues)

Distributed VCS



Version Control System

Distributed Version Control System:[DVCS]

- They allow developers to clone the repository and work on that version. Developers will have the entire history of the project on their own hard drives
- Advantages of Distributed Version Control Systems:
 - Performing actions other than pushing and pulling, is extremely fast because the tool only needs to access the hard drive, not a remote server
 - Everything but pushing and pulling can be done offline
- Disadvantages of Distributed Version Control Systems:
 - If your project contains many large files, then the space will be more on local drives
 - If your project has a very long history then downloading the entire history can take an impractical amount of time

Version Control System

What do you want from your version control system?

- **History Tracking** - VCS records a history of changes made to files, including who made each change, when it was made, and what changes were introduced.
- **Revision Management** - VCS allows users to create, update, and revert revisions of files. Each revision represents a snapshot of the file at a specific point in time, enabling users to access previous versions of the code or files as needed.
- **Branching and Merging** - VCS supports branching, which allows users to create separate lines of development for features, bug fixes, or experiments without affecting the main codebase. Branches can be merged back into the main codebase when changes are complete and tested.
- **Collaboration** - Multiple developers can work concurrently on the same files

Version Control System

What do you want from your version control system?

- **Conflict Resolution** - In cases where multiple users make conflicting changes to the same file, VCS helps identify and resolve conflicts by highlighting the differences and allowing users to manually resolve them.
- **Auditing and Accountability** - VCS provides auditing capabilities by tracking all changes made to files and attributing them to specific users.
- **Backup and Disaster Recovery** - VCS serves as a backup mechanism by storing copies of files and their entire revision history. In case of accidental deletions, data loss, or system failures, VCS enables users to restore previous versions of files and recover lost work.

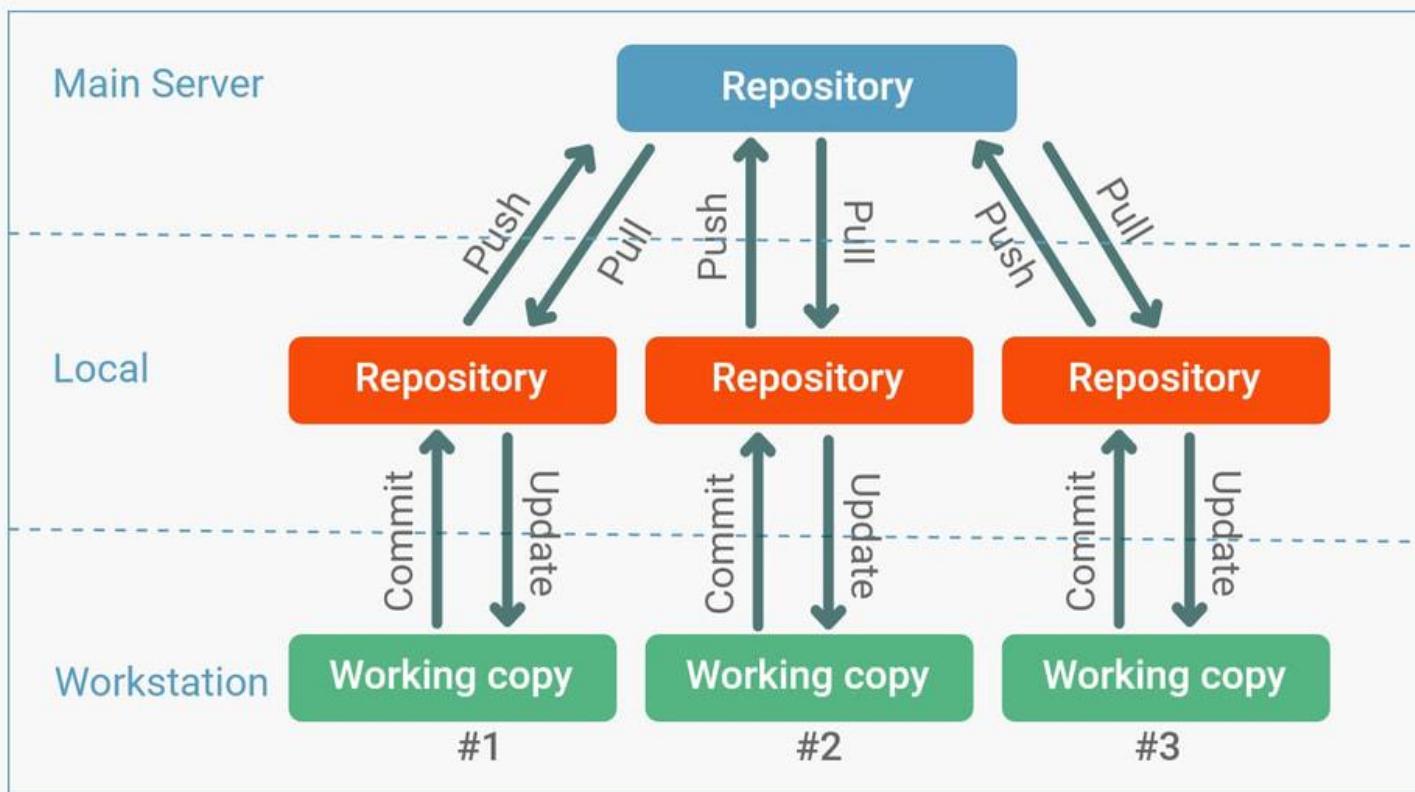


Introduction to GIT



[Ref: “ProGit” by Scott Chacon and Ben Straub, Second Edition, 2023]

Distributed VCS

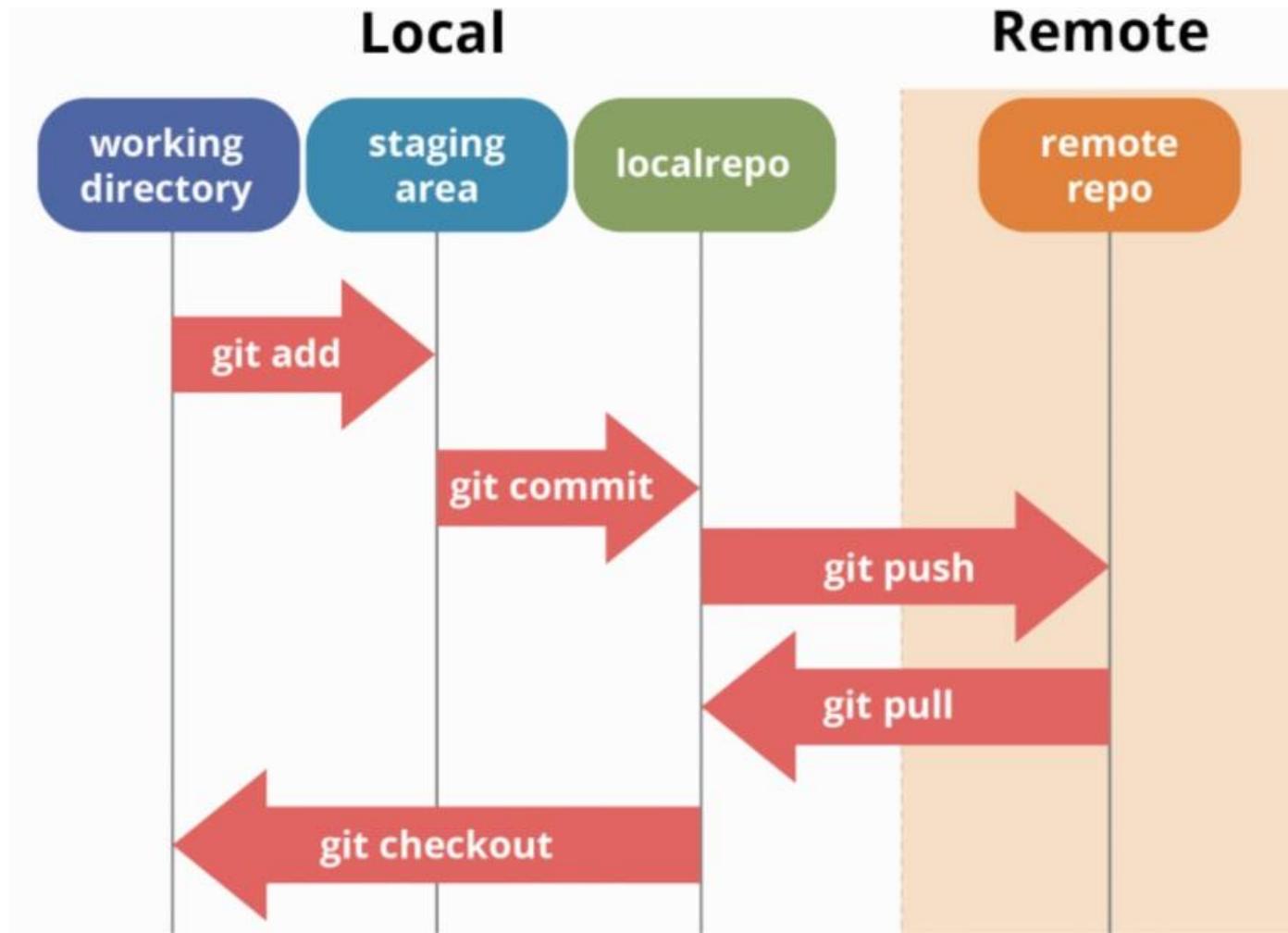


About GIT

- Git is a distributed version control system (VCS) that is widely used for tracking changes in source code during software development.
- It allows multiple developers to work on a project simultaneously and keeps track of all modifications made to the codebase over time.
- Git was created by Linus Torvalds, the creator of the Linux operating system, in 2005.



Basic GIT Workflow



Basic GIT Workflow

1. Working Directory:

- The working directory is where you do your work. It's a directory on your local machine where you have all your project files.
- This is where you edit and modify your files.
- Files in the working directory can be in various states: untracked (new files), modified (existing files with changes), or staged (files ready to be committed).



Basic GIT Workflow

2. Staging Area (Index)

- The staging area is an intermediate area between your working directory and the local repository.
- It acts as a holding area for changes that you want to include in your next commit.
- You use the “git add” command to move changes from the working directory to the staging area.
- Files in the staging area are not yet committed to the repository but are staged and ready to be included in the next commit.



Basic GIT Workflow

3. Local Repository

- The local repository is where Git stores all the committed changes for your project on your local machine.
- It resides in the “.git” directory at the root of your project.
- When you commit changes using “git commit” command, Git creates a *snapshot of the files* in the staging area and stores it in the local repository.
- The local repository maintains the complete history of your project, including all commits, branches, tags, and other metadata.



Basic GIT Workflow

4. Remote repository

- The remote repository is a version of your project hosted on a remote server, such as GitHub, GitLab, or Bitbucket.
- It serves as a centralized location where multiple developers can collaborate on the same project.
- You can push changes from your local repository to the remote repository using the “git push” command.
- Likewise, you can fetch changes from the remote repository to your local repository using the “git fetch” or “git pull” commands.





Git Basic Commands



Git Basic Commands

Getting a Git Repository or Creating a Git Repository

- You typically obtain a Git repository in one of two ways
 - You can take a local directory that is currently not under version control, and turn it into a Git repository
 - OR
 - You can clone an existing Git repository from elsewhere
- Initializing a Repository in an Existing Directory
 - Go to the Project Directory which is not under version control
 - And type

#git init



Git Basic Commands

Cloning an Existing Repository

- If you want to get a copy of an existing Git repository
- Instead of getting just a working copy, Git receives a full copy of nearly all data that the server has
- Every version of every file for the history of the project is pulled down by default when you run `git clone`
- Benefit: if your server disk gets corrupted, you can often use nearly any of the clones on any client to set the server back to the state it was in when it was cloned

```
#git clone <url>
```



Git Basic Commands

Getting the Status

- Display the current state of the repository
- Shows changes in the working directory
- Shows changes in the staging area (index)
- Shows which files have been modified, staged, or untracked

#git status



Git Basic Commands

Add file(s) to Staging area

- The Git command to add file(s) to staging area is “git add”
- This command updates the index using the current content found in the working tree, to prepare the content staged for the next commit
- Before using commit command you must use the add any new or modified files to the index

```
#git add <file name>
```



Git Basic Commands

Committing your changes

- Once the staging area is set up, you can commit your changes
- Remember that anything that is still unstaged i.e. any files you have created or modified that you haven't run git add on since you edited them, those won't go into this commit

#git commit

#git commit –m “Commit Message”



GIT stores data as Snapshots and Commit Objects

- Git stores data as a series of snapshots
- Which means when you make a commit, Git stores a commit object that contains a pointer to the snapshot of the content you staged

Snapshots:

- Git captures the state of the entire repository at the time of each commit by taking a snapshot of all the files in the working directory and staging area. This snapshot represents the complete contents of the project at that particular moment.
- Each commit in Git contains a full snapshot of the project's files, not just the changes made since the previous commit.

Commit Objects:

- When you make a commit in Git, Git creates a commit object that contains metadata such as the author, timestamp, commit message, and a reference to the snapshot of the project's files.
- The commit object also includes pointers to the parent commit(s), forming a chain of commits that represents the commit history of the repository.



Git Basic Commands

Removing Files

- Removes file from both staging area and working directory

```
#git rm <filename>
```



Git Basic Commands

Moving Files or Renaming Files in a Git Repository

- Unlike many other VCS systems, Git doesn't explicitly track file movement
- To rename a file in Git, you can run

```
#git mv <file_from> <file_to>
```



Git Basic Commands

Viewing the Commit History of Git Repository

- After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened
- The most basic and powerful tool to do this is the “git log” command
- By default, with no arguments, git log lists the commits made in that repository in reverse chronological order i.e. the most recent commits show up first
- To see only the commits of a certain author, use the “–author” option

#git log

#git log --author=john



Git Basic Commands

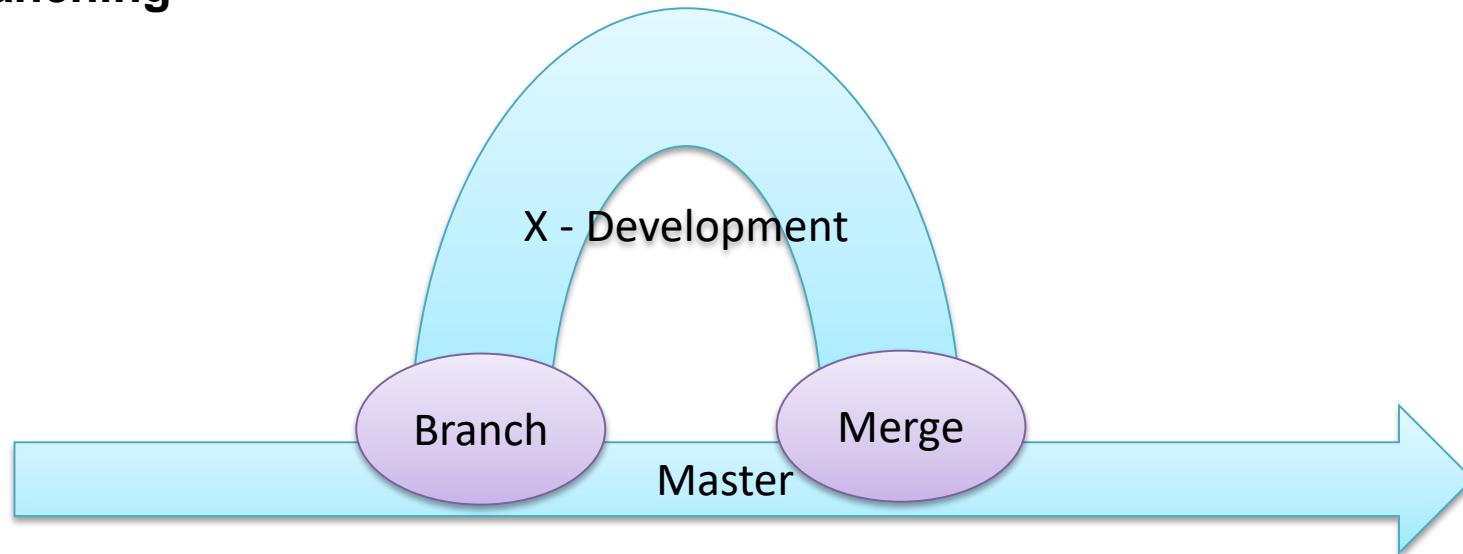
Git Branching

- Every Version Control System has Branching Support
- Branching means you diverge from the main line of development and continue to do work without messing with that main line
- Branches are used to develop features isolated from each other
- The way Git branches is incredibly lightweight, making branching operations nearly instantaneous



Git Basic Commands

Git Branching



```
#git branch <branch name>
```



Git Basic Commands

Git Branching

- What happens if you create a new branch?
- Creation of Branch creates a new pointer for you to move around
- How does Git know what branch you're currently on?
- Git keeps a special pointer called HEAD; HEAD acts as a pointer to the local branch you're currently on
- The git branch command only created a new branch; it didn't switch to that branch, you will be still on master branch
- To switch to an existing branch, you run “git checkout” or “git switch” command

#git checkout <branch name>

#git switch <branch name>



Git Basic Commands

Git Merging

- When to Merge?
- If you have completed the feature development or the Individual task branched, and upon completion of that now you need to add that component to the Main branch
- Use the “git merge” command

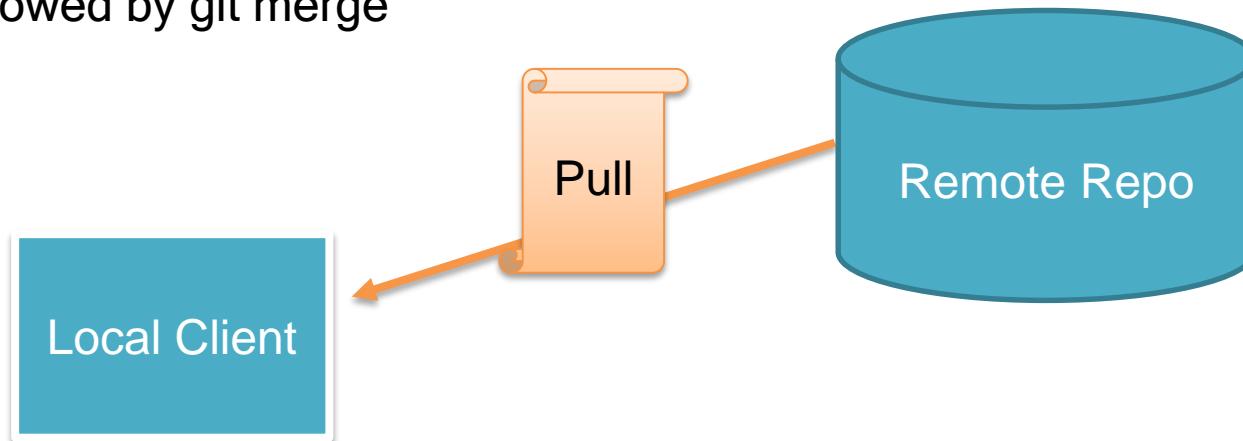
```
#git merge <name of branch>
```



Git Basic Commands

Git Pull

- The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content
- The git pull command is actually a combination of two other commands, git fetch followed by git merge



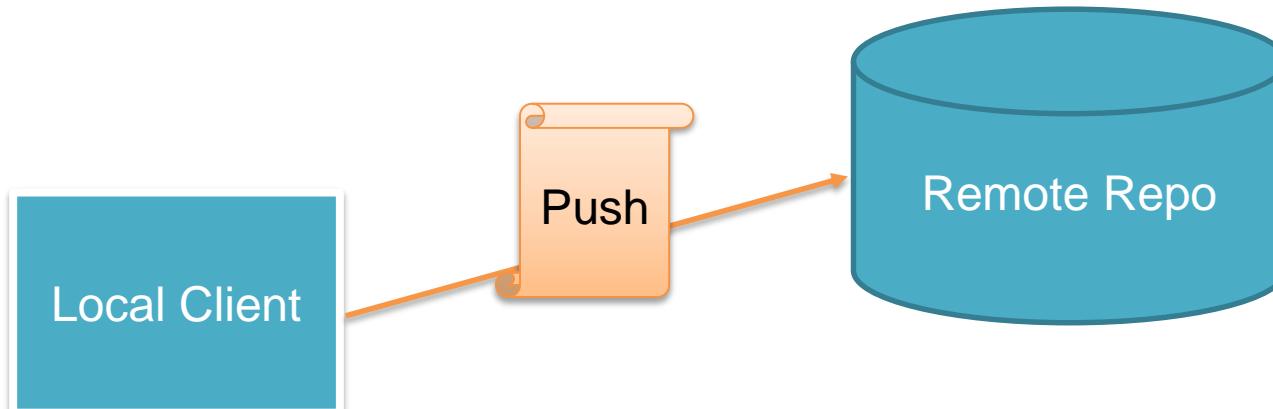
#git pul



Git Basic Commands

Git Push

- The git push command is used to upload local repository content to a remote repository
- Pushing is how you transfer commits from your local repository to a remote repo
- git push is one component of many used in the overall Git "syncing" process



#git push



Git Commands

Basic Git Commands:

Git: configurations

```
$ git config --global user.name "FirstName LastName"  
$ git config --global user.email "your-email@email-provider.com"  
$ git config --global color.ui true  
$ git config --list
```

Git: starting a repository

```
$ git init  
$ git status
```

Git: staging files

```
$ git add <file-name>  
$ git add <file-name> <another-file-name> <yet-another-file-name>  
$ git add .  
$ git add --all  
$ git add -A  
$ git rm --cached <file-name>  
$ git reset <file-name>
```

Git: committing to a repository

```
$ git commit -m "Add three files"  
$ git reset --soft HEAD^  
$ git commit --amend -m <enter your message>
```

Git: pulling and pushing from and to repositories

```
$ git remote add origin <link>  
$ git push -u origin master  
$ git clone <clone>  
$ git pull
```

Git: branching

```
$ git branch  
$ git branch <branch-name>  
$ git checkout <branch-name>  
$ git merge <branch-name>  
$ git checkout -b <branch-name>
```

Demo

Git commands and GitHub Integration

[Refer: “Lab Sheet 1 - Module 3 - Git Commands and GitHub Integration”]



BITS Pilani
Pilani Campus



GitHub



GitHub

- A Cloud-based hosting and collaboration platform which allows professionals to host, store, and record their code in the Github cloud.
- The biggest source code host with over 200 million repositories.
- Hosting is free with an unlimited private and public remote repositories.
- Users can take advantage of the benefits of version history with revision control. This feature, along with pull requests, make collaboration on code files transparent and easier to trace.



GitHub Components

- Repositories
- Branches
- Commits
- Pull Requests
- Git (the version control tool GitHub is built on)

Repository

- A repository stores everything pertinent to a specific project including files, images, spreadsheets, data sets, and videos, often sorted into files.
- ".git" folder inside a project - tracks all changes made to files in your project, building a history over time.
- Each file's revision history.
- Manage your project's work within the repository.
- Lets you and others work together on projects from anywhere.

Branch

- Way to work on a new feature without affecting the main codebase.
- Is a feature of version control systems, which means that Git tracks at which point in the version history you created it.
- Git lets you merge a branch back into mainline, merging them.

Git vs GitHub

GIT VERSUS GITHUB

<p>Git is a distributed version control system which tracks changes to source code over time.</p>	<p>GitHub is a web-based hosting service for Git repository to bring teams together.</p>
<p>Git is a command-line tool that requires an interface to interact with the world.</p>	<p>GitHub is a graphical interface and a development platform created for millions of developers.</p>
<p>It creates a local repository to track changes locally rather than store them on a centralized server.</p>	<p>It is open-source which means code is stored in a centralized server and is accessible to everybody.</p>
<p>It stores and catalogs changes in code in a repository.</p>	<p>It provides a platform as a collaborative effort to bring teams together.</p>
<p>Git can work without GitHub as other web-based Git repositories are also available.</p>	<p>GitHub is the most popular Git server but there are other alternatives available such as GitLab and BitBucket.</p>

Git - Pull Request Life Cycle

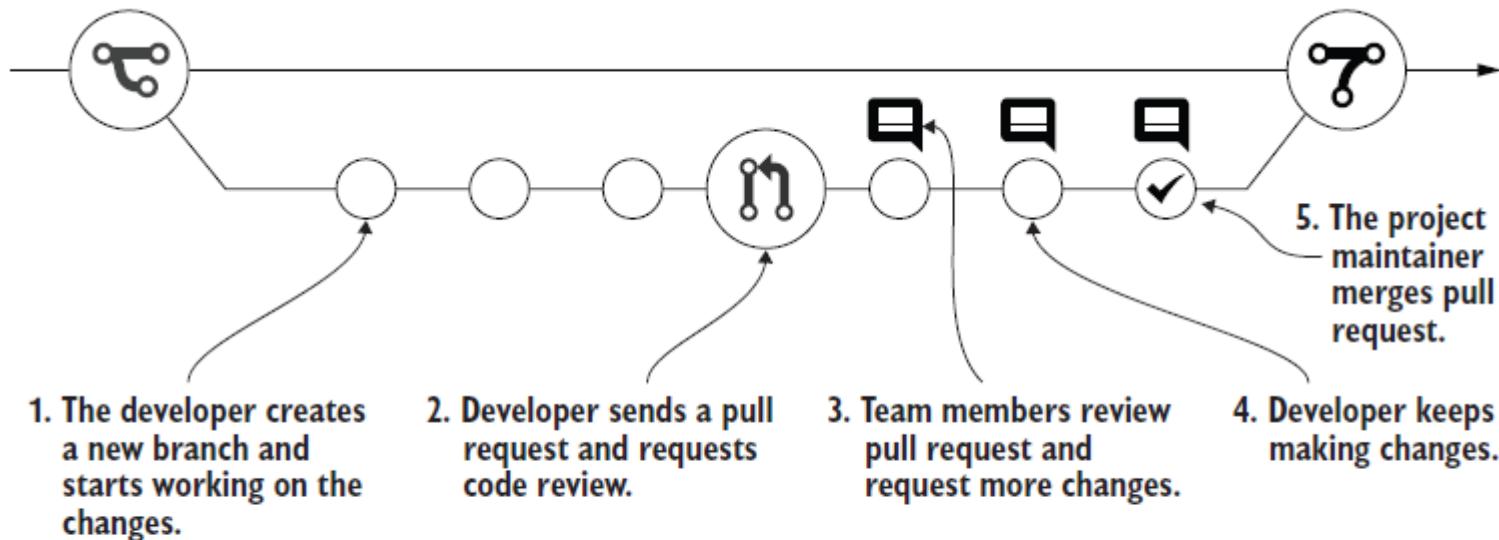


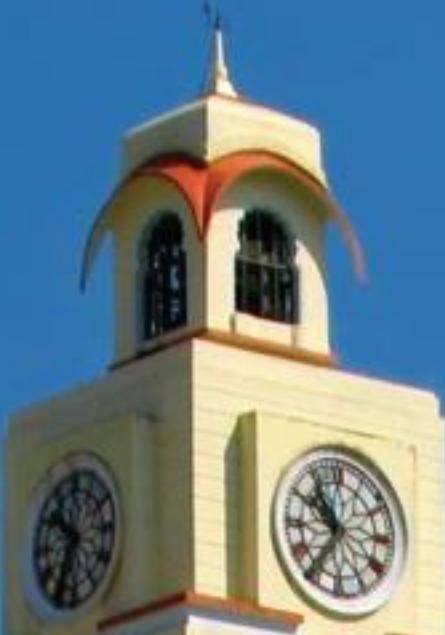
Figure 1.6 The pull request life cycle allows multiple rounds of code review and revisions until the changes are approved. Then the changes may be merged to the main branch and the pull request branch deleted.

Git - Pull Request Life Cycle

1. You create a new branch (often from the main or master branch) in your repository to work on a new feature, bug fix, or other changes.
 2. Once you've made your changes and committed them to your branch, you push that branch to the remote repository (e.g., on GitHub).
 3. You then create a pull request from your branch on GitHub, specifying which branch you want to merge your changes into (e.g., the main or master branch).
 4. The pull request includes information about the changes you've made, allowing other contributors or maintainers of the repository to review the changes, provide feedback, and discuss any potential issues.
 5. If the changes are approved, the maintainers can merge the pull request, effectively integrating your changes into the target branch (e.g., the main or master branch).
-



Thank You!



BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lectures No. 5

Evaluation

Evaluation Component	Name (Quiz, Lab, Project, Midterm exam, End semester exam, etc)	Type (Open book, Closed book, Online, etc.)	Weight	Duration	Day, Date, Session, Time
EC – 1	Quiz 1		5%		February 19-28, 2024
	Quiz 2		5%		March 19-28, 2024
	Assignment I		10%		April 19-28, 2024
	Assignment II		10%		To be announced
EC – 2	Mid-term Exam	Closed book	30%	2 hours	15/03/2024 (FN)
EC – 3	End Semester Exam	Open book	40%	2 ½ hours	17/05/2024 (FN)

Quiz 1: 24-Feb-24 till 26-Feb-24 for 5 Marks [Incl. today's session]

Assignment 1: 24-Feb-24 till 4-Mar-24 for 10 marks

Agenda

Continuous Integration

- Traditional CI setup
- Building the application
- Dependency Management
- Types of testing
- Code quality analysis
- Automated Testing
- Continuous code inspection - Code quality
- Essential CI Practices

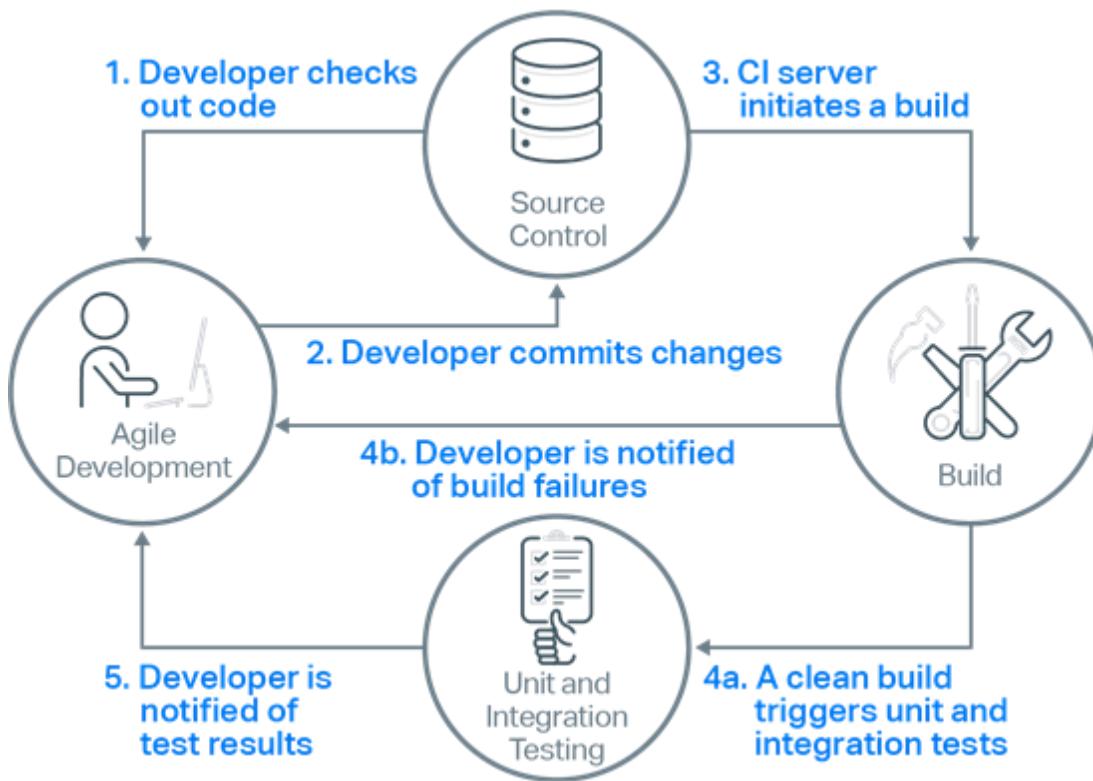


Continuous Integration (CI)

Continuous Integration (CI)

- Continuous Integration is a DevOps practice in which developers regularly commit and push their local changes back to the shared repository (such as GitHub, usually several times a day).
- Before each commit, developers can run unit tests locally on their source code as an additional check before integrating.
- CI entails both an **automation component** (e.g. a CI or build service) and a **cultural component** (e.g. learning to integrate frequently).
- A continuous integration service automatically builds and runs unit tests on the new source code changes to catch any errors immediately.

CI Workflow



Continuous Integration systems will usually run a series of tests automatically upon merging in new changes

The outcome of these tests is often visualized, where “**green**” means the tests passed and the **newly integrated build is considered clean**, and failing or “**red**” tests means the **build is broken and needs to be fixed**

Continuous Integration (CI) Benefits

- Frequent Integration:** Developers integrate their code changes into the main codebase multiple times a day, rather than waiting for long periods before merging their work. This helps in identifying and resolving conflicts and integration issues early in the development process.
- Automated Builds:** CI systems automatically build the integrated codebase whenever changes are pushed to the repository. This includes compiling code, running unit tests, and performing other validation tasks.
- Automated Testing:** CI systems execute automated tests, including unit tests, integration tests, to ensure that the integrated changes haven't introduced any regressions or errors.
- Immediate Feedback:** CI systems provide immediate feedback to developers about the status of their code changes. If the build or tests fail, developers are notified promptly, allowing them to address issues quickly.

Aspects of Continuous Integration

1. Version Control System
 2. Automated Build which includes Dependency Management
 3. Automated Testing
 4. Agreement of the Team (Cultural aspect)
-

1. Version Control

- Everything in a project must be checked in to a single version control repository: code, tests, build scripts, workflow files, and anything else needed to create, install, run, and test your application.
- There are several Version Control Systems available
 - Git
 - Apache Subversion etc

2. Automated Build

- A **build** is a process that compiles and packages source code into an executable form.
- An application might have many dependencies required for successful implementation, so to avoid manually downloading these dependencies ourselves, a **build tool** is used.

In the Build stage, we take the provided code and build it for testing purposes.

<https://medium.com/strategio/the-build-process-9fe14e844019>

Overview of Dependencies

- A dependency occurs whenever one piece of software depends upon another in order to build or run
- In most trivial of applications, there will be some dependencies
- Most software applications have a dependency on their host operating/ runtime environment
- Like Java applications depend on the JVM which provides an implementation of the Java SE API, .NET applications on the CLR, Rails applications on Ruby and the Rails framework, C applications on the C standard library etc.

Types of Dependencies

1. Libraries and Components
2. Build time dependencies and Run time dependencies

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.
- For example, do you compile your entire application in a single step, or compile each component independently when it changes?

Build time dependencies and Run time dependencies

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

Ex. For Python based Flask Application

Build dependency -> “pip install Flask-RESTful”
(library to build REST APIs using Flask framework -- required in build phase)

Runtime dependency -> “pip install Flask-SQLAlchemy” (For interactions with Database -- required during the execution)

In requirements.txt file

```
# Build-time dependencies  
Flask==2.0.2  
Flask-RESTful==0.3.9  
# Runtime dependencies  
Flask-SQLAlchemy==3.0.0
```

Common dependency problem with libraries at run time

Here are some of the most common ones:

1. **Missing dependencies:** This occurs when a required dependency is not installed or available on the system. This can happen when a developer forgets to include a necessary dependency in the project, or when a package manager fails to install the dependency due to an error or conflict.
2. **Version conflicts:** This occurs when different dependencies require different versions of the same software library. This can lead to compatibility issues and unexpected behavior in the software. It can also make it difficult to manage dependencies and ensure that the software works correctly.
3. **Circular dependencies:** This occurs when two or more dependencies rely on each other in a circular manner. This can lead to a situation where it's impossible to install or update the dependencies, and can cause unexpected behavior in the software.
4. **Security vulnerabilities:** This occurs when a dependency has a known security vulnerability that can be exploited by attackers. It's important to regularly check for security vulnerabilities in dependencies and update them to the latest secure version.
5. **Deprecated dependencies:** This occurs when a dependency is no longer maintained or updated by the developer, making it obsolete or outdated. This can lead to compatibility issues and security vulnerabilities.

Managing Dependencies - Automated

To manage libraries in a software project in an automated way, you can leverage dependency management tools and package managers.

Managing Dependencies - Automated

Ex 1: Pip and requirements.txt (for Python projects)

- Create a “requirements.txt” file in your project's root directory
- Add the names and versions of your project dependencies in the requirements.txt file, each on a separate line.
- Use the command “pip install -r requirements.txt” to automatically install all the dependencies listed in requirements.txt.
- This command will resolve dependencies and download the required libraries.

Managing Dependencies - Automated

Ex 2: Gradle/Maven (for Java projects)

- For Maven, you manage dependencies in the “pom.xml” file, while for Gradle - you manage them in the “build.gradle” file.
- Add your project dependencies in the appropriate section of the configuration file.
- Use commands like “mvn install” or “./gradlew build” to automatically resolve dependencies and download the required libraries.

Managing Dependencies - Automated

Ex 3: Composer (Dependency Manager for PHP projects)

- Create a “composer.json” file in your project's root directory
- Add your project dependencies in the “require” section of the composer.json file.
- Use the command – “composer install” to automatically install all the dependencies listed in composer.json.
- This command will resolve dependencies and download the required libraries.
- Composer is used in Laravel, Symfony, Zend frameworks for dependency management

Managing Dependencies - Automated

Ex 4: Docker

- Use Docker to create containers for your application and its dependencies
- Define your project dependencies in a “Dockerfile” using the appropriate package manager for your language (e.g., npm, pip, Maven).
- Build your Docker image, which will automatically download and install the required libraries specified in the Dockerfile.
- Run your application within a Docker container, ensuring that all dependencies are managed and isolated within the container environment.

Dockerfile

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY ..
```

pip: for Python projects.

```
FROM maven:latest
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline
COPY src ./src
RUN mvn package
```

Maven: for Java projects.

```
FROM node:latest
WORKDIR /app
COPY package*.json .
RUN npm install
COPY ..
```

npm (Node.js/npm): for JavaScript and Node.js projects.

Build Tools

Technology	Tool
Rails	Rake
.Net	MsBuild
Java	Ant, Maven, Buildr, Gradle
C,C++	Scons
Python	PyBuilder

3. Testing

- Software Testing is a method to check whether the actual software product matches expected requirements
- A *test case* has a set of test inputs, execution conditions, and expected results developed for a particular objective

Testing strategies

1. **Unit tests** - Test a small part of a service, such as a class.
2. **Integration tests** - Verify that a service can interact with infrastructure services such as databases and other application services.
3. **API tests** - Test responses from APIs / Gateways
4. **End-to-end tests** - Acceptance tests for the entire application.

Unit Testing

- Unit testing is a practice of testing the smallest portions of service or business logic that can be tested in isolation.
- Unit testing helps to ensure that each service/ component performs as expected and is less prone to failure

Automation Frameworks for Unit Testing

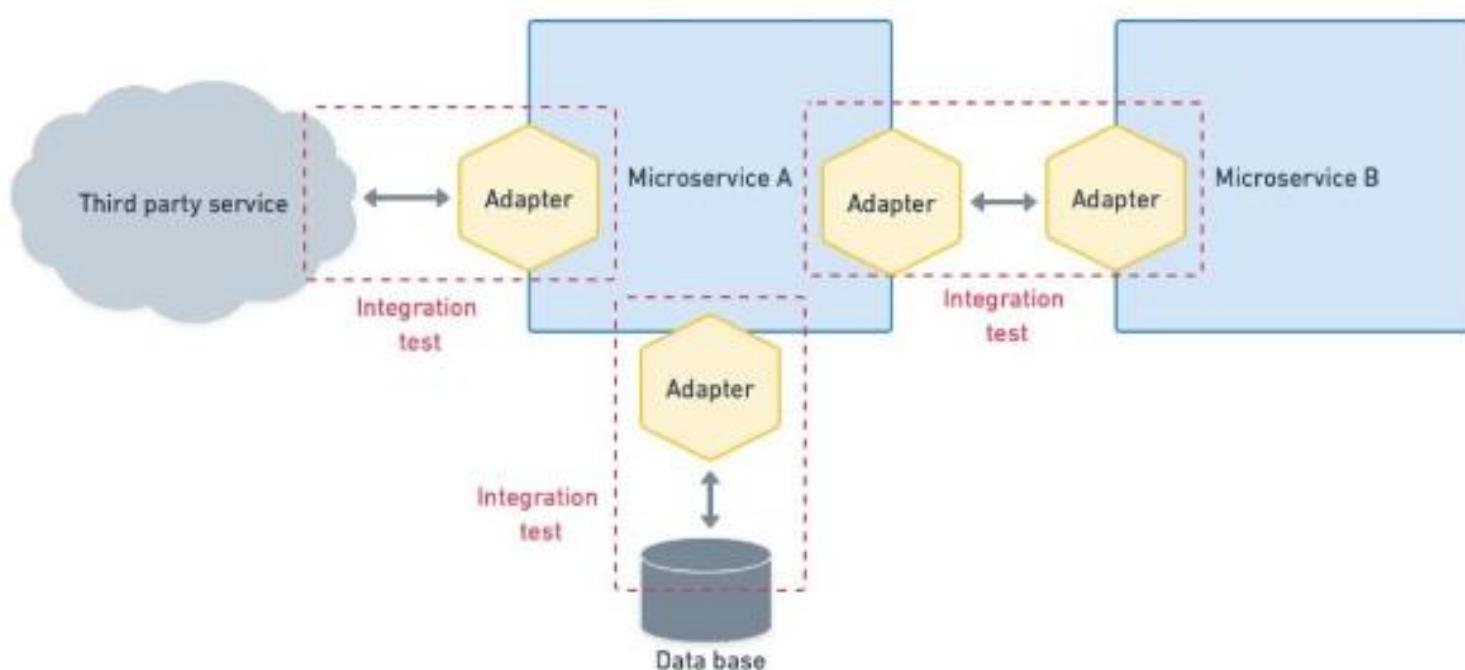
- Pytest (Python)
- UnitTest (Python)
- JUnit (Java)
- NUnit (C#.Net and VB.Net)
- PHPUnit (PHP)

<https://semaphoreci.com/blog/test-microservices>

Integration Tests

- The goal is to **identify interface defects** by making microservices interact.
- Integration tests use real services.
- Integration tests are not interested in evaluating behavior or business logic of a service. Instead we want to make sure that the services can communicate with one another and their own databases.
- We're looking for things like missing HTTP headers and mismatched request/response pairings.

Integration Tests



Using integration tests to check that the microservices can communicate with other services, databases, and third party endpoints.

Integration Tests

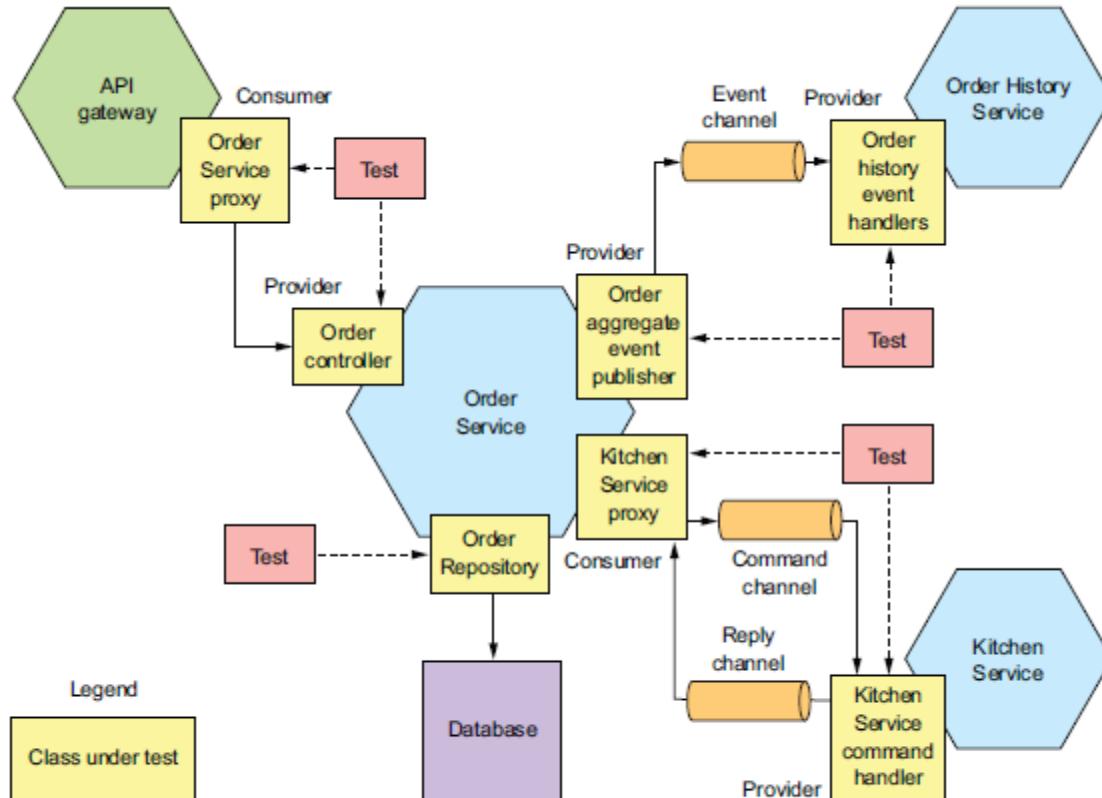


Figure 10.1 Integration tests must verify that a service can communicate with its clients and dependencies. But rather than testing whole services, the strategy is to test the individual adapter classes that implement the communication.

API Testing

- To test responses from APIs / Gateways

Popular tools

- Postman
- Telerik Fiddler [View -> Tabs -> API Tests] - Automate API Calling
- API Testing with Fiddler - <https://www.telerik.com/blogs/api-testing-with-telerik-fiddler>

4. Agreement of the Team (Cultural aspect)

- This is more about People & Culture
- **Continuous integration is a practice, not a tool**
- It requires a degree of commitment and discipline from development team or people involved
- **“Fix first before Proceed”** - before continuing with a task or process, any existing issues or problems should be addressed and resolved first.
- 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

Demo

Objective

- To demonstrate the basic process of “build” and “test” using PyBuilder automation tool

Lab Sheet 2 - Module 4 - Continuous Integration



Continuous Integration Tools

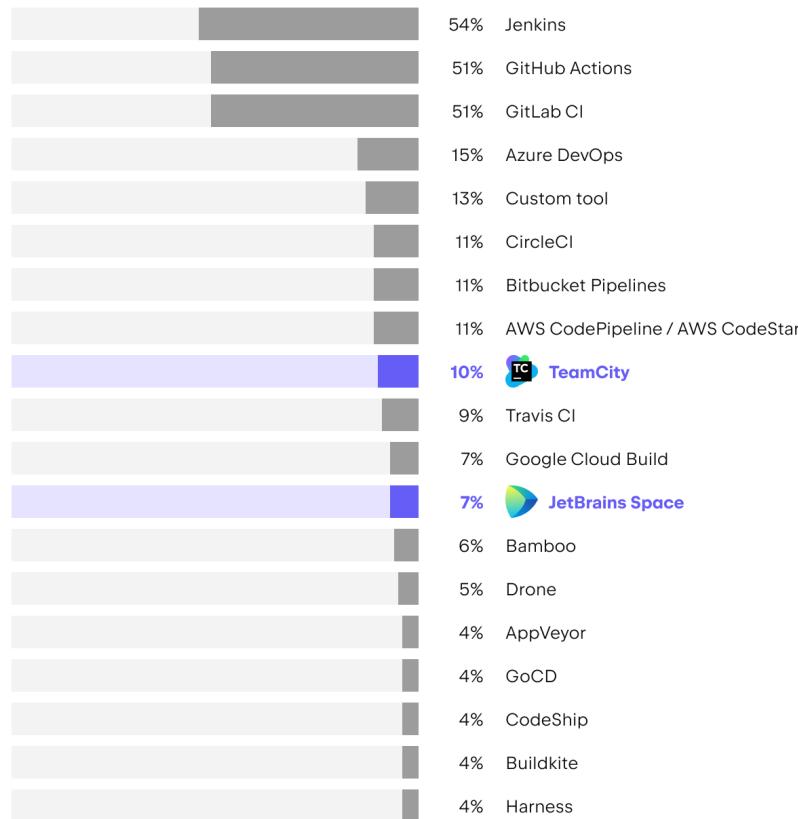
What is a CI tool

- A CI tool is a platform that automates the process of building, testing, and publishing your software.
- CI tools integrate with version control system so they can fetch the latest changes from your repository.
- Most CI tools consist of a central server (often known as a build server) and one or more build agents (or runners) that run on separate machines.
- The build server provides a UI for configuring your build and test pipelines, stores the details of each job, and initiates each pipeline run.
- The build, test, and other tasks are distributed to the build agents by the build server. If multiple build agents are available, multiple pipelines can run at the same time, and tasks from the same pipeline can be run in parallel.

<https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/>

CI Tool – Survey 2024

Which Continuous Integration (CI) systems do you regularly use?



<https://blog.jetbrains.com/teamcity/2023/07/best-ci-tools/>

GitHub Actions

- GitHub's native CI/CD and automation system
- GitHub Actions was introduced by GitHub in 2018.
- It is integrated right into GitHub and enabled by default in every GitHub repository.

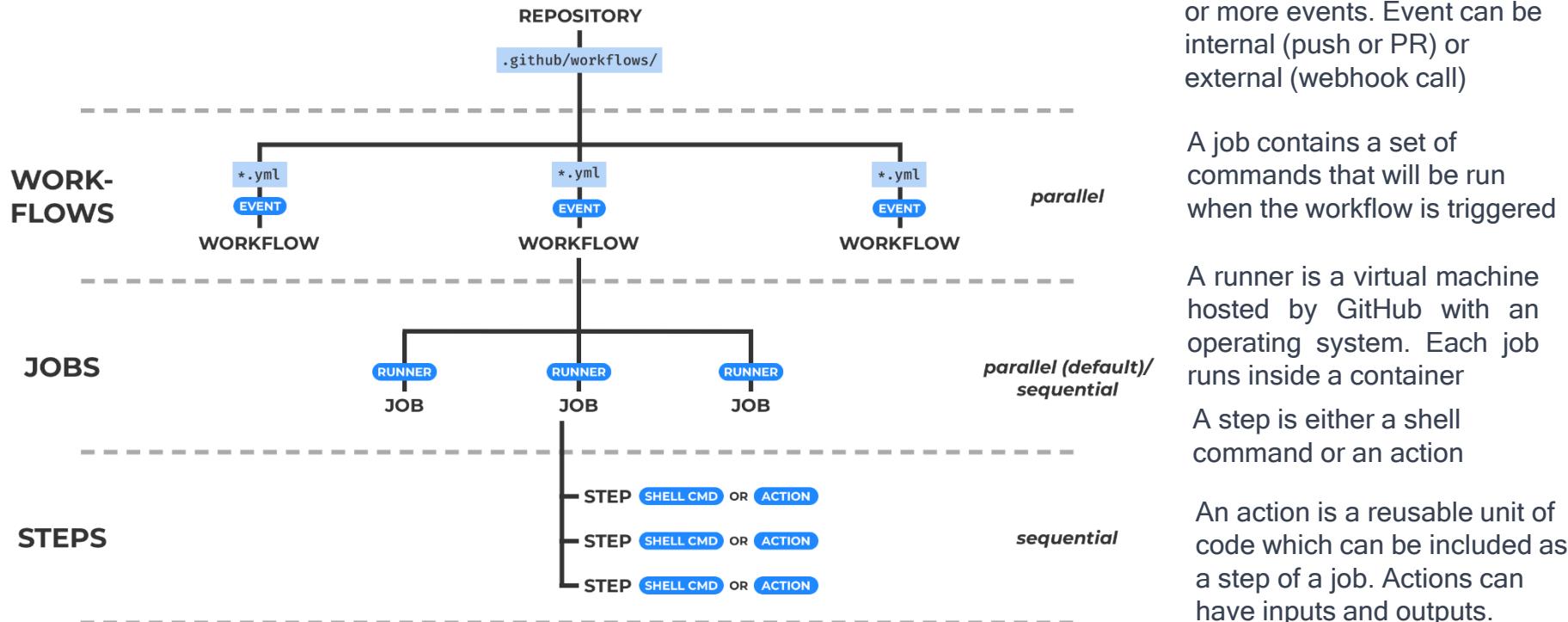
Typical GitHub Actions – Workflow structure

```
your-repository/
|-- .github/
|   |-- workflows/
|       |-- python-app.yml # Your GitHub Actions workflow file
|-- app/
|   |-- main.py
|-- tests/
|   |-- test_main.py
|-- requirements.txt
|-- Dockerfile
|-- README.md
|-- ... (other project files)
```

GitHub Actions



GitHub Actions



Demo

Objective

1. Use GitHub Actions to define a Continuous Integration Workflow / Pipeline for the Python Project
2. Demonstrate the Workflow with Build, Lint, and Test as jobs in the pipeline

Lab Sheet 2 - Module 4 - Continuous Integration



CI Best Practices

Continuous Integration Best Practices

1. Maintain a code repository
2. Automate the build
3. Make the build self-testing
4. Everyone commits to the baseline every day
5. Every commit (to baseline) should be built
6. Keep the build fast
7. Test in a clone of the production environment
8. Make it easy to get the latest deliverables
9. Everyone can see the results of the latest build
10. Automate deployment

Continuous Integration Practices

1. Maintain a code repository

- This practice advocates the use of a revision control system for the project's source code. All artifacts required to build the project should be placed in the repository. In this practice and the revision control community, the convention is that the system should be buildable from a fresh checkout and not require additional dependencies. It is preferred for changes to be integrated rather than for multiple versions of the software to be maintained simultaneously. The mainline should be the place for the working version of the software.

2. Automate the build

- A single command should have the capability of building the system. Many build tools, have existed for many years. Other more recent tools are frequently used in continuous integration environments. Automation of the build should include automating the integration, which often includes deployment into a production-like environment. In many cases, the build script not only compiles binaries but also generates documentation, website pages, statistics and distribution media

Continuous Integration Practices

3. Make the build self-testing

- Once the code is built, all tests should run to confirm that it behaves as the developers expect it to behave.

4. Everyone commits to the baseline every day

- By committing regularly, every committer can reduce the number of conflicting changes. Checking in a week's worth of work runs the risk of conflicting with other features and can be very difficult to resolve. Early, small conflicts in an area of the system cause team members to communicate about the change they are making. Committing all changes at least once a day is generally considered part of the definition of Continuous Integration. In addition, performing a nightly build is generally recommended. These are lower bounds; the typical frequency is expected to be much higher.

Continuous Integration Practices

5. Every commit (to baseline) should be built

- The system should build commits to the ***current working version*** to verify that they integrate correctly. A common practice is to use Automated Continuous Integration, although this may be done manually. Automated Continuous Integration employs a continuous integration server or daemon to monitor the revision control system for changes, then automatically run the build process.

6. Every bug-fix commit should come with a test case

- When fixing a bug, it is a good practice to push a test case that reproduces the bug. This avoids the fix to be reverted, and the bug to reappear, which is known as a regression.

7. Keep the build fast

- The build needs to complete rapidly so that if there is a problem with integration, it is quickly identified.

Continuous Integration Practices

8. Test in a clone of the production environment

- Having a test environment can lead to failures in tested systems when they deploy in the production environment because the production environment may differ from the test environment in a significant way. However, building a replica of a production environment is cost-prohibitive. Instead, the test environment or a separate pre-production environment ("staging") should be built to be a scalable version of the production environment to alleviate costs while maintaining technology stack composition and nuances.

Continuous Integration Practices

9. Everyone can see the results of the latest build

- It should be easy to find out whether the build breaks and, if so, who made the relevant change and what that change was.

10. Automate deployment (CD)

- Most CI systems allow the running of scripts after a build finishes. In most situations, it is possible ***to write a script to deploy the application to a live test server*** that everyone can look at. A further advance in this way of thinking is continuous deployment, which calls for the software to be deployed directly into production, often with additional automation to prevent defects or regressions.



Thank You!



BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lecture No. 6

Agenda

Continuous Integration and Continuous Delivery using GitOps

- What is GitOps
- GitOps Models
- GitOps Continuous Integration (CI)
 - CI stages
- GitOps Continuous Delivery (CD)
 - CD stages



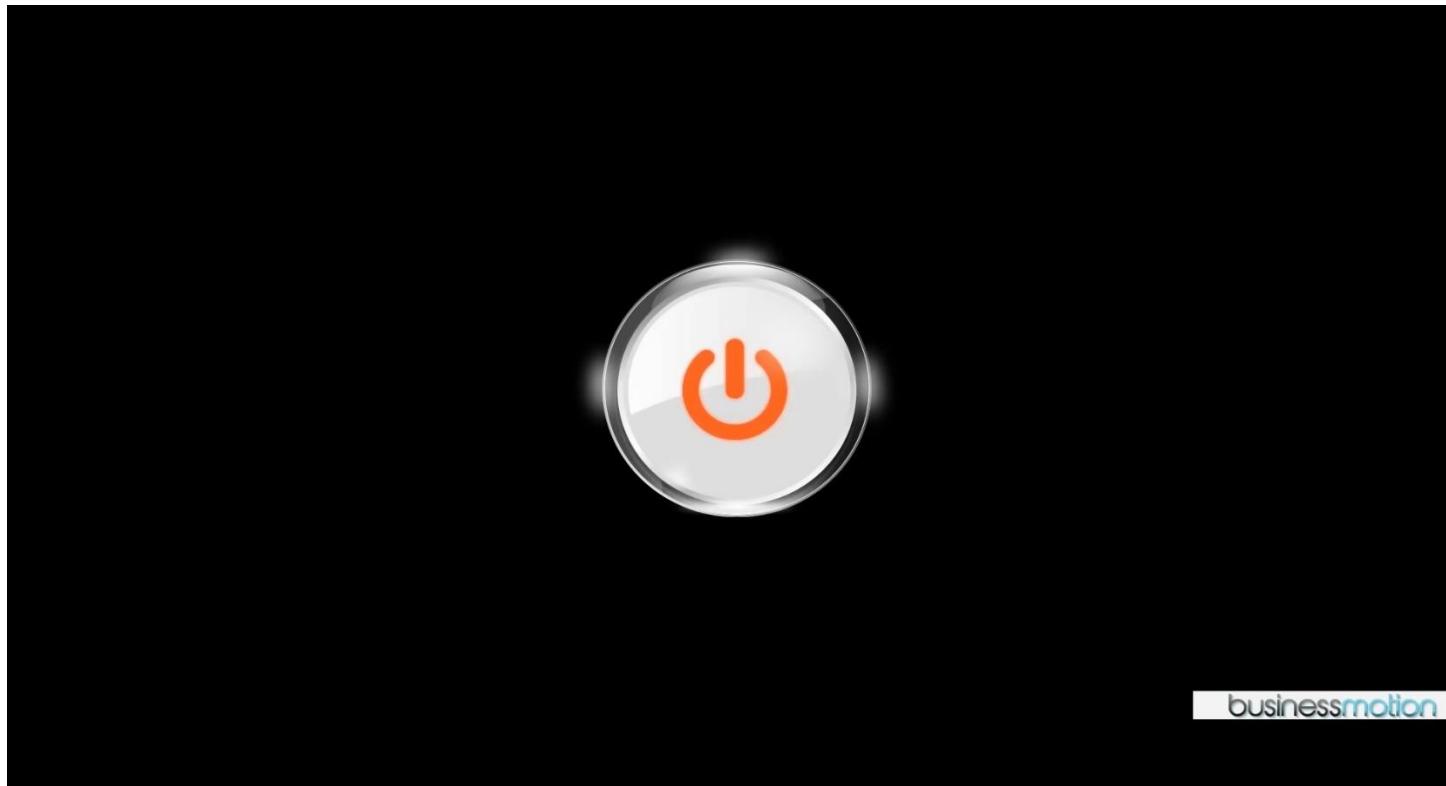
GitOps

Ref: T2 -> “GitOps and Kubernetes”

GitOps

- Term coined in 2017 in a series of Blogs by Alexis Richardson, cofounder and CEO of Weaveworks.
- Term is used in Cloud Native Community and Kubernetes Community
- GitOps is a set of best practices and principles for *managing applications* and **infrastructure configurations** using **Git** as the single source of truth for **declarative configuration** and version control.
- Developer-centric experience for managing applications with fully automated pipelines or workflows.

What is GitOps and How it Works!



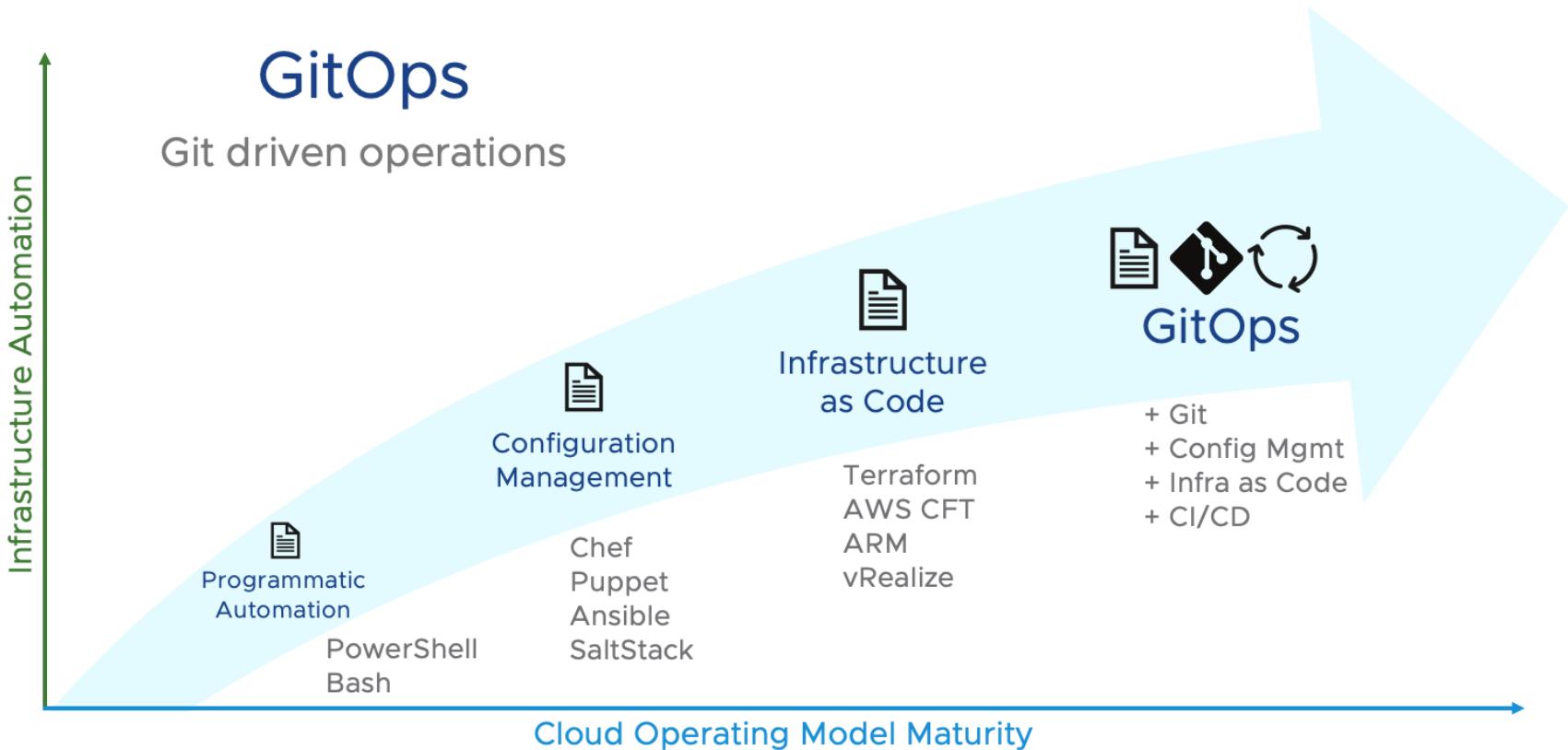
<https://www.youtube.com/watch?v=f5EpcWp0THw>

[Duration – 10 minutes]

GitOps Definitions

- Atlassian describes GitOps as “code-based infrastructure and operational procedures that rely on Git as a source control system”
- GitLab describes “an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation”

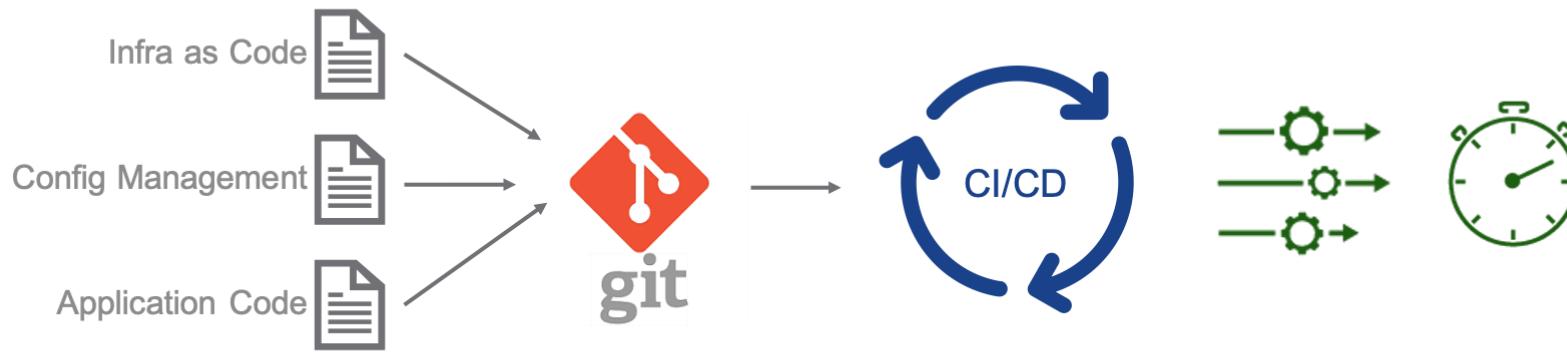
GitOps Evolution



<https://blogs.vmware.com/cloud/2021/02/24/gitops-cloud-operating-model/>

GitOps

GitOps-in-a-nutshell



“Source of Truth” for
declarative code

Update to code source
triggers a pipeline

Pipeline runs a series of tasks, resulting in the
update of the runtime environment to match
the source

GitOps Practices

GitOps = XaC + PRs + CI/CD

1. XaC -> X as Code [Version Control]

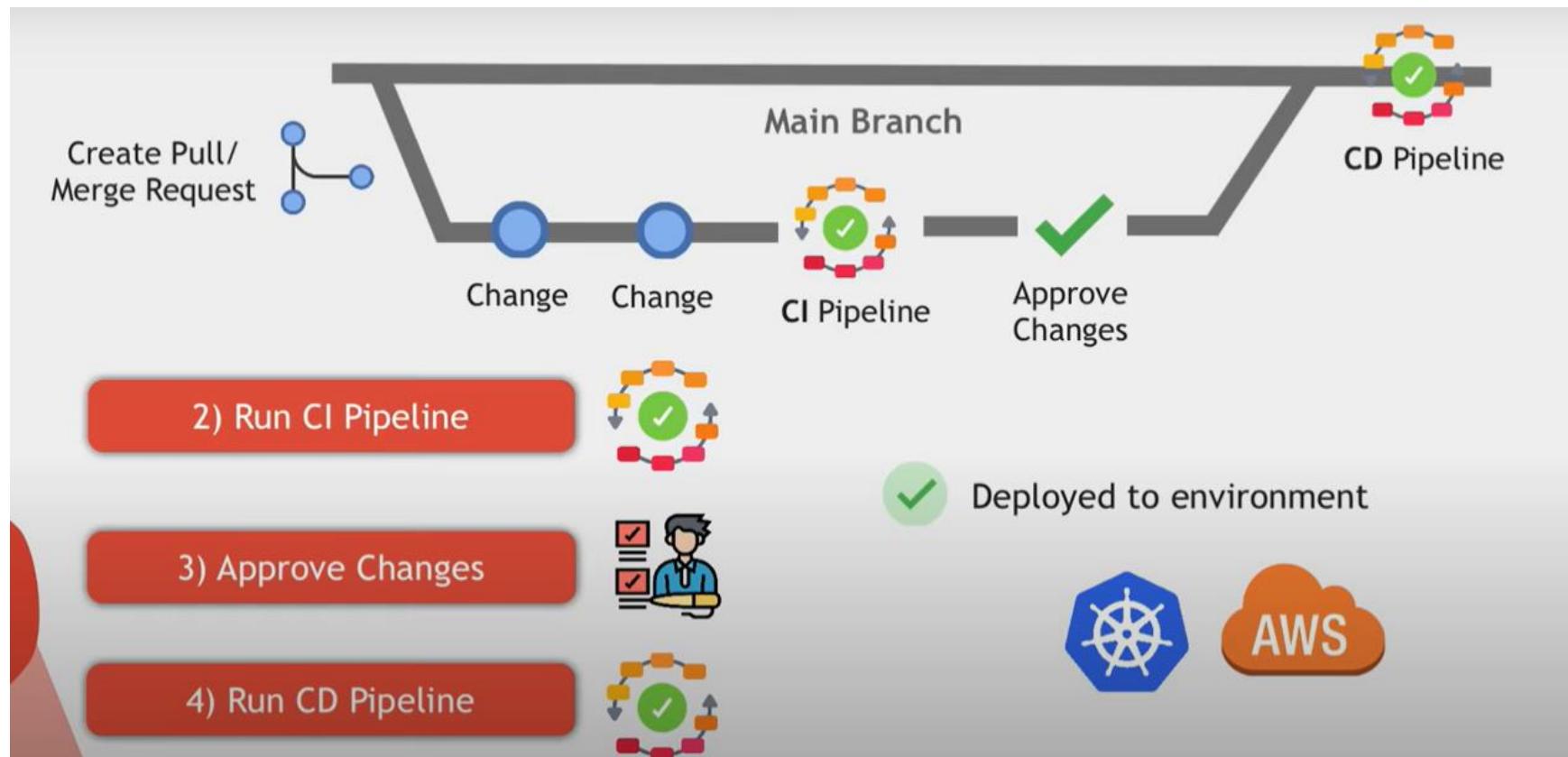
Keep Code, Infrastructure, Configurations in Git

[Ex: Dockerfile, main.yaml (github workflow PaC), manifest.yaml, IaC, etc.)

2. PRs -> Pull requests as the central point of developer collaboration for code review and change orchestration. Used to start the workflow.

3. CI/CD -> Automate all changes made to environments, via CI/CD

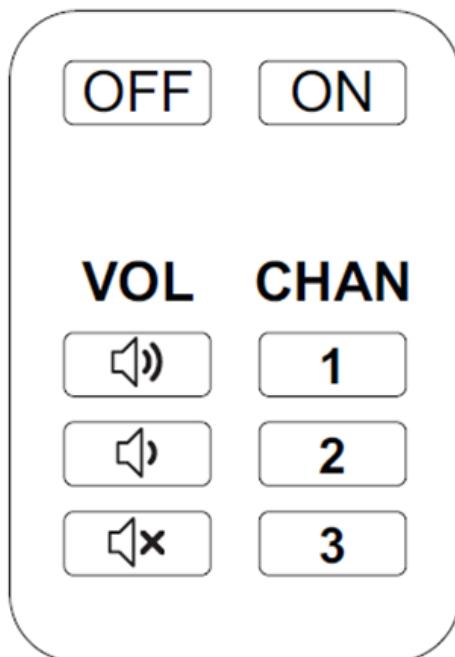
GitOps Workflow – Pull Request Life Cycle



[https://dzone.com/refcardz/the-essentials-of-gitops#:~:text=While%20logically%20GitOps%20is%20simply,delivery%20\(CI%2FCD\)](https://dzone.com/refcardz/the-essentials-of-gitops#:~:text=While%20logically%20GitOps%20is%20simply,delivery%20(CI%2FCD))

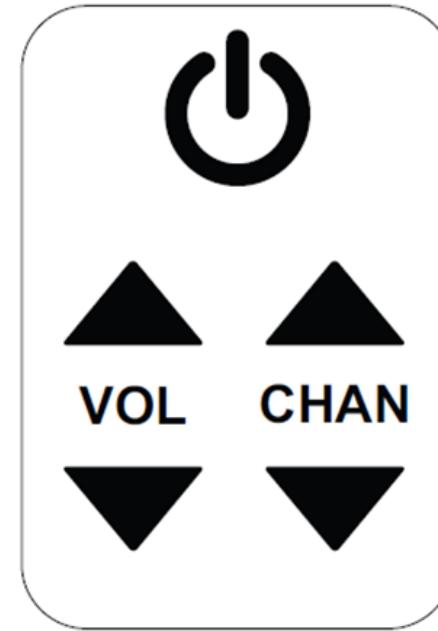
Declarative Models vs Imperative Models

Declarative model - You describe what you want to achieve as opposed to how to get there. Ex: YAML files in CI/CD pipelines



Declarative remote control

Imperative model - you describe a sequence of instructions for manipulating the system to reach your desired state.



Imperative remote control

Declarative Models vs Imperative Models

- **Declarative:**
 - Focuses on **what the desired state is.**
 - **Describes the end result without detailing the steps.**
 - Commonly used in configuration management tools like Ansible, Puppet, and Terraform.
- **Imperative:**
 - Focuses on **how to achieve the desired state.**
 - **Provides step-by-step instructions for execution.**
 - Commonly used in scripts and procedural programming languages.

Declarative Models vs Imperative (Procedural) Models

DECLARATIVE VS. PROCEDURAL

Declarative	"Make me a cake."
Procedural	Mix 3 cups of flour, 2 ¼ cups of sugar, 2 tsp. of baking powder, 1 tsp. salt, 3 sticks of butter, ½ tsp. of vanilla extract, 1 cup milk, and 8 large egg whites. Bake at 350 for 40 minutes.



Declarative Models vs Imperative Models

- *Idempotency* is a property of an operation whereby the operation can be performed any number of times and produce the same result
- In other words, an operation is said to be idempotent if you can perform the operation an arbitrary number of times, and the system is in the same state as it would be if you had performed the operation only once.
- Idempotency is one of the properties that distinguish declarative systems from imperative systems.
- Declarative systems are idempotent; imperative systems are not.
- Ex: “Make me a cup of cake” - said three times [Declarative]

Vs

- “Mix Three Cups of Flour” - said three times [Imperative]
-

Declarative CI pipeline configuration file in GitLab CI [What needs to be done]

- “gitlab-ci.yml” file serves as a declarative description of the CI workflow, defining the desired steps to be executed in each stage of the pipeline.
- The CI platform (GitLab) interprets this configuration file and automates the execution of the defined pipeline whenever changes are detected in the repository.
- **Only the Desired state of the Pipeline is mentioned in the YAML file**

Same example using Imperative Style

[How it needs to be done]

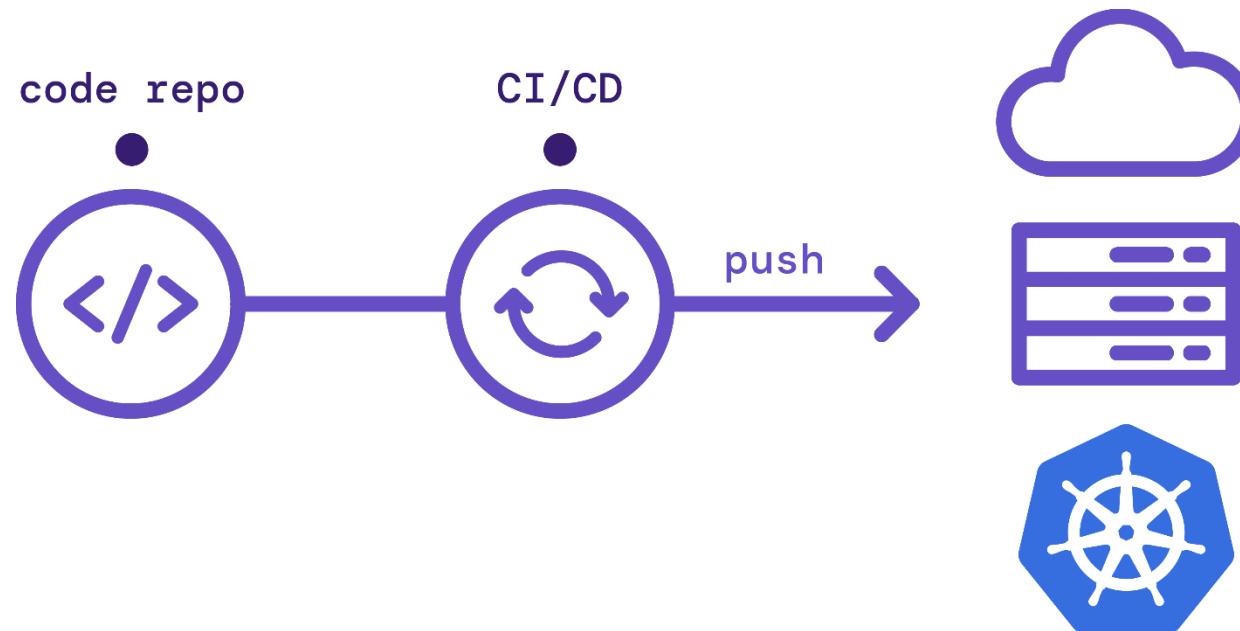
```
before_script:  
  - python3 -m venv venv  
  - source venv/bin/activate  
  - pip install docker  
  
build_and_deploy:  
  script:  
    - echo "Building Docker image..."  
    - docker build -t myapp .  
    - echo "Logging in to Docker registry..."  
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY  
    - echo "Tagging Docker image..."  
    - docker tag myapp:latest $CI_REGISTRY_IMAGE:latest  
    - echo "Pushing Docker image to registry..."  
    - docker push $CI_REGISTRY_IMAGE:latest
```

GitOps Models

Agentless GitOps Model [Push Model]

Agentless GitOps is a traditional model, also known as "push-based GitOps", in which your CI/CD tool reads from your Git repository and pushes changes into your environment. [No agent within the deployment environment]

Ex: Jenkins

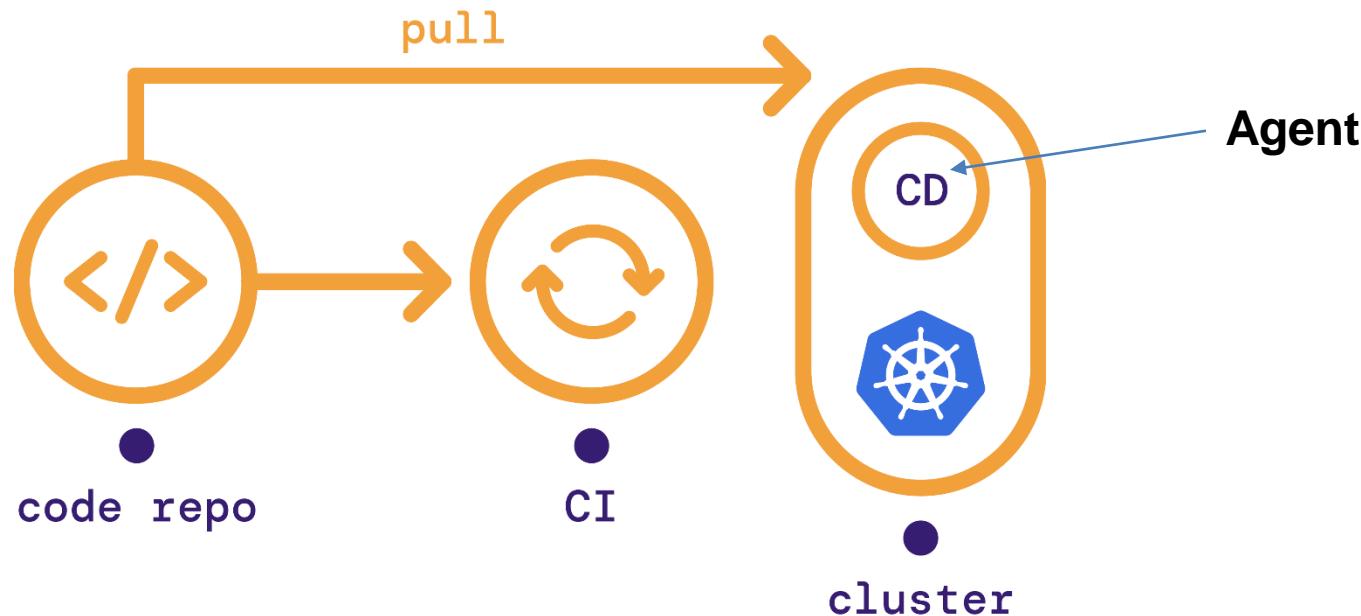


GitOps Models

Agent-based GitOps Model [Pull Model]

Also known as “pull-based” GitOps, uses an agent that runs inside your infrastructure. This agent pulls changes in from an external Git repository when it detects that the state of the environment is out of sync with the source of truth.

Ex: ArgoCD, FluxCD



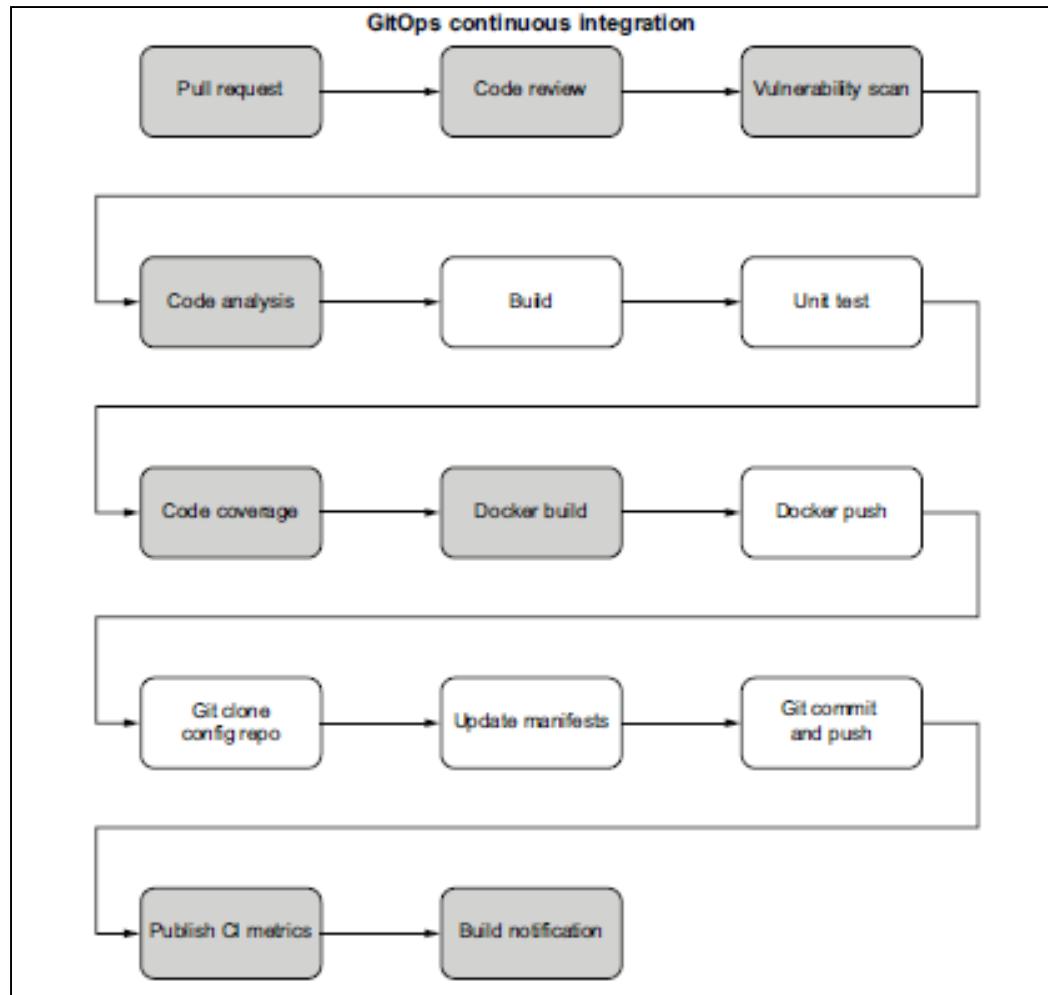
GitOps Tooling

Tooling Type	Example
Git code repository	Git
Git management tool	Bitbucket, GitHub , GitLab
Continuous integration tool	CircleCI, Jenkins, Jenkinsx, GitHub Actions
Continuous delivery tool	ArgoCD , FluxCD, Spinnaker
Container registry	Docker Hub , AWS ECR , GHCR
Infrastructure provisioning	AWS CloudFormation , Pulumi, Terraform, AWS SAM
Configuration manager	Ansible, Chef, Puppet, Helm charts (config specific to Kubernetes)
Container orchestration	Kubernetes , Nomad

Continuous Integration using GitOps

- Continuous integration (CI) is a software development practice in which all developers merge code changes in a central repository (Git).
- With CI, each code change (commit) triggers an automated build-and-test stage for the given repo and provides feedback to the developer(s) who made the change.
- The main difference between GitOps compared to traditional CI is that with GitOps, the CI pipeline also updates the application manifest with the new image version after the build and test stages have been completed successfully [“Deploy” Job in CI pipeline, after the “Build” and “Test” Jobs]

GitOps CI



Continuous Delivery using GitOps

- Continuous delivery (CD) is the practice of automating the entire software release process.
- CD includes infrastructure provisioning in addition to deployment.
- What makes GitOps CD different from traditional CD is using a GitOps operator to monitor the manifest changes and orchestrate the deployment.
- As long as the CI build is complete and the manifest is updated, the GitOps operator takes care of the eventual deployment.

GitOps CD (Continuous Delivery)

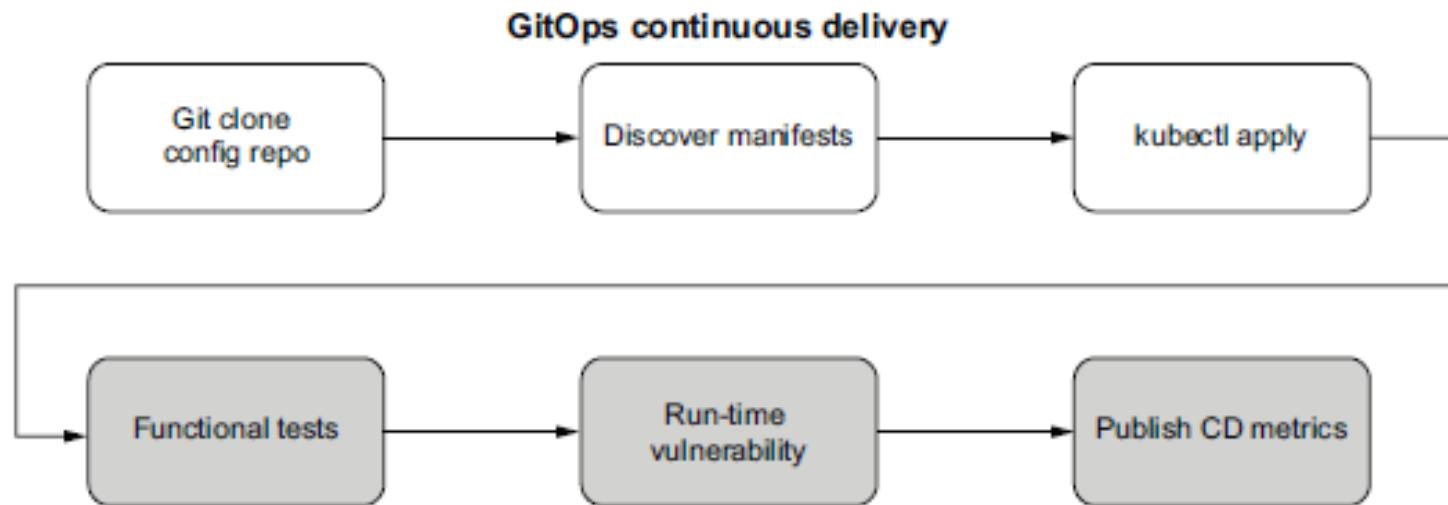


Figure 4.3 These are the stages in a GitOps CD pipeline. White boxes are from the GitOps CI pipeline from figure 2.9, and gray boxes are the additional stages for building a complete CI pipeline.



GitOps CI – In Action

DEMO I

- Application -> RESTful Microservice application developed in Python using FastAPI framework

Tools

1. GitHub Action for Continuous Integration
2. Flake8 for Linting
 - Rules - <https://www.flake8rules.com/>
 - "pycodestyle" - Enforces Python style conventions
 - "pyflakes" - Analyzes source code for syntax errors, undefined names, unused imports etc.

Linting

- Linting is the process of analyzing code for potential errors, bugs, or stylistic inconsistencies without executing the code. Linters are automated tools that perform static analysis on source code to identify patterns that might be problematic, violate coding conventions, or lead to errors during runtime.

DEMO - II

- Application -> RESTful Microservice application developed in Python using FastAPI framework

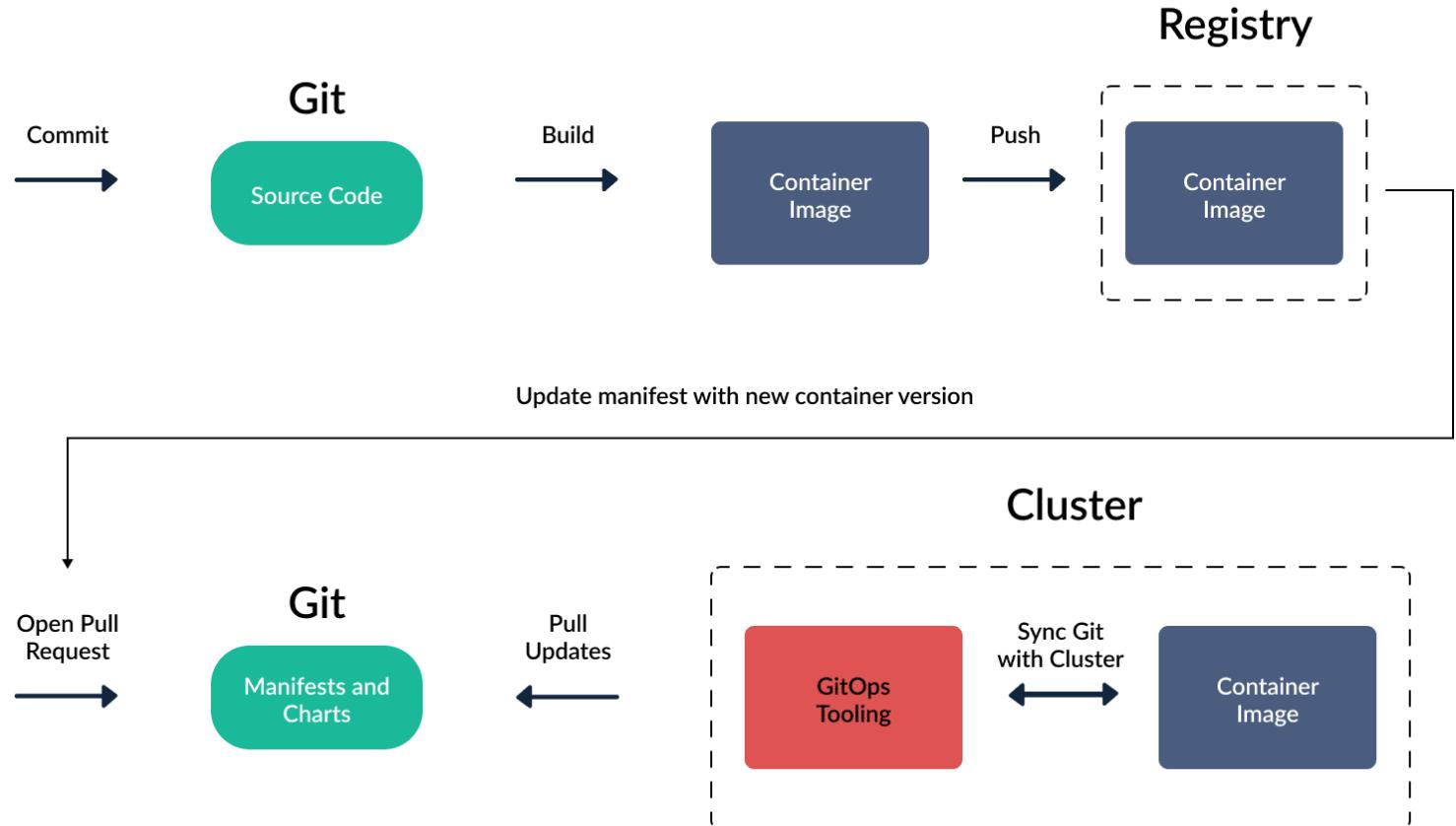
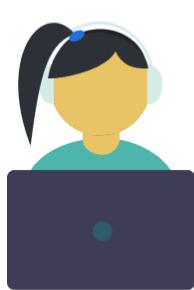
Tools

1. GitHub Action for Continuous Integration
2. Integration with SonarCloud for Static Code Analysis



Thank You!

GitOps [Additional Content]





BITS Pilani
Pilani Campus

DevOps for Cloud

Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani



CC ZG507 – DevOps for Cloud Lectures No. 7 and 8

Agenda

[Cover the pre-requisites for Continuous Delivery / Deployment]

1. Container based Architecture - Docker
2. Kubernetes - Architecture and Components
3. Demo of Docker and Kubernetes



Container based Architecture

- Docker

What are Containers

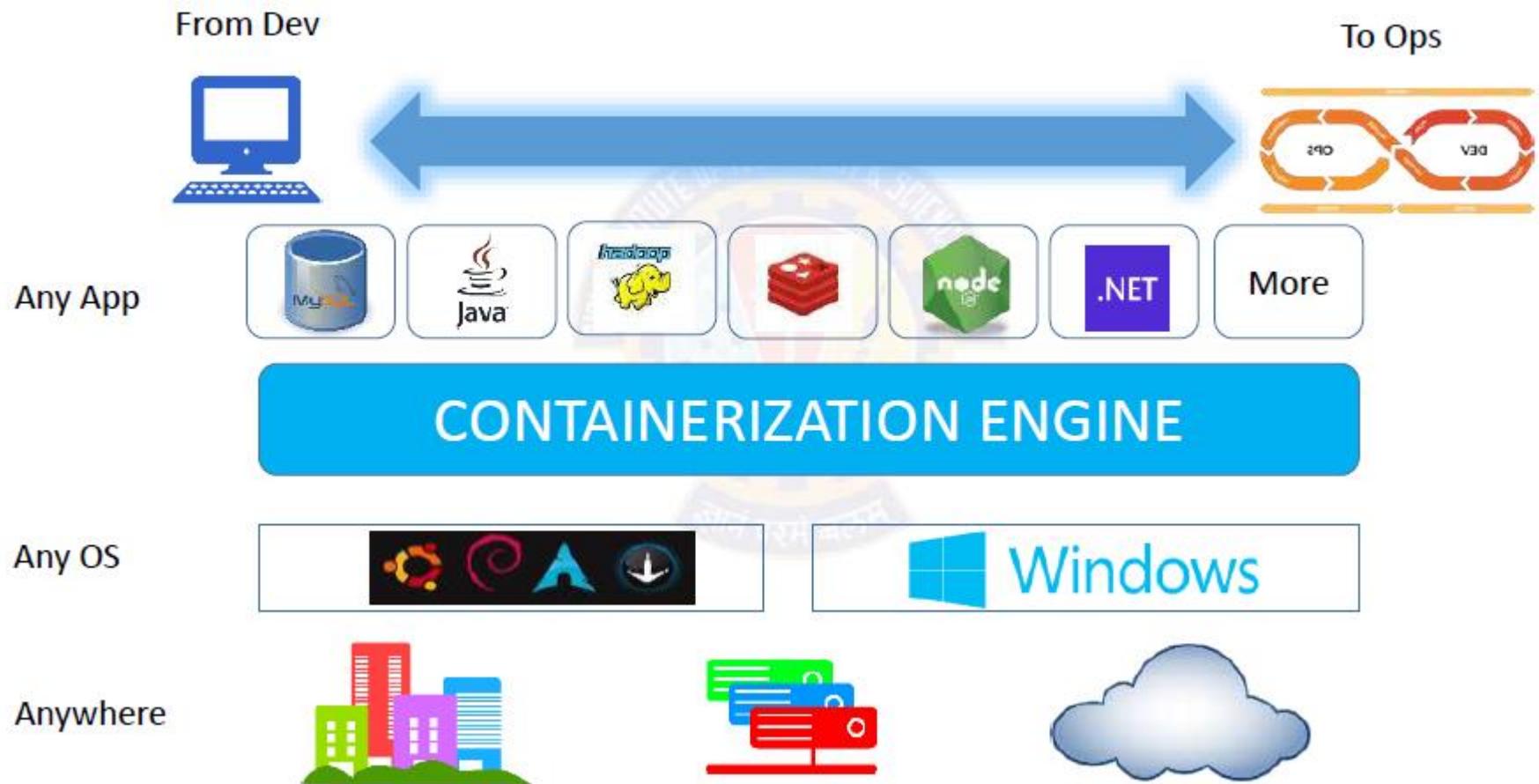
- A software container is a standardized package of software
- Everything needed for the software to run is inside the container
- The software code, runtime, system libraries, and settings are all inside a single container
- **Container based deployments is favoured for Microservices**

Cloud Native Development

1. Microservices Architectural Style
2. **Containers for Deployment**
3. Cloud
4. DevOps

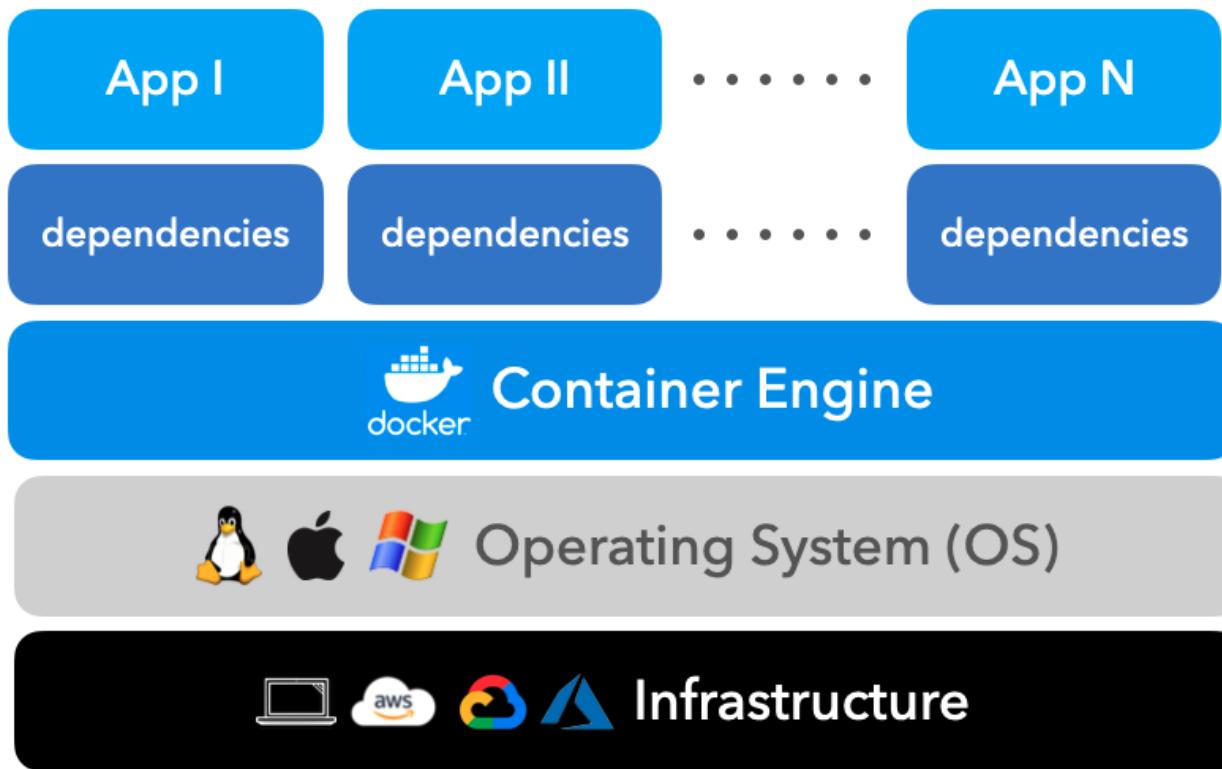


Containers – Build, Ship, Run Any App Anywhere

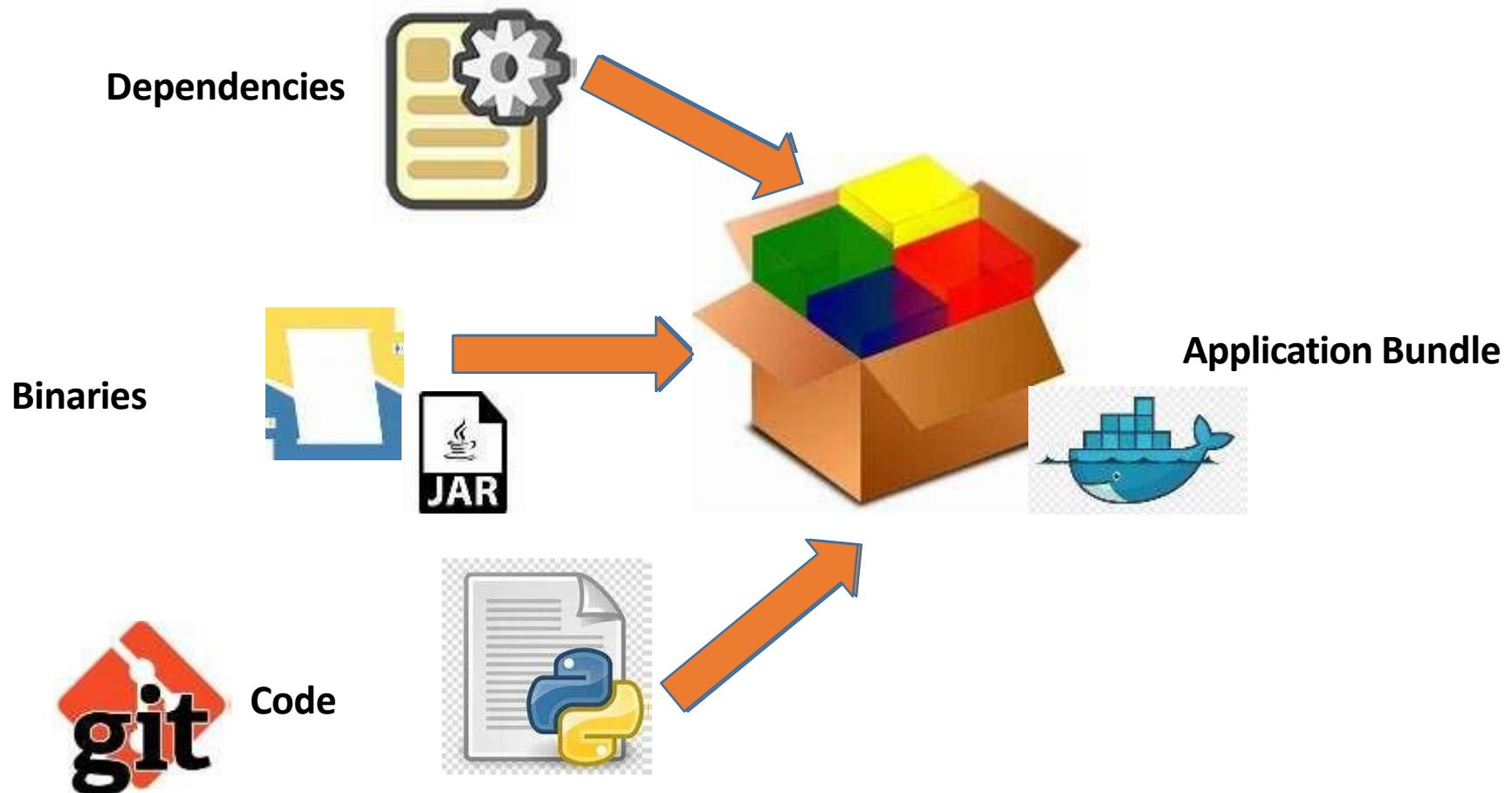


Docker

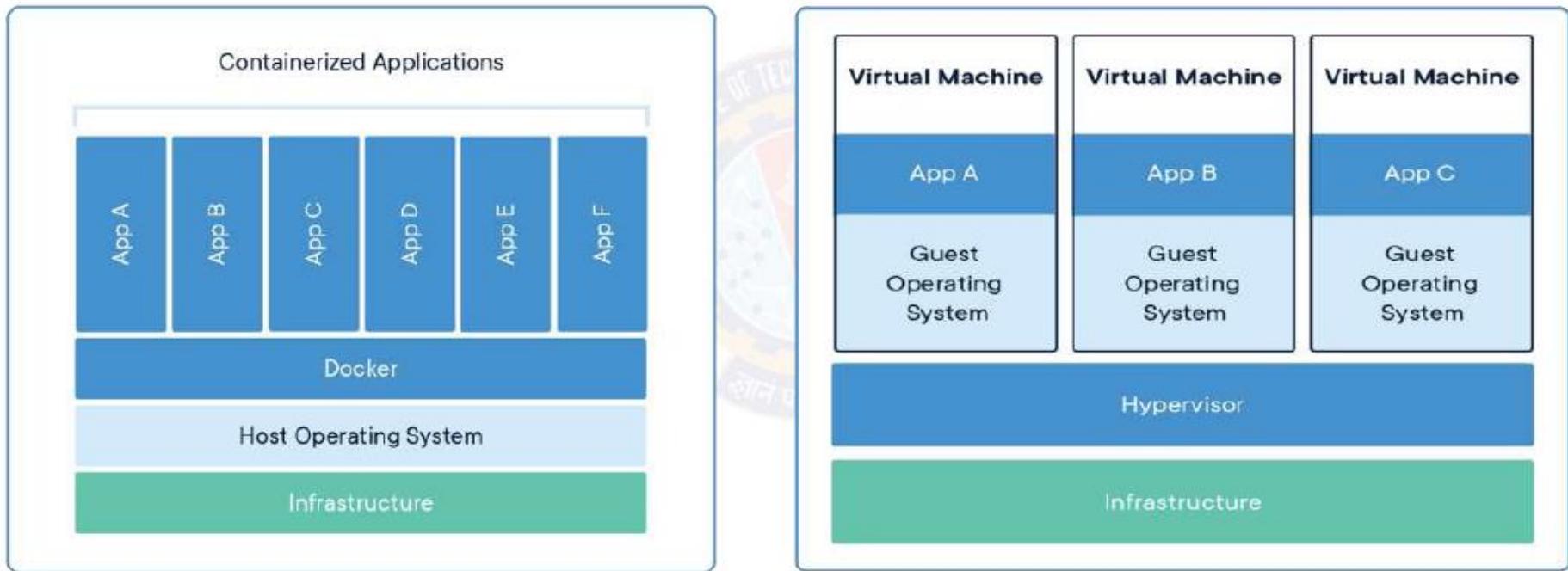
- Is a popular Container technology
- It is lightweight, portable and self-sufficient container



Docker - Container to Package Software



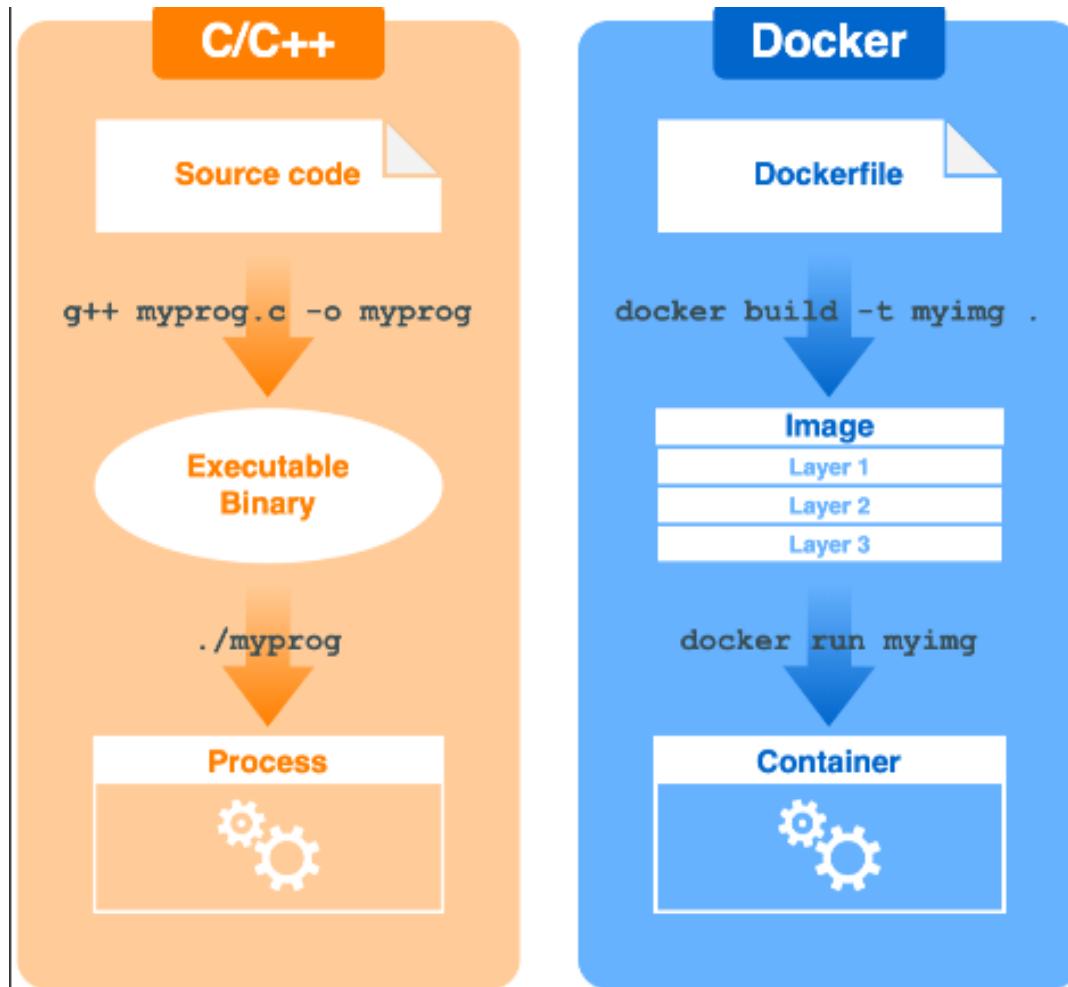
Containers vs Virtual Machines



Basic Definitions

- Docker - It is a way to package Software so that it runs on any hardware (OS & hardware).
- Requires knowledge of three things - *dockerfile, images and containers.*
 - a. Docker File - Blueprint for building a Docker Image. **Set of Instructions**
 - b. Docker Image - Template for running Docker Containers. Docker image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, libraries, and system tools.
 - c. Docker Container - A docker container is a portable unit of software - that has the application, along with the associated dependency and configuration. It is running process of the image. May include Application (Web - Node.js) and Database (DB - MySQL). One image file can be made to spawn multiple processes across environments.

Docker



Step 1: Dockerfile

**Step 2: Build Image
[docker build]**

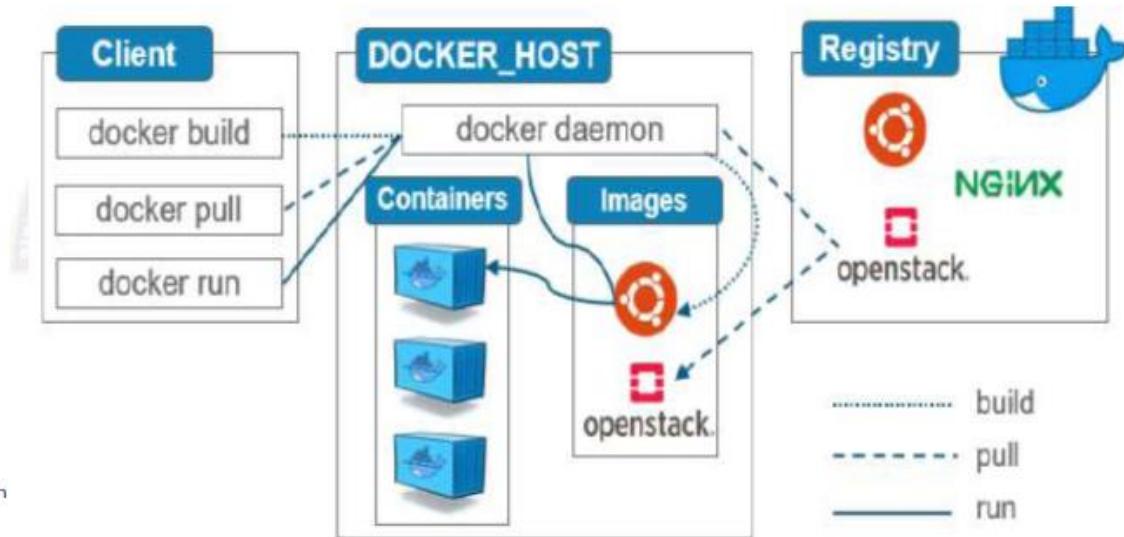
**Step 3: Run Image as Container
[docker run]**

Purpose of Docker

- Reproduce the environment in Development, Staging and Production.
- We can define the environment with the Docker File.
- People can use the dockerfile to create an image. Images can be uploaded to Cloud (public or private) repositories using “docker push” command and pulled using “docker pull” command from the repository.
- Kubernetes - Used to scale the dockers to an infinite load.

Docker Architecture

- Docker uses a client-server architecture.
- The Docker daemon
 - The Docker daemon (`dockerd`) listens for Docker API requests
 - Manages Docker objects such as images, containers, networks, and volumes.
 - Builds, runs, and distributes containers
- The Docker client
 - The Docker client talks to the Docker daemon
 - The Docker client and daemon can run on the same system
 - The Docker client can communicate with more than one daemon..
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface.





Docker Desktop

- Docker Desktop is a one-click-install application for your Mac, Linux, or Windows environment that enables you to build and share containerized applications and microservices.
- It provides a straightforward GUI (Graphical User Interface) that lets you manage your containers, applications, and images directly from your machine.

Docker Hub

- Docker Hub is the world's largest library and community for **container images**
- Contains Official Images for MongoDB, NodeJS, Redis, Ubuntu, Python, MySQL etc.
- URL - <https://hub.docker.com/>

Lab Sheet 3 – Module 6

Objectives

1. To demonstrate the steps to install and run Docker
 2. To build custom docker image in Docker Desktop
 3. To dockerize a RESTful Microservice application
 4. To push the docker image from Docker Desktop to Docker Hub
 5. To push the docker image from Docker Desktop to Amazon ECR (Elastic Container Registry)
-



Task 1

Docker Desktop Installation

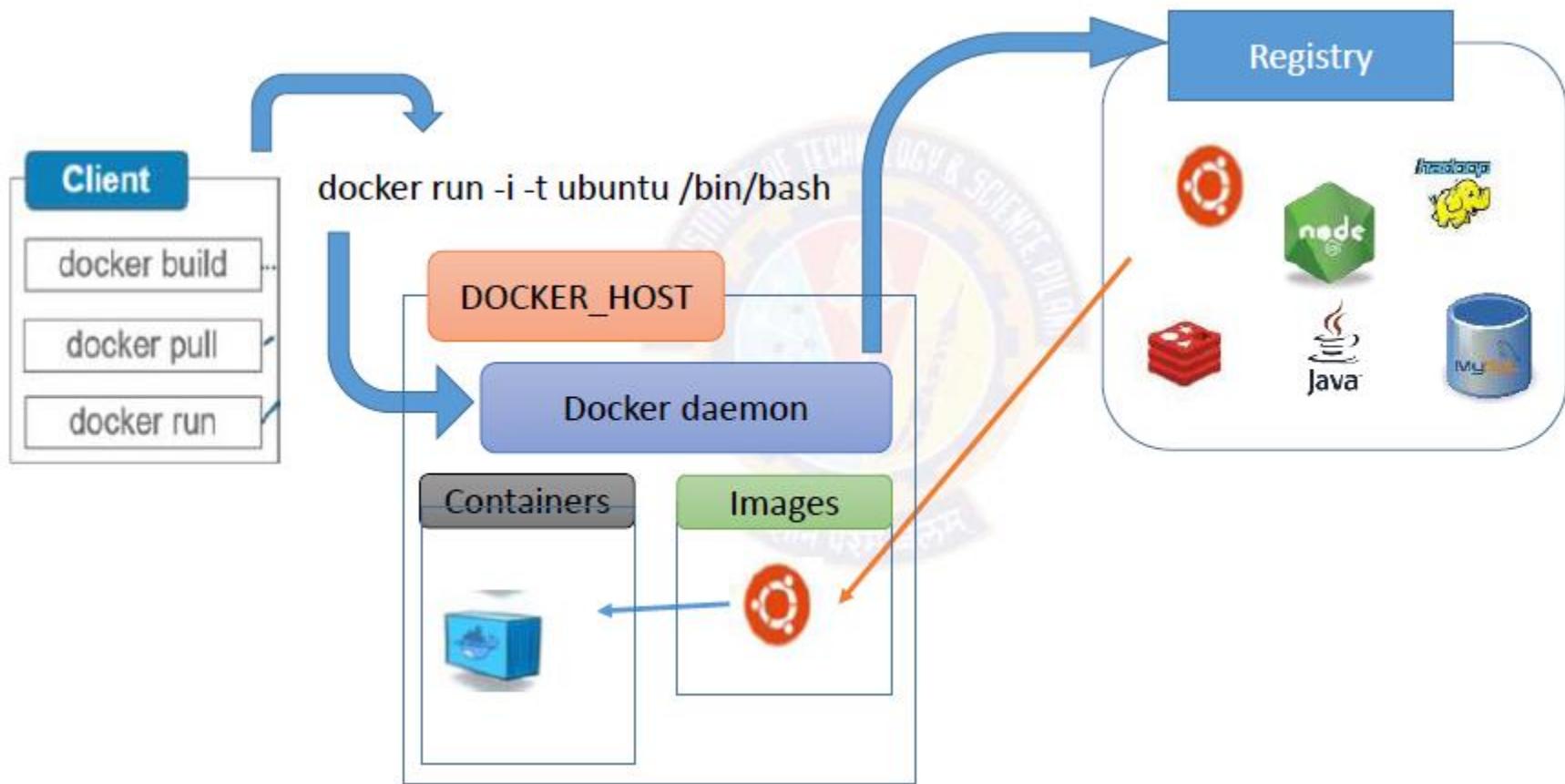
Task 2

Run base ‘Hello-World’ Image

Task 3

Demonstrate Interactive Terminal

Example for running a Docker Container



Dockerfile

Dockerfile is a text file, named *Dockerfile*, that includes specific keywords that dictate how to build a specific image.

Keywords are:

- **ADD** copies the files from a source on the host into the container's own filesystem at the set destination.
- **CMD** can be used for executing a specific command within the container.
- **ENTRYPOINT** sets a default application to be used every time a container is created with the image.
- **ENV** sets environment variables.
- **EXPOSE** associates a specific port to enable networking between the container and the outside world.
- **FROM** defines the base image used to start the build process.
- **MAINTAINER** defines a full name and email address of the image creator.
- **RUN** is the central executing directive for Dockerfiles.
- **USER** sets the UID (or username) which is to run the container.
- **VOLUME** is used to enable access from the container to a directory on the host machine.
- **WORKDIR** sets the path where the command, defined with CMD, is to be executed.
- **LABEL** allows you to add a label to your docker image.



Task 4

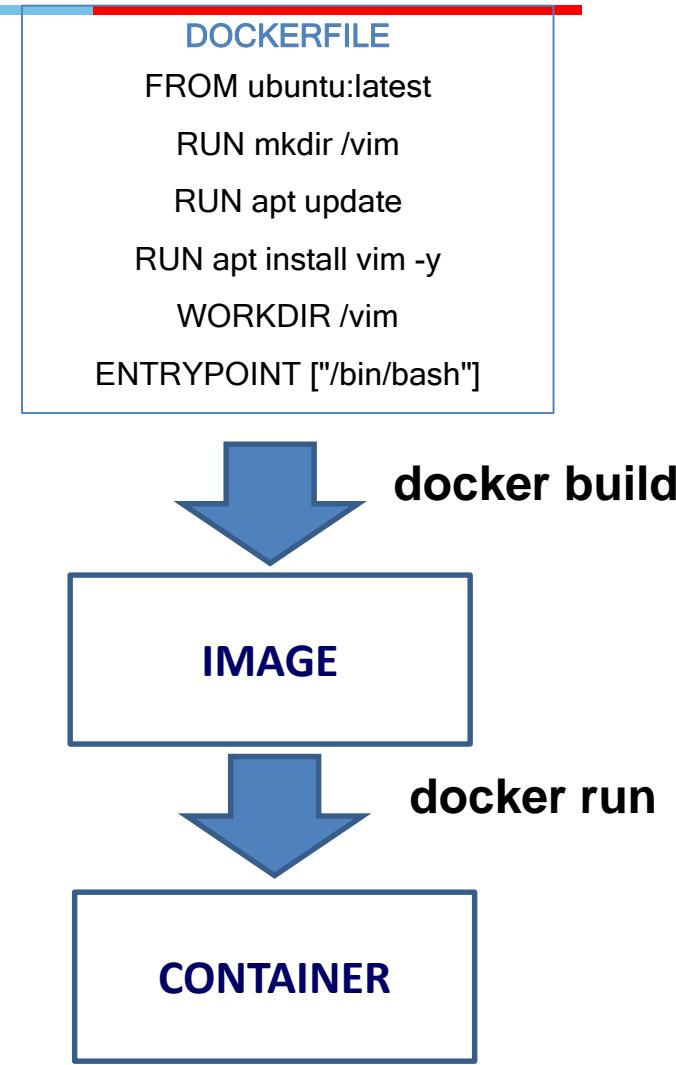
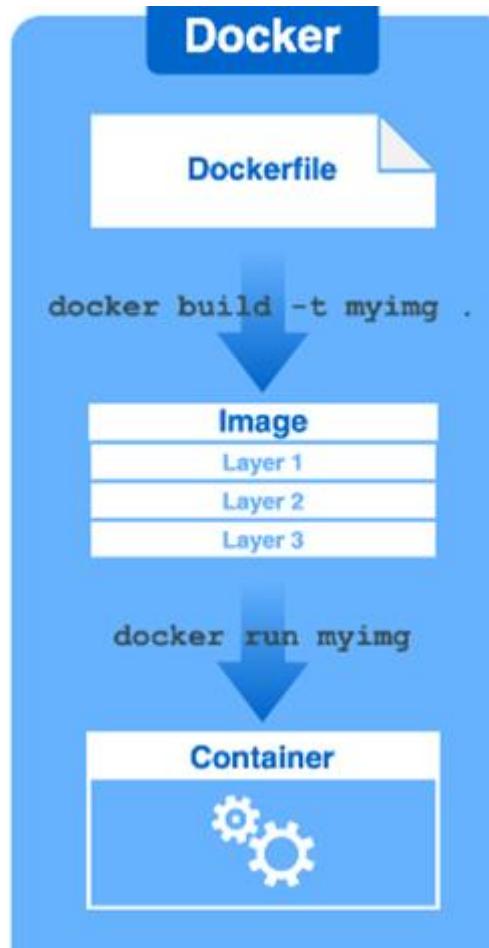
Build a Custom Docker Image

Dockerfile

```
# First Layer - from which Docker Image to start with
FROM ubuntu:latest

# Run indicates execute the commands inside the container
RUN mkdir /vim
RUN apt update
# -y is used to avoid prompt to install or not
RUN apt install vim -y
# WORKDIR informs which container working directory
WORKDIR /vim
# Which command to run in container - any executable as start point
ENTRYPOINT ["/bin/bash"]
```

Docker - Example



Task 5

Dockerize a RESTful Microservice Application

Dockerize a RESTful Microservice App

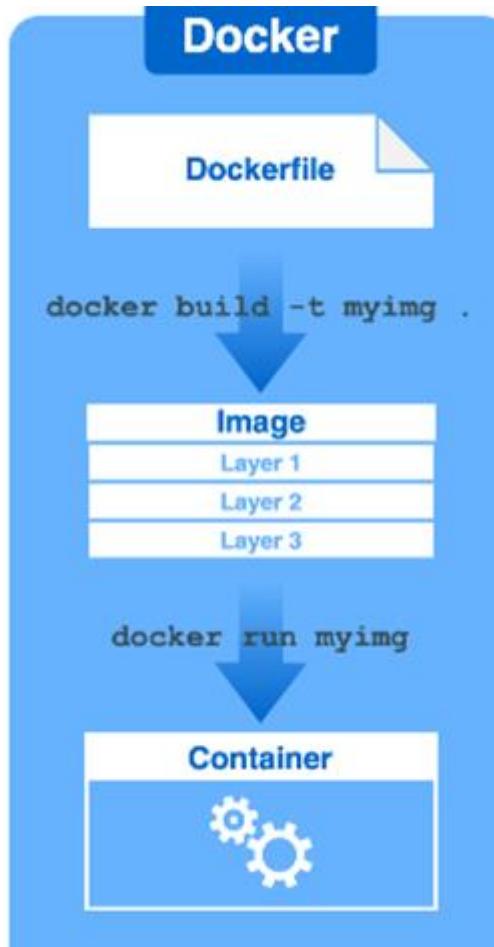
- Dockerize the RESTful Python Web Application
- Source Code (GitHub) URL - <https://github.com/shreyassureshrao/RestApp>

Main Components of the Code

1. dockerfile [contains instructions for creating docker image]
2. main.py [contains the FastAPI RESTful code]
3. user.json [contains the JSON file that contains the book object]
4. test_main.py [contains the unit tests]
5. requirements.txt [required dependencies to be installed for Python code execution]
6. “Steps to run code.txt” [Detailed steps to create the RESTful application and dockerize the application is mentioned in this file]

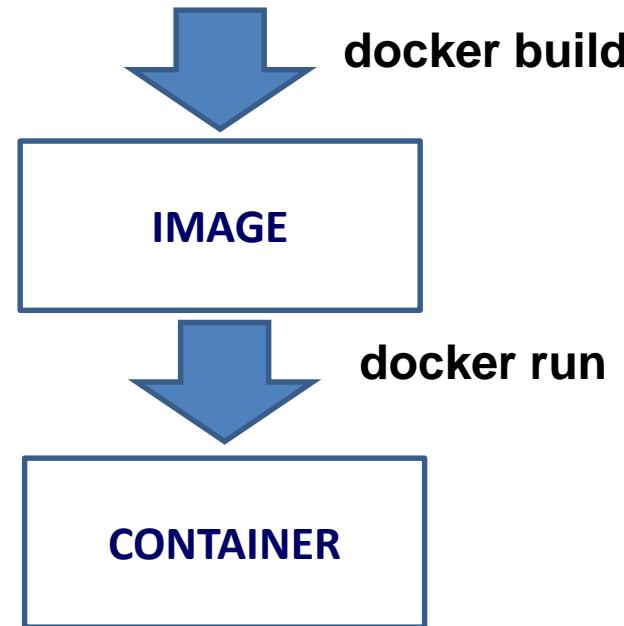
Note: If you are using Visual Studio Code, need to install ‘Docker’ as extension

Docker - Example



DOCKERFILE

```
FROM python:3.9
WORKDIR /code
COPY ./requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
COPY ./app /code/app
CMD ["unicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
```



Docker - Example

CODE

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import Optional
import json

app = FastAPI()

class User(BaseModel):
    id: Optional[int] = None
    name: str
    email: str
    address: str
    mobile: int

import os
here = os.path.dirname(os.path.abspath(__file__))
filename = os.path.join(here, 'user.json')

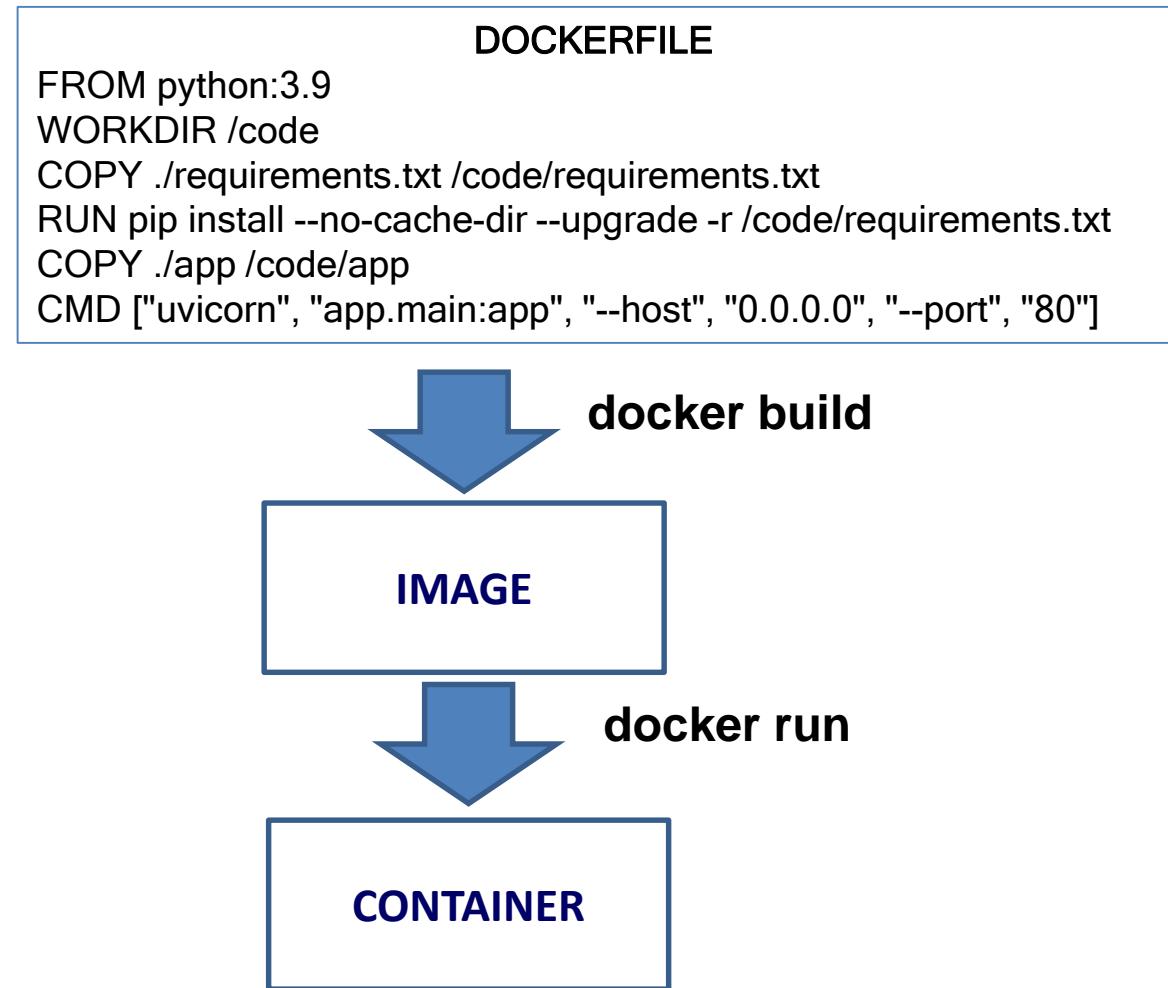
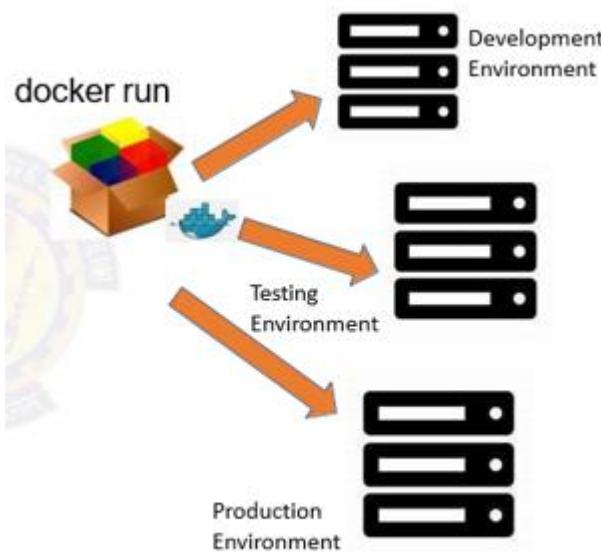
with open(filename, 'r') as f:
    user = json.load(f)['user']

@app.get('/')
def getAllUsers():
    return user
```

DEPENDENCY

```
fastapi>=0.68.0,<0.69.0
pydantic>=1.8.0,<2.0.0
uvicorn>=0.15.0,<0.16.0
```

Docker - Demo



Task 6

Push Image to Docker Hub

Task 7

Push Image to Amazon ECR



KUBERNETES



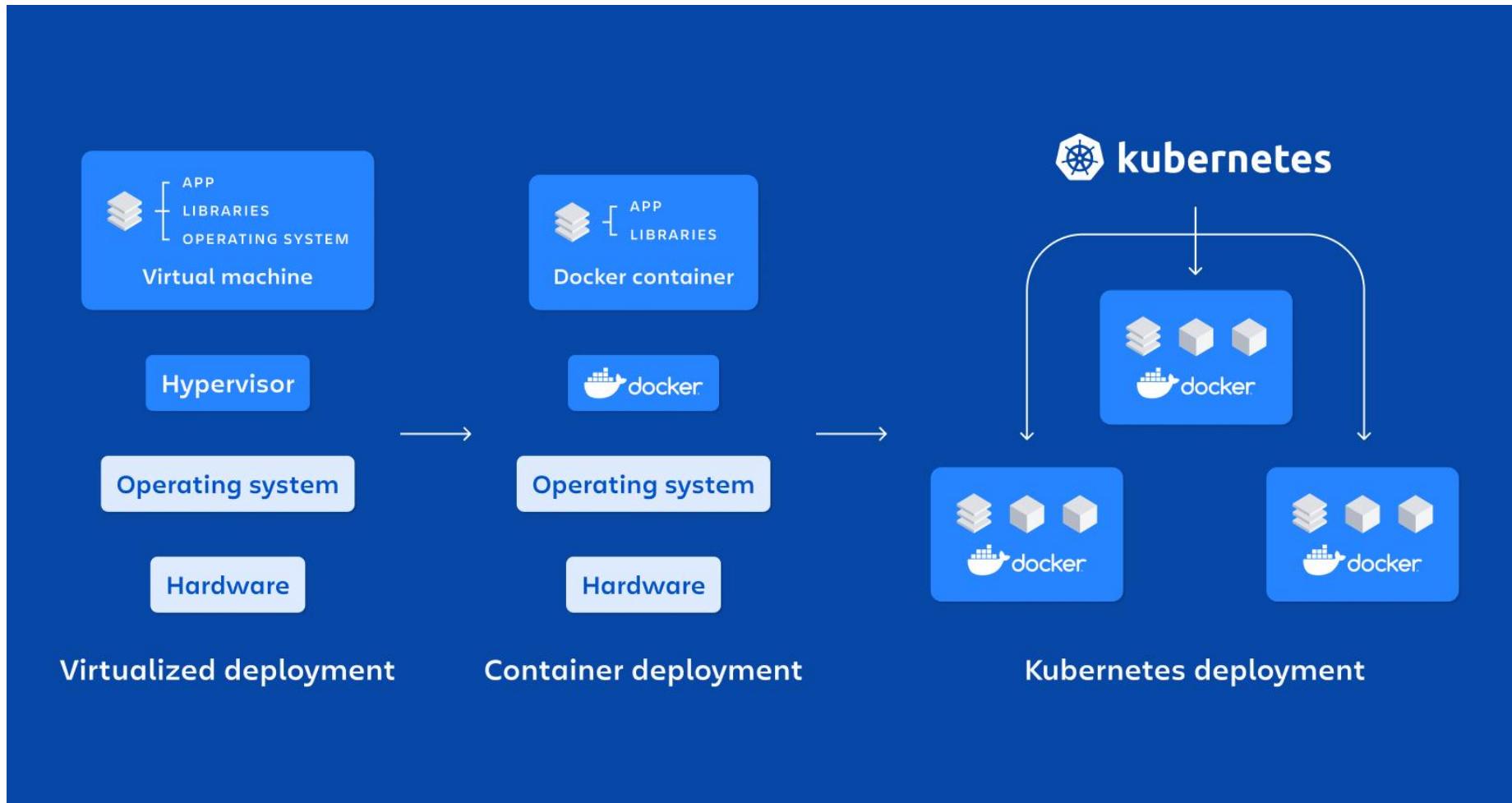
Kubernetes / K8s

- Kubernetes is a *container orchestration tool* that manages applications/ services available on a container platform like Docker
- Developed by Google labs and later donated to CNCF (Cloud Native Computing Foundation)
- Open source
- Written in Golang

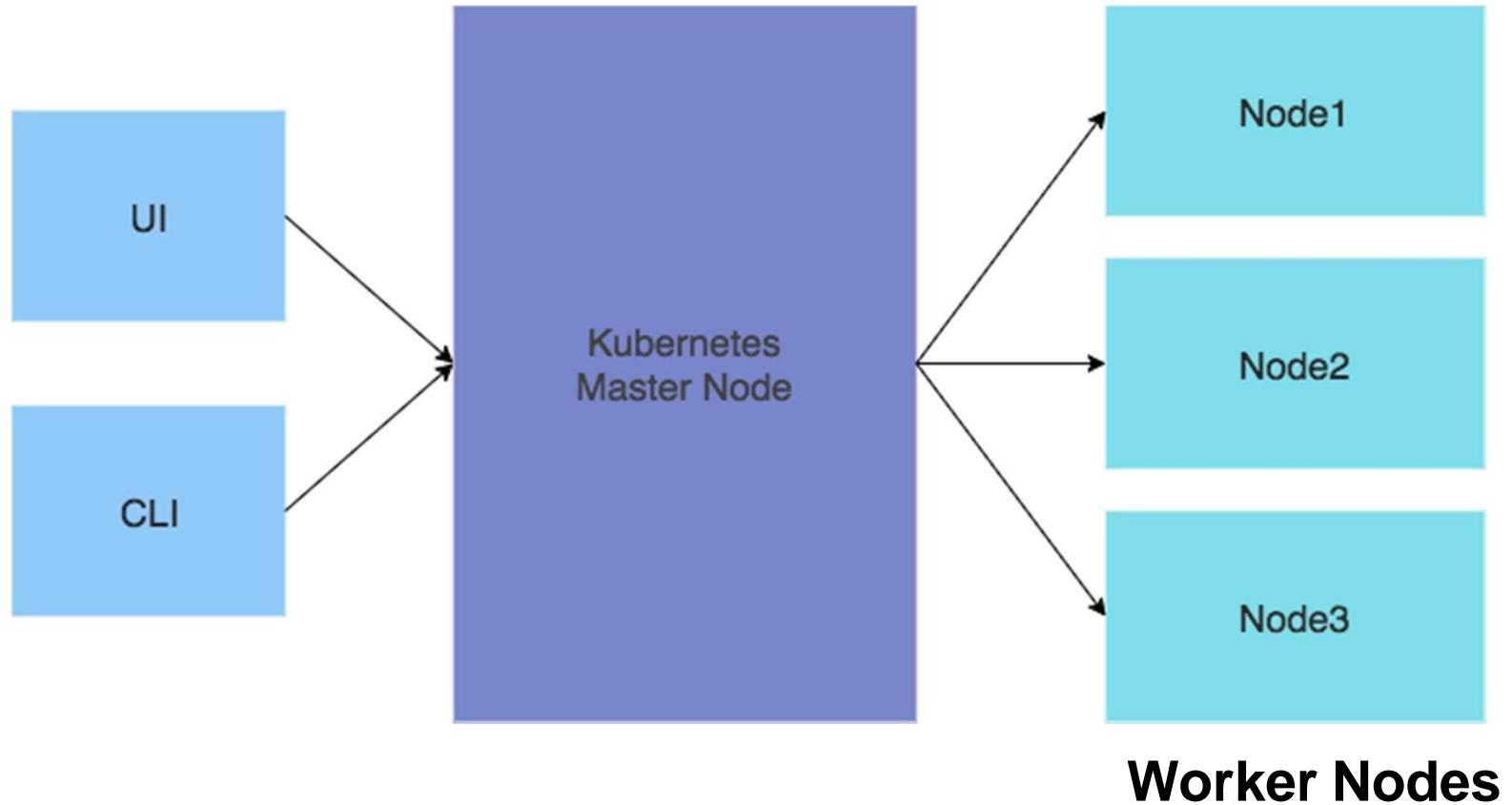


kubernetes

Kubernetes / K8s

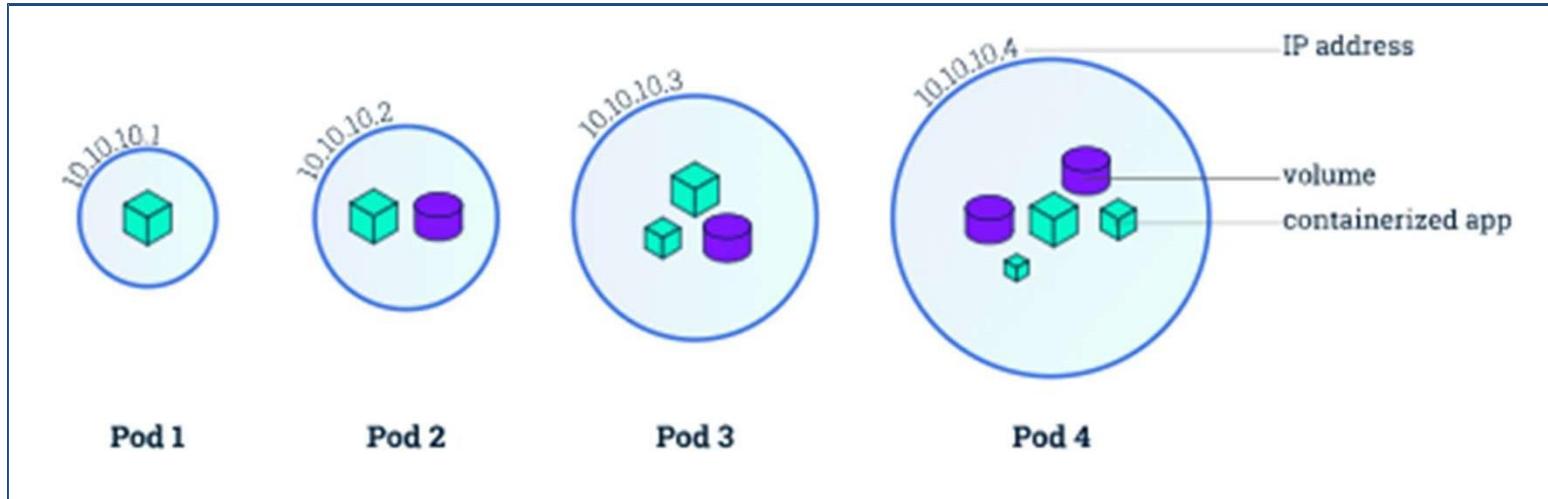


Kubernetes Components



Worker Nodes

Kubernetes Components

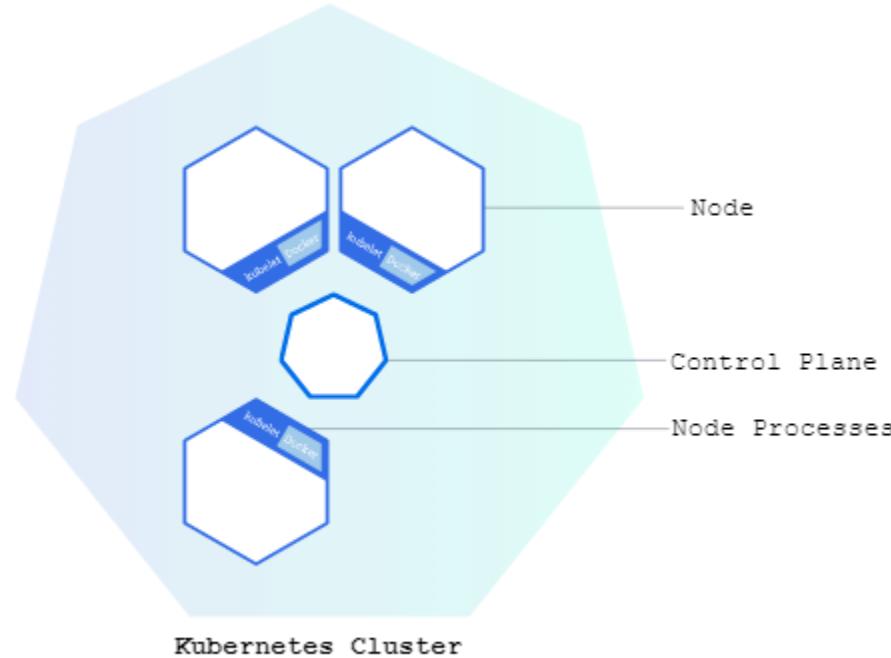


Node

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.

A *Pod* is a group of one or more containers, with shared storage and network resources

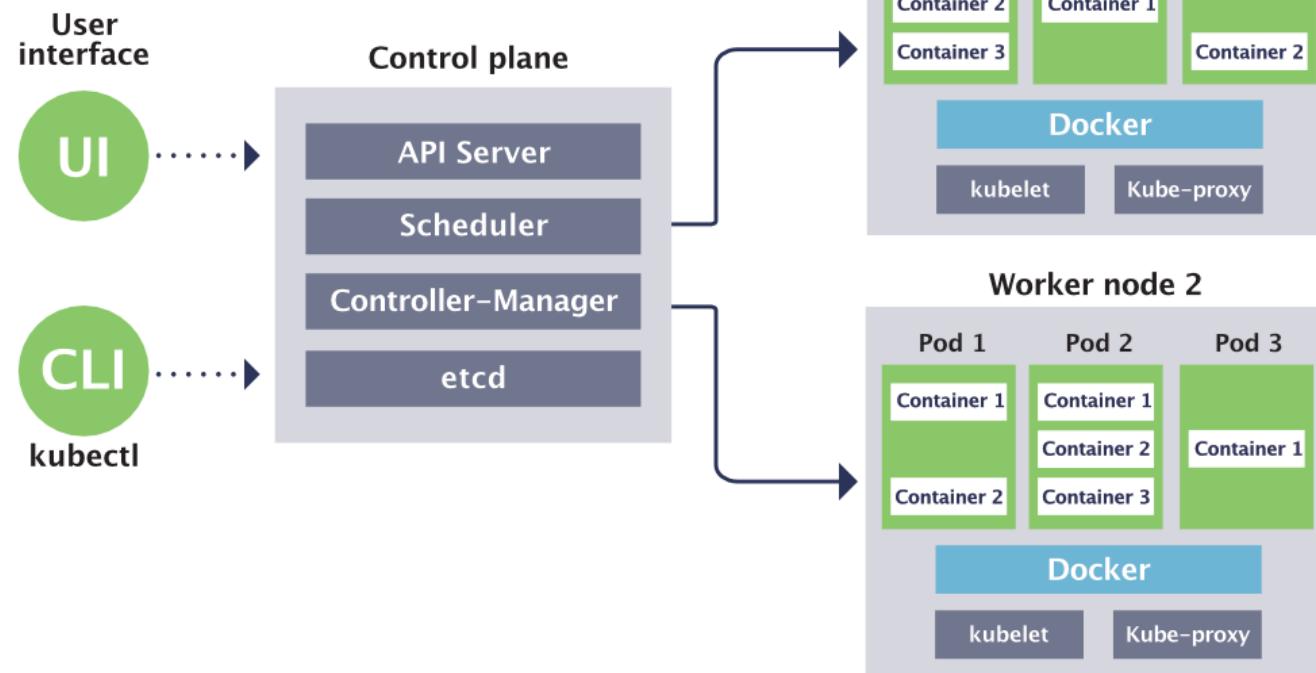
Kubernetes Cluster



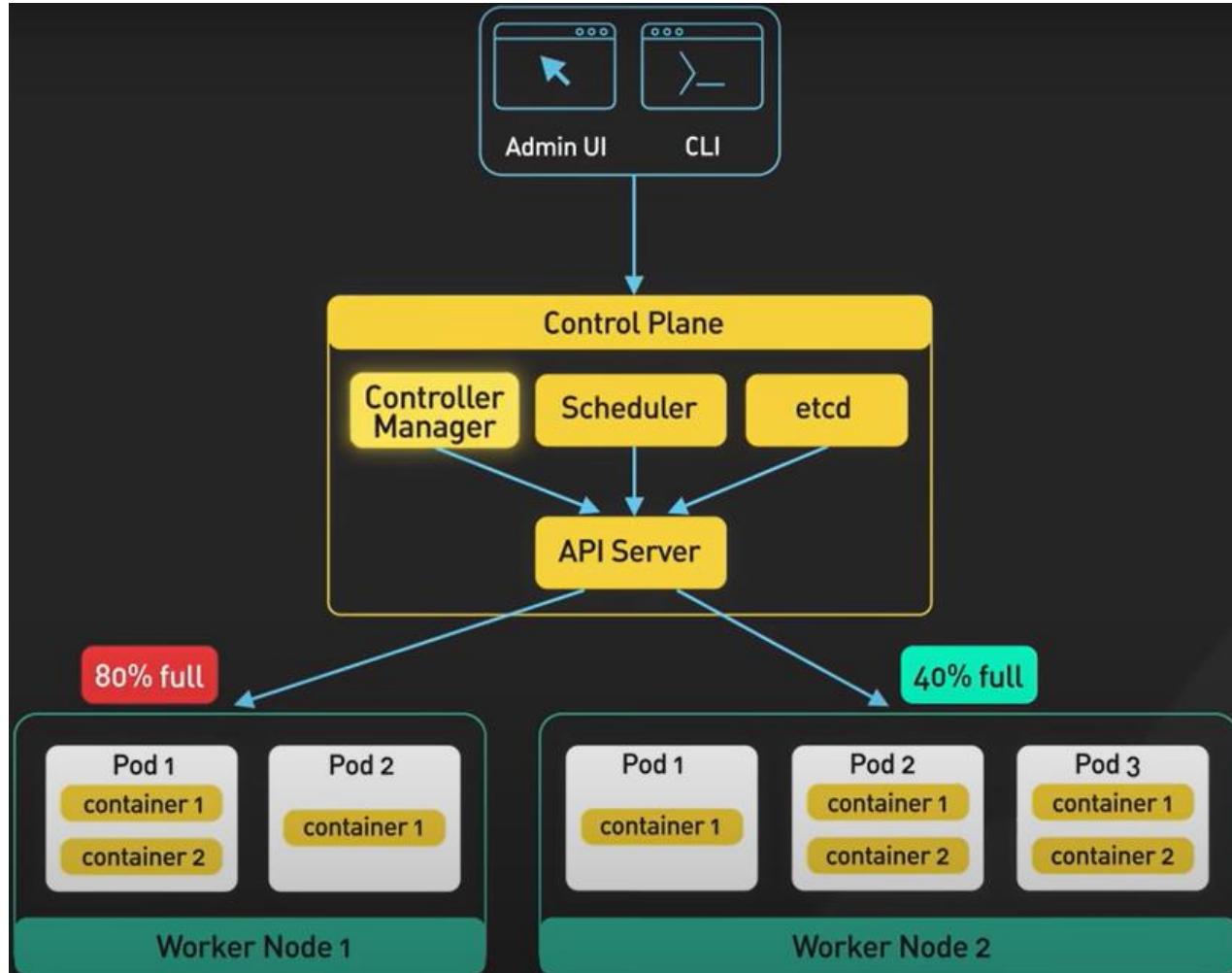
Control Plane nodes are also called as 'Master Nodes'

Kubernetes Architecture

Kubernetes architecture



Scheduler – Schedules Pods on Cluster nodes



Kubernetes Components

- **Master Node:** The master node (Control Plane Node) is responsible for managing the entire cluster. It contains various components like the API server, etcd, controller manager, and scheduler.
- **Worker Nodes:** These are the nodes where containers run. They are responsible for running the application workload and communicating with the master node.
- **Pods:** Pods are the smallest deployable units in Kubernetes. They are used to encapsulate one or more containers and share storage and network resources.
- **Services:** Services are used to expose a group of pods to the network. They provide a stable IP address and DNS name that other services and pods can use to communicate with the group of pods.
- **Controllers:** Controllers are used to manage the lifecycle of pods and services. They ensure that the desired number of replicas of a pod or service are running and will automatically create or delete pods as necessary.

Master Node Components

- **API Server:** The API server is the central control point for the Kubernetes cluster. It exposes the Kubernetes API, which is used by other components to communicate with the cluster. It stores the state of the cluster in etcd.
 - **etcd:** etcd is a distributed key-value store that stores the configuration data of the Kubernetes cluster. It is the source of truth for the current state of the cluster and is used by the API server, controller manager, and scheduler.
 - **Controller Manager:** The controller manager is responsible for running the various controllers that regulate the state of the cluster. For example, the ReplicaSet controller ensures that the desired number of pod replicas are running, and the Deployment controller manages rolling updates of deployments.
 - **Scheduler:** The scheduler is responsible for assigning pods to nodes in the cluster. It takes into account factors such as resource requirements, node availability, and user-defined policies.
-

How Deployment works in Kubernetes

- **Kubectl** - The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.
- **Kubelet** - Manages the state of the node and ensures that containers are running and healthy. It also monitors the state of the containers and restarts them if they fail.
- **Kube-proxy** - Manages network connectivity to and from the Pods

<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

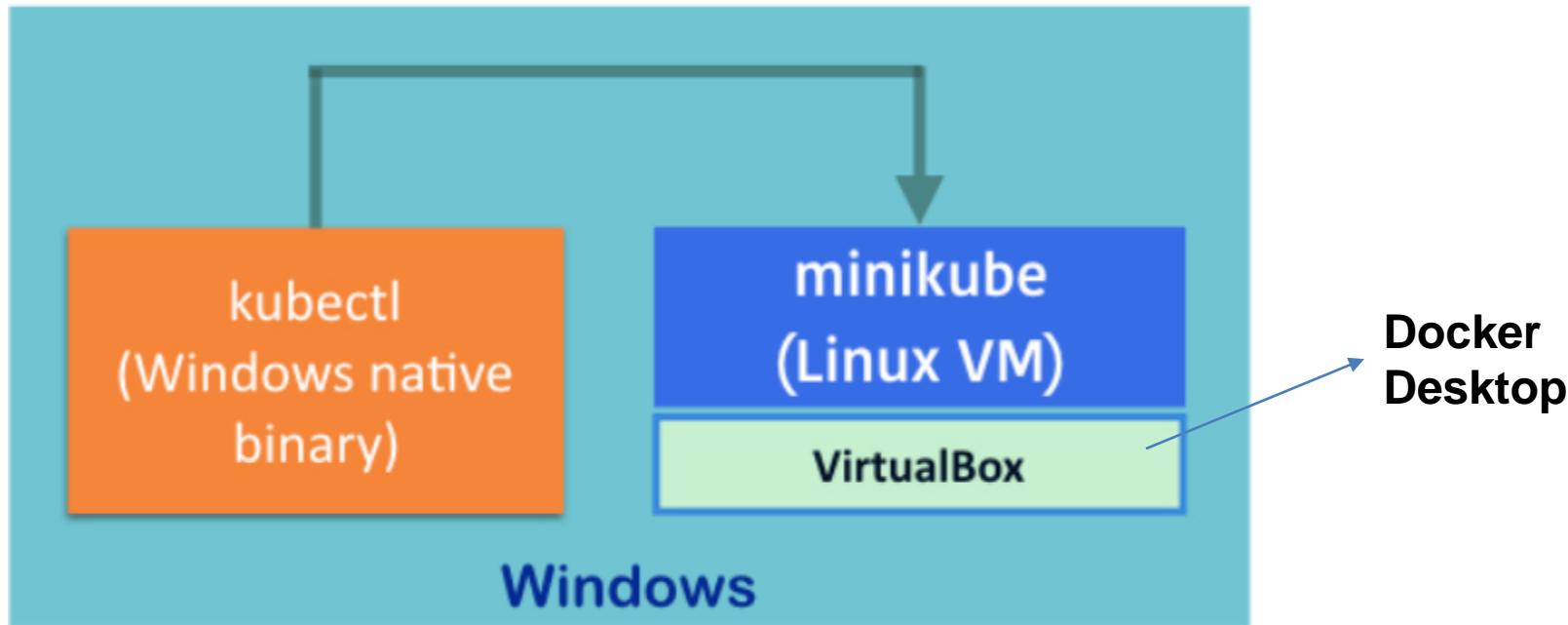
Minikube

- Is a **Single Node Kubernetes cluster**, which we can create on local machine
- Helps in development and experimentation
- Can be run inside of a Container (Docker) or Virtual Machine



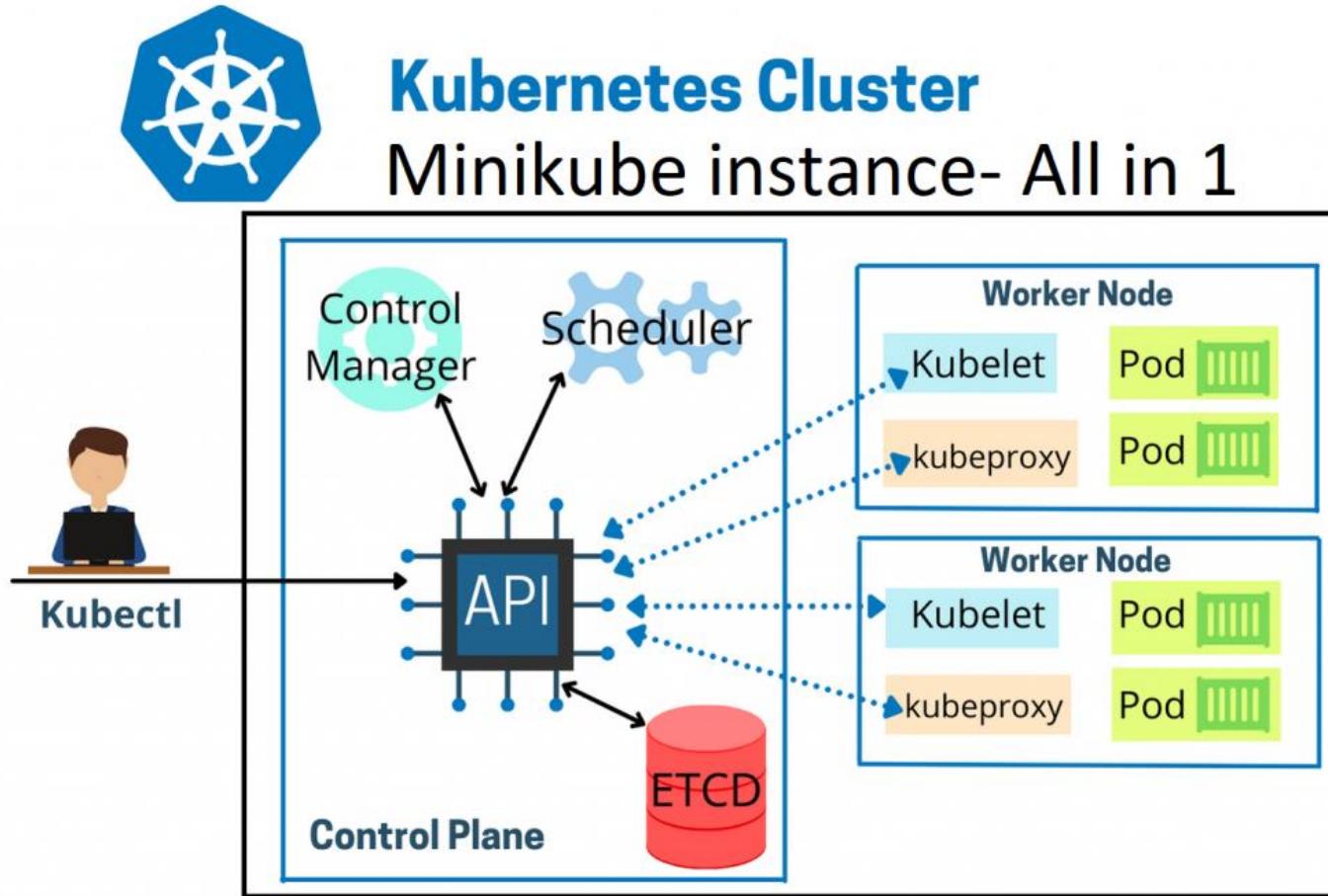
<https://minikube.sigs.k8s.io/docs/>

Minikube Architecture



Container or virtual machine manager, such as: Docker, QEMU, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

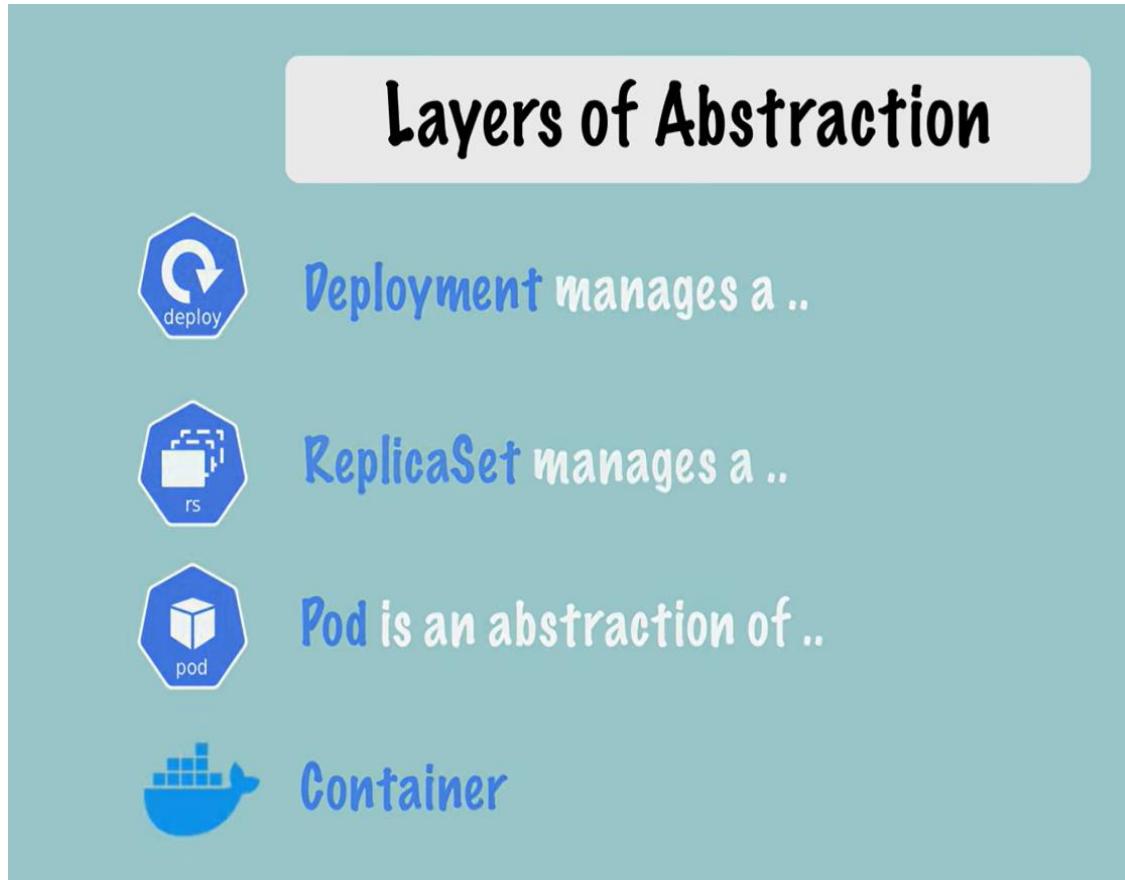
Minikube Architecture



DEMO

Refer file - “Steps to install and run Minikube”

Layers of Abstraction in Kubernetes



Basic YAML – Deployment File

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-cont
      image: nginx:latest
```

To apply this YAML file to your Kubernetes cluster, you can use the `kubectl apply` command:

```
kubectl apply -f pod-definition.yaml
```

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-cont
          image: nginx:latest
```

specify the Pod's metadata and containers.

to manage and select resources, such as in Services or ReplicaSets.

YAML Configuration File in Kubernetes

Three parts of Configuration File:

1. Metadata [Mentioned in config file]
2. Specification [Mentioned in config file]
3. Status [Not present in Config file]
 - Automatically generated and added by Kubernetes
 - Matches Desired state vs Actual State
 - [Ex: 2 replicas mentioned in config file (desired) vs 1 replica running (actual)] -> Self-Healing feature
 - 'etcd' component holds the current state of kubernetes cluster, which is displayed by Status

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-cont
          image: nginx:latest
```

Kubernetes Dashboard

View the created application in Minikube dashboard

Command: **minikube dashboard**

See ‘Deployments’, ‘Pods’, ‘Replica Sets’, ‘services’, and view their metadata



Thank You!