NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Software Deployment Requirements

Cloud Infrastructure Engineering

Nanyang Technological University & Skills Union - 2022/2023

Course Content

- Quick Check-In
- Dive into the basics of SDLC, CI/CD, Containerization Tools, Deployment Types, and Deployment Methodologies.
- Explore the importance of CI/CD and Deployments

Time	What	How or Why
7:15pm - 7:30pm	Part 1 - Presentation	SDLC
7:30pm - 7:45pm	Part 2 - Presentation	Agile
7:45pm - 8:00pm	Part 3 - Activity	Activity on SDLC
8:00pm - 8:10pm	Break	
8:10pm - 8:25pm	Part 4 - Presentation	CI/CD
8:25pm - 8:40pm	Part 5 - Presentation	Containerization
8:40pm - 9:00pm	Part 6 - Activity	Activity on Agile
9:00pm - 10:00pm	Summary & Assignments	

Recap

- Security Groups
  - Used to control inbound and outbound traffic to AWS resources e.g. EC2
- Why Security Groups?
  - First layer of traffic filtering on resource level
- Best Practices
  - Limit traffic to only permitted IP ranges
  - Delete unused security groups

Self Study Check-In

Q1) What is the Agile Methodology?

Q2) What is the Waterfall Methodology?

Lesson Overview

General Overview

IT software deployment is a necessity in any organization’s IT environment.

Software deployment can make or break the transitions between software deployments, which significantly affect business operations.

That is why it’s essential to have a solid software deployment process in place.

SDLC

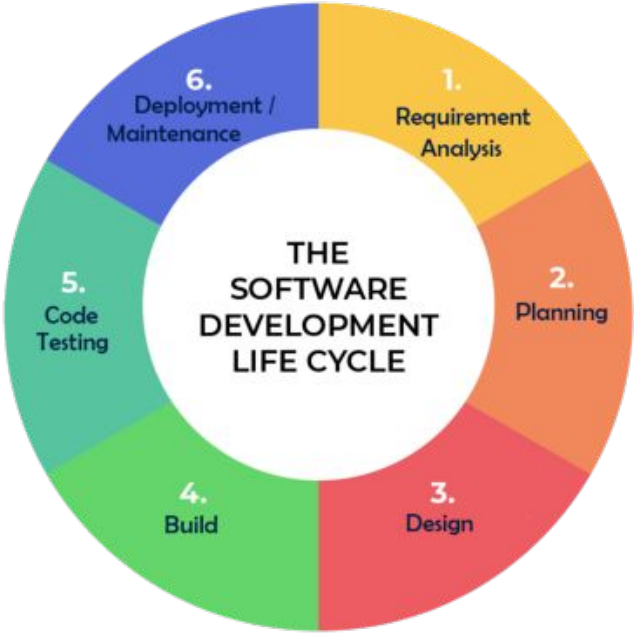
What Is SDLC?

SDLC - **Software Development Life Cycle**

Software deployment involves **all the activities** required to get a **software system or application ready for use on a device or a server**. Using software deployment software will help to ensure that all applications in your organization's environment operate smoothly.

The SDLC refers to a methodology with clearly defined processes for creating high-quality software.

6 Stages of SDLC



Requirement Analysis

“What are the current problems?”

This stage of the SDLC means getting input from **all stakeholders,** including customers, salespeople, industry experts, and programmers.

Learn the **strengths and weaknesses** of the current system with improvement as the goal.

Planning

“What do we want?”

In this stage of the SDLC, the team **determines the cost and resources required for implementing** the analyzed requirements.

It also details the risks involved and provides sub-plans for softening those risks.

Design

“How will we get what we want?”

This phase of the SDLC starts by **turning the software specifications into a design plan** called the Design Specification.

Other times, you may see terms like Business Requirements Documents.

Build

“Let’s create what we want.”

At this stage, the **actual development starts.**

It’s important that every developer sticks to the agreed blueprint.

Code Test

“Did we get what we want?”

In this stage, we **test for defects and deficiencies.**

We fix those issues until the product meets the original specifications.

Deployment & Maintenance

“Let’s start using what we got.”

At this stage, the goal is to **deploy the software to the production environment** so users can start using the product.

The plan rarely turns out perfect when it meets reality.

Further, as conditions in the real world change, we need to update and advance the software to match.

Deployment Methodologies

- Basic Deployment
- Rolling Update/ Deployment
- Blue-Green Deployment
- Canary Deployment

Credits:

https://www.harness.io/blog/blue-green-canary-deployment-strategies

Basic Deployment

In a basic deployment, **all nodes/servers** within a target environment are **updated at the same time** with a **new service or artifact version.**

Because of this, basic deployments are not outage-proof and they slow down rollback processes or strategies. Of all the deployment strategies shared, it is the riskiest.

Basic Deployment

Basic Deployment

Pros:

The benefits of this strategy are that it is **simple, fast, and cheap**.

Use this strategy if 1) your application service is **not business, mission, or revenue-critical**, or 2) your deployment is to a lower environment, during off-hours, or **with a service that is not in use**.

Basic Deployment

Cons:

Of all the deployment strategies shared, it is the **riskiest** and **does not fall into best practices.**

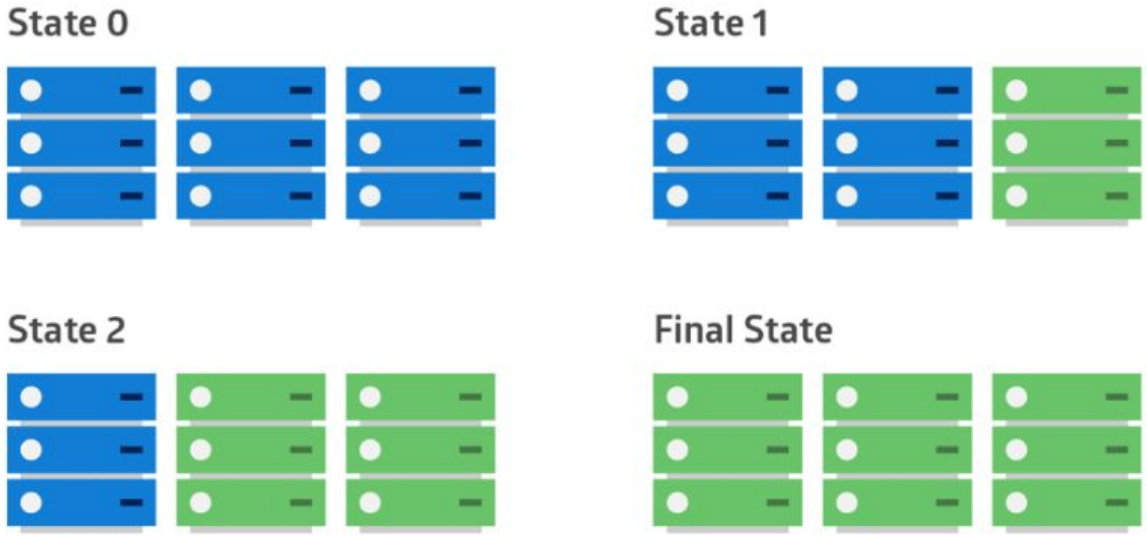
Basic deployments are **not outage-proof** and do not provide for easy rollbacks.

Rolling Deployment

A rolling deployment is a deployment strategy that updates running instances of an application with the new release version.

All nodes/servers in a target environment are **incrementally updated** with the service or artifact version in **integer N batches.**

Rolling Deployment



Rolling Deployment

Pros:

The benefits of a rolling deployment are that it is relatively simple to roll back, **less risky than a basic deployment**, and the implementation is **simple**.

Rolling Deployment

Cons:

- Since nodes are updated in batches, rolling deployments require services to **support both new and old versions of an artifact**.
- Verification of an application deployment at every incremental change also **makes this deployment slow.**

Blue-Green Deployment

Blue-green deployment starts by having the **original environment plus a duplicate environment**. This enables you to **preserve the old environment** while deploying the new application simultaneously.

Once the new application is deployed, make sure that everything runs properly. When you've determined that the **new environment is free of issues**, you can switch back to the new environment and then end the old environment.

Blue-Green Deployment

Blue-Green Deployment

Pros:

One of the benefits of the blue-green deployment is that it is **simple, fast, well-understood, and easy to implement**.

Rollback is also **straightforward**, because you can simply flip traffic back to the old environment in case of any issues.

Blue-green deployments are therefore not as risky compared to other deployment strategies.

Blue-Green Deployment

Cons:

Replicating a production environment can be complex and **expensive**, especially when working with microservices.

Quality assurance and user acceptance testing **may not identify all of the anomalies or regressions either,** and so shifting all user traffic at once can present risks.

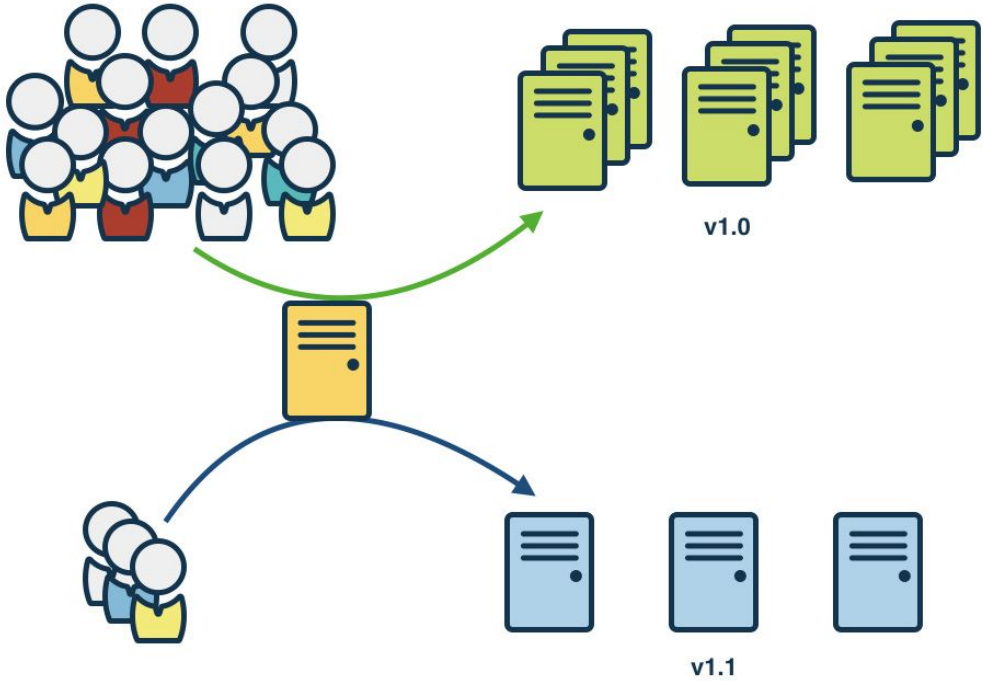
Canary Deployment

A canary deployment is a deployment strategy that **releases an application or service incrementally to a subset of users.**

All infrastructure in a target environment is updated in small phases (e.g: 2%, 25%, 75%, 100%).

A canary release is the **lowest risk-prone,** compared to all other deployment strategies, because of this control.

Canary Deployment



Canary Deployment

Pros:

Canary deployments allow organizations to **test in production with real users** and use cases and **compare different service versions** side by side.

It's **cheaper** than a blue-green deployment because it does not require two production environments.

And finally, it is **fast and safe to trigger a rollback** to a previous version of an application.

Canary Deployment

Cons:

Drawbacks to canary deployments involve **testing** in production and the implementations needed.

Scripting a canary release can be complex: manual verification or testing can take time, and the required monitoring and instrumentation for testing in production may involve additional research.

Break Time

Introduction To CI/CD

CI/CD

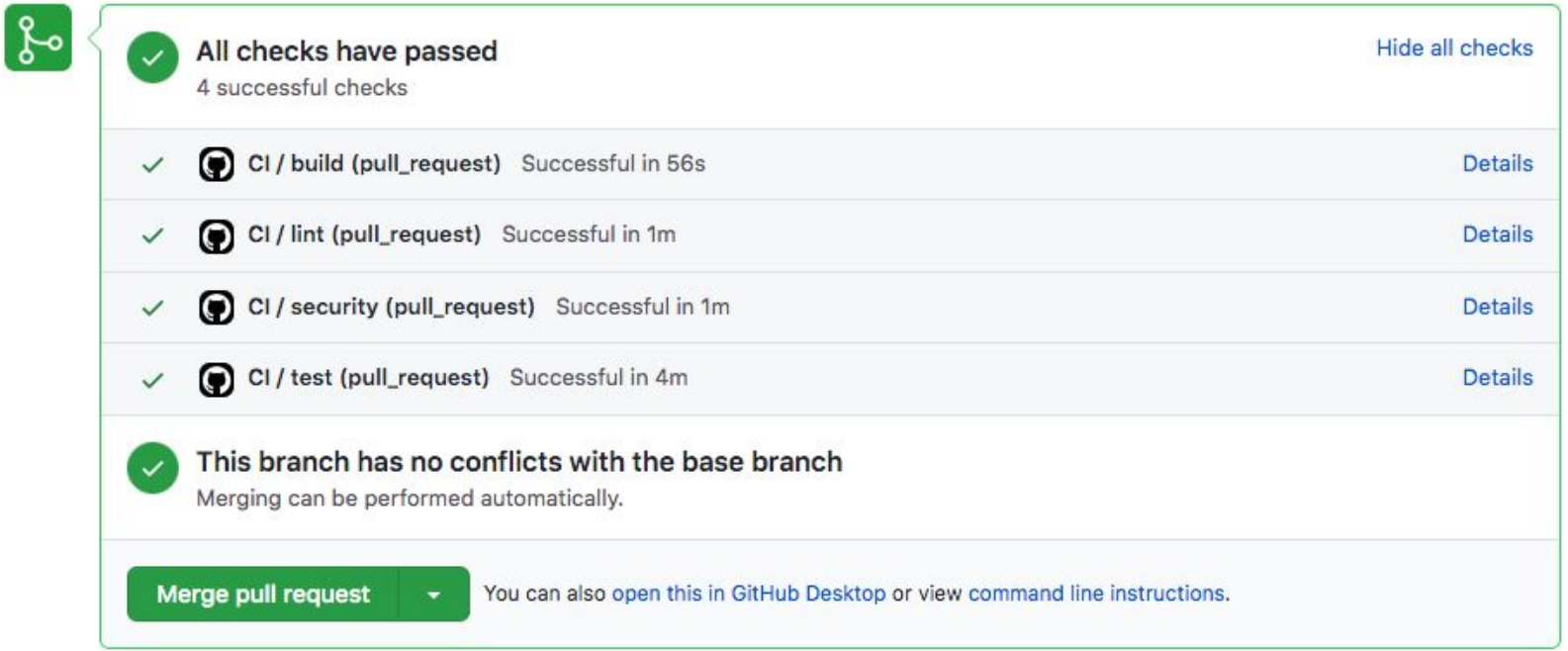
CI is a modern software development practice in which **incremental code changes are made frequently and reliably.**

Automated build-and-test steps triggered by CI ensure that code changes being merged into the repository are **reliable**.

The code is then delivered quickly and seamlessly as a part of the CD process.

In the software world, the CI/CD pipeline refers to the automation that enables incremental code changes from developers' desktops to be delivered quickly and reliably to production.

CI


All checks have passed
4 successful checks
Hide all checks
CI / build (pull_request) Successful in 56s
Details
CI / lint (pull_request) Successful in 1m
Details
CI / security (pull_request) Successful in 1m
Details
CI / test (pull_request) Successful in 4m
Details
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.

CI/CD

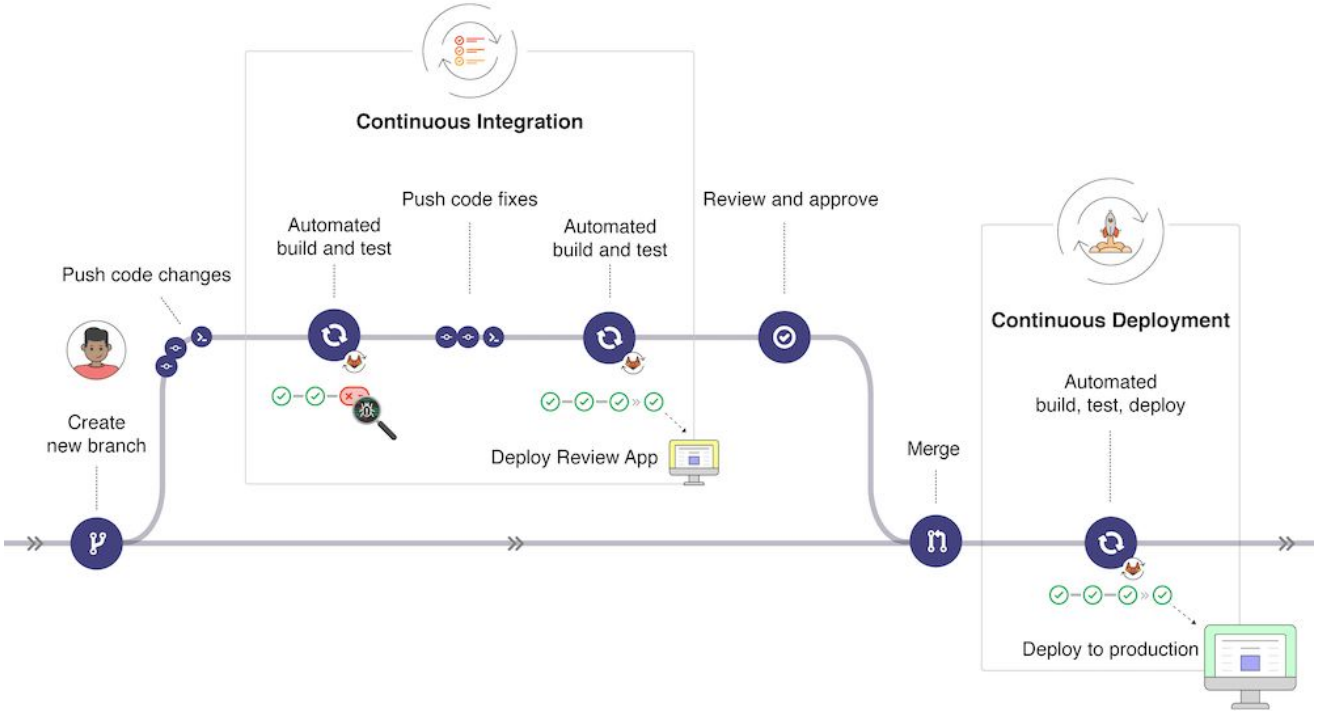
CI/CD allows organizations to **ship software quickly and efficiently.**

CI/CD facilitates an effective process for getting products to market faster than ever before, continuously delivering code into production, and ensuring an ongoing flow of new features and bug fixes via the most efficient delivery method.

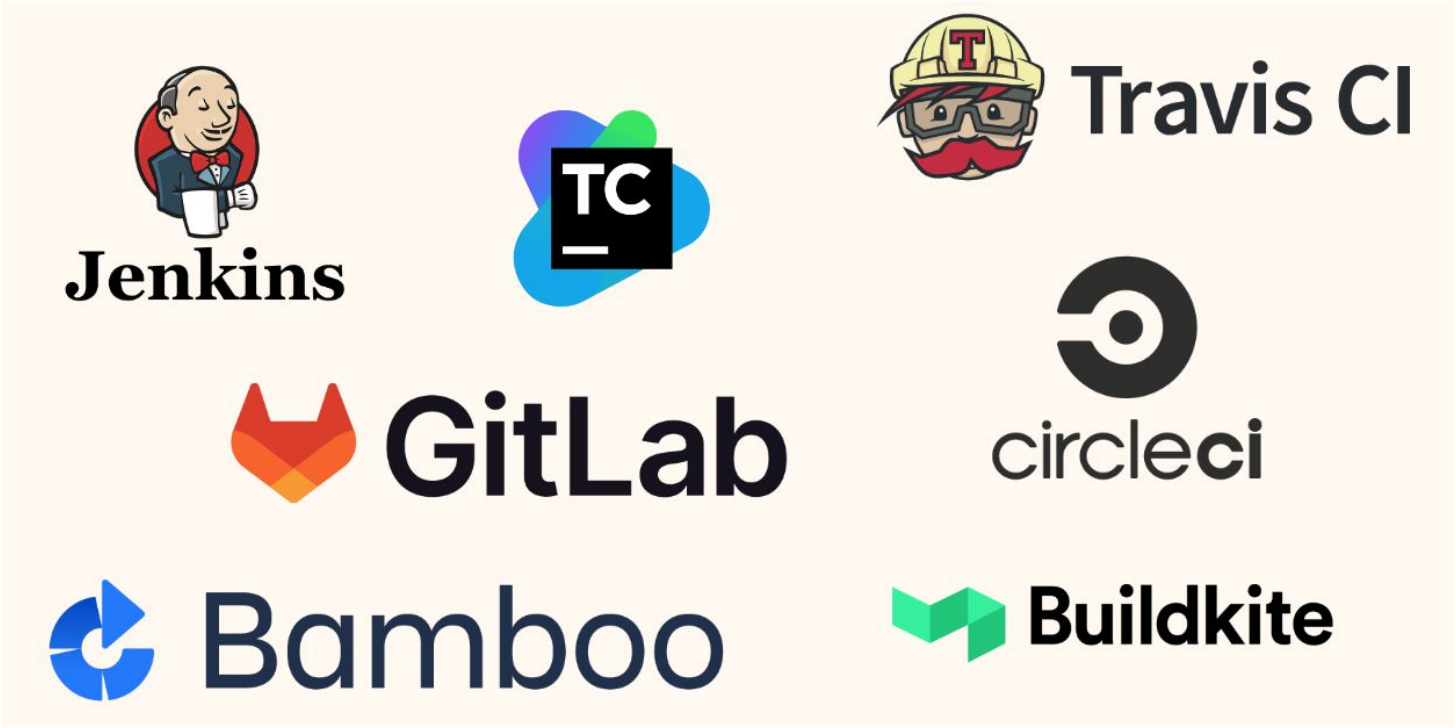
CI/CD


CONTINUOUS INTEGRATION
CONTINUOUS DELIVERY
CONTINUOUS DEPLOYMENT
BUILD
TEST
MERGE
AUTOMATICALLY RELEASE TO REPOSITORY
AUTOMATICALLY DEPLOY TO PRODUCTION

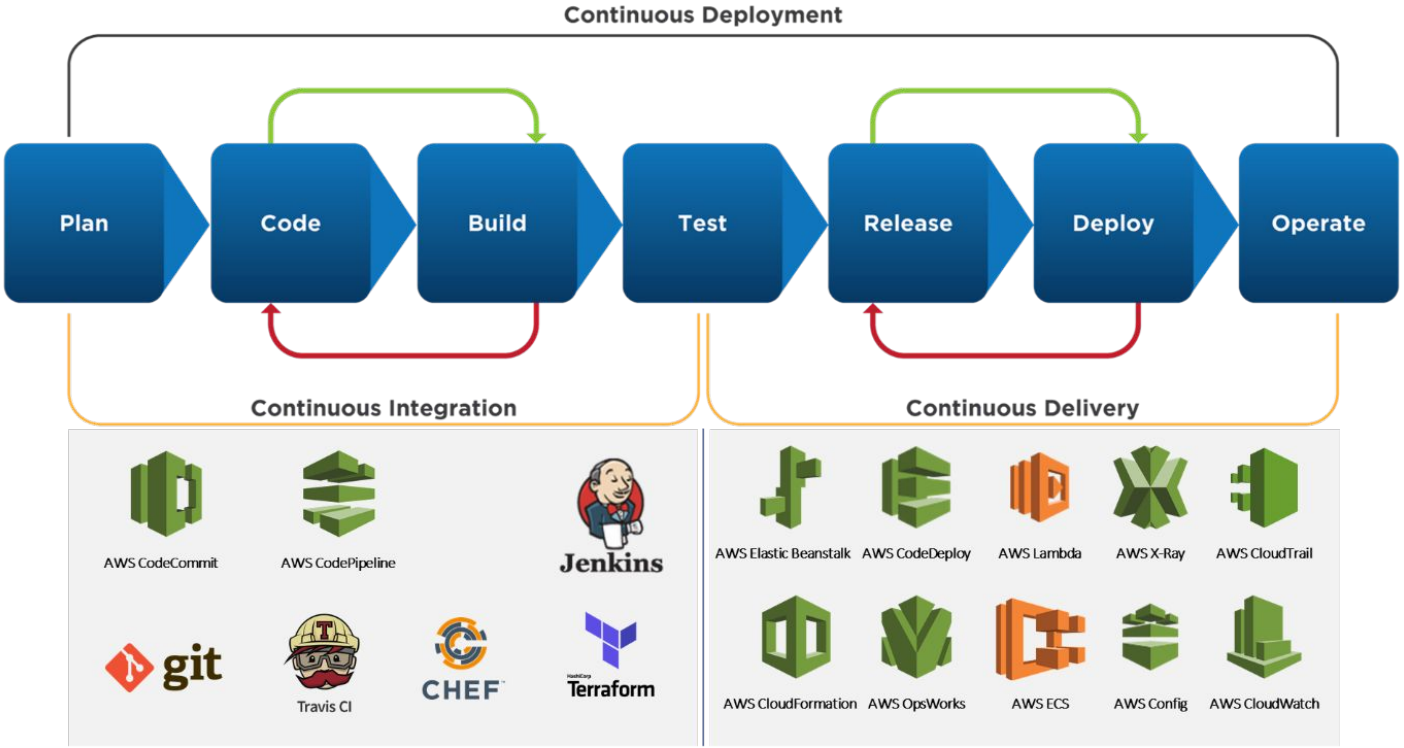
CI/CD


Continuous Integration
Push code fixes
Review and approve
Automated
build and test
Automated
build and test
Push code changes
Create
new branch
Deploy Review App
Merge
Continuous Deployment
Automated
build, test, deploy
Deploy to production

Common CI/CD Tools



AWS CI/CD Tools



Introduction To Containerization

Containerization Concepts

Containerization is a **form of virtualization** where applications run in isolated user spaces, called containers, while using the same shared operating system (OS).

One of the benefits of containerization is that a **container is essentially a fully packaged and portable computing environment**.

Containerization Concepts

Everything an application needs to run—its **binaries, libraries, configuration files, and dependencies**—is **encapsulated and isolated in its container**.

The container itself is abstracted away from the host OS, with only limited access to underlying resources—much like a lightweight virtual machine (VM).

As a result, the **containerized application can be run on various types of infrastructure**—on bare metal, within VMs, and in the cloud—without needing to refactor it for each environment.

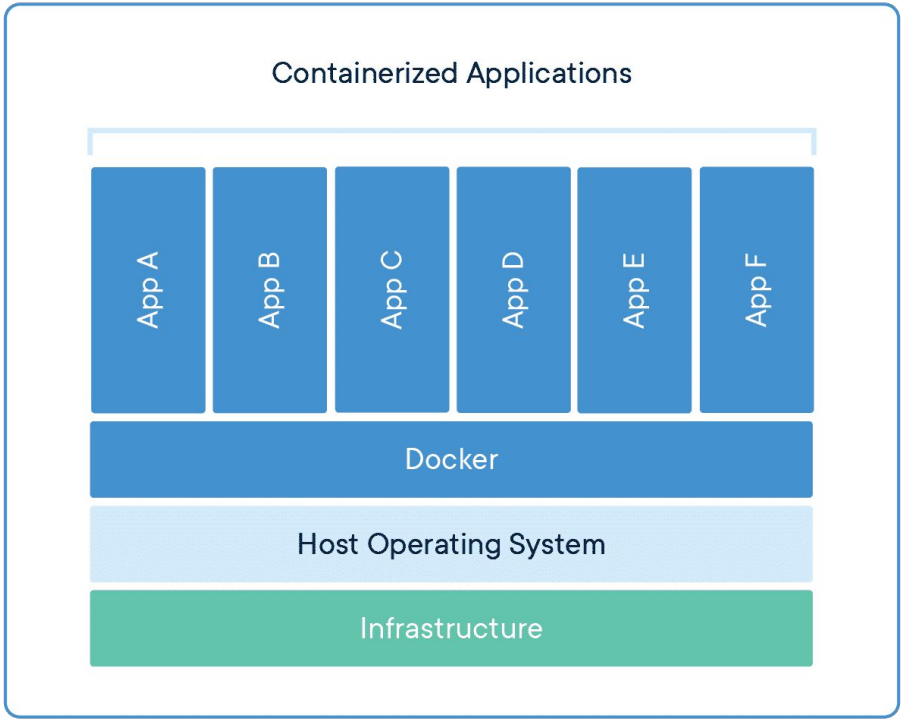
Containerization Concepts

With containerization technology, there's less overhead during startup and no need to set up a separate guest OS for each application since they all share the same OS kernel.

Because of this high efficiency, **containerization is commonly used for packaging up the many individual microservices that make up modern apps**.

Containerization Concepts

Containerization Concepts



Deployment Methodologies

Deployment Methodologies

It describes an **overall work process or roadmap for the project**.

Methodologies can include Agile development, DevOps, Rapid Application Development (RAD), Scaled Agile Framework (SAFe), Waterfall and others.

- Agile Methodology
- Waterfall Methodology

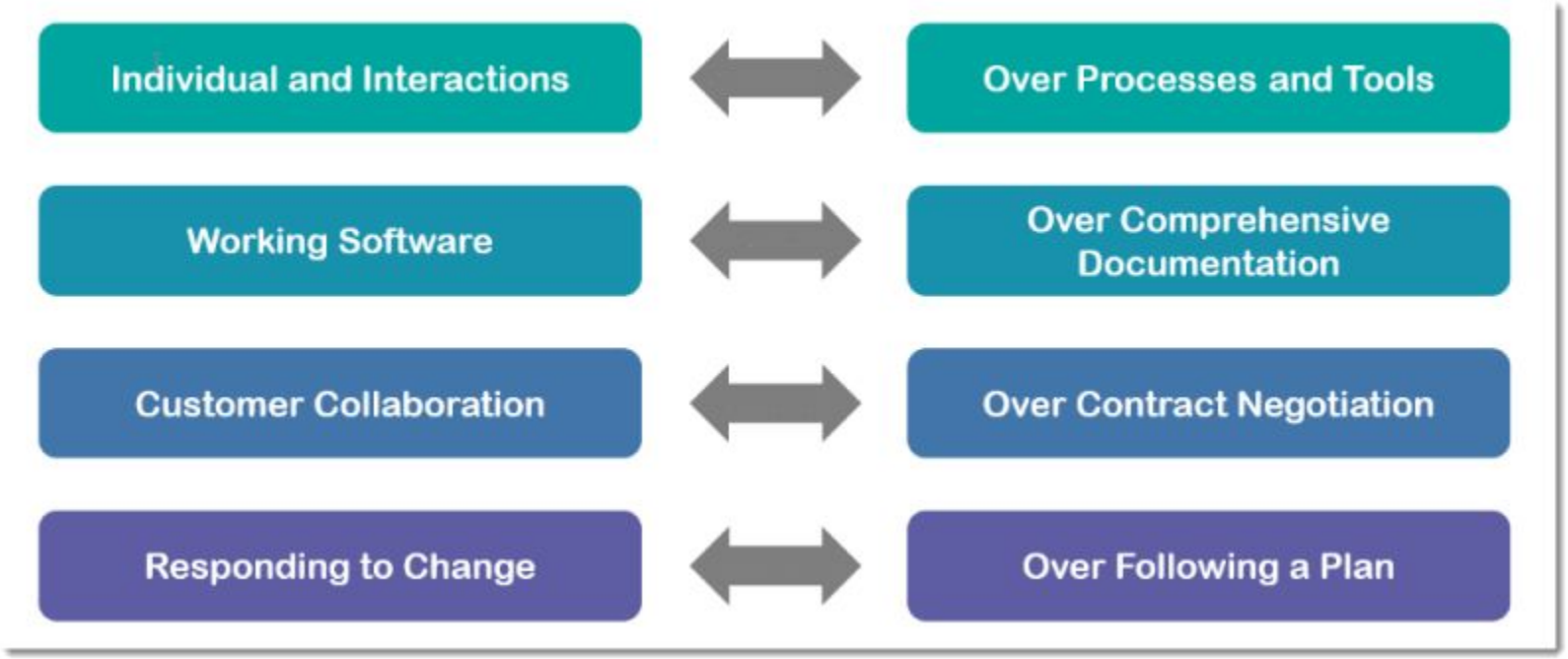
Agile

Agile is an **iterative approach to project management and software development** that helps teams deliver value to their customers faster and with fewer headaches.

Instead of betting everything on a "big bang" launch, an agile team **delivers work in small, but consumable, increments**.

Requirements, plans, and results are evaluated continuously so teams have a natural mechanism for responding to change quickly.

Agile


Individual and Interactions
Over Processes and Tools
Working Software
Over Comprehensive Documentation
Customer Collaboration
Over Contract Negotiation
Responding to Change
Over Following a Plan

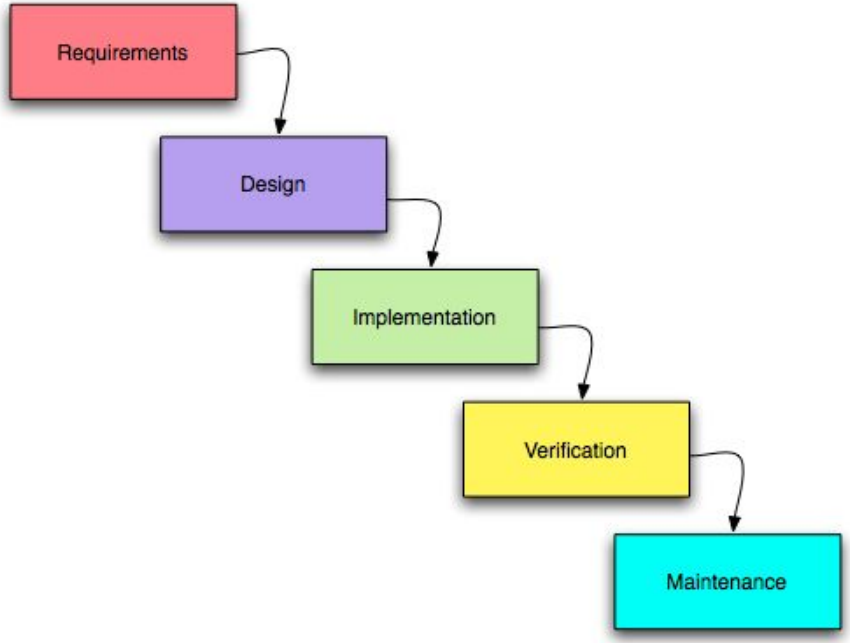
Waterfall

The Waterfall methodology — also known as the Waterfall model — is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase **completely wrapping up before the next phase begins**.

With the majority of the **research done upfront**, estimates of the time needed for each requirement are more accurate, and this can provide a more predictable release date.

With a Waterfall project, **if parameters change along the way, it's harder to change course than it is with Agile methodology**.

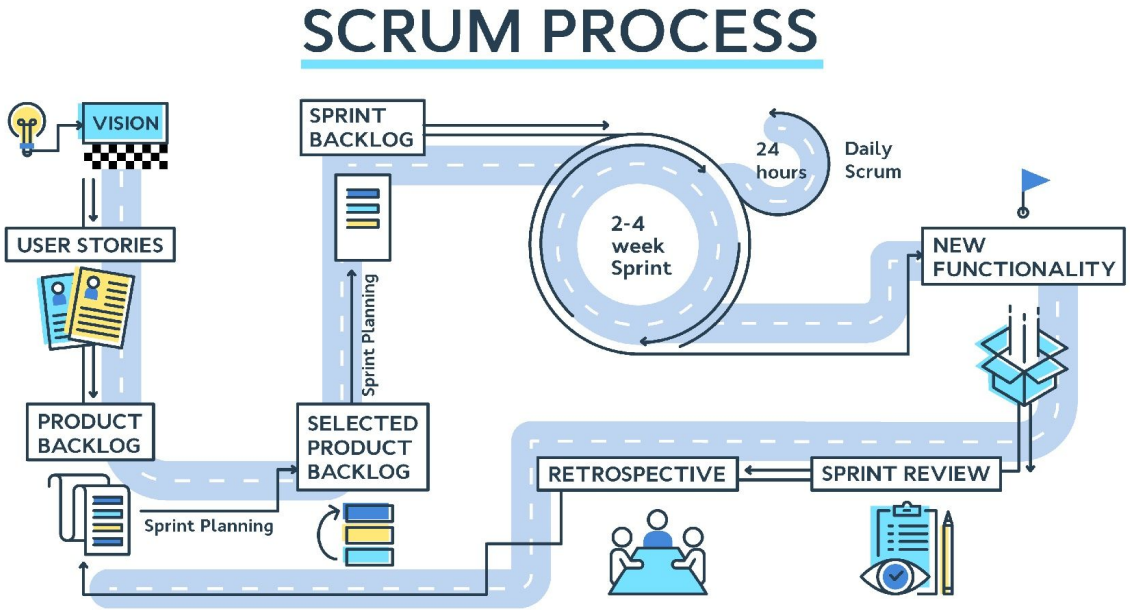
Waterfall


Requirements
Design
Implementation
Verification
Maintenance

Agile Principles

Agile Principles

SCRUM


SCRUM PROCESS
VISION
USER STORIES
PRODUCT BACKLOG
Sprint Planning
SELECTED PRODUCT BACKLOG
Sprint Planning
SPRINT BACKLOG
2-4 week Sprint
24 hours
Daily Scrum
NEW FUNCTIONALITY
SPRINT REVIEW
RETROSPECTIVE

SCRUM

In SCRUM, there are four meetings:

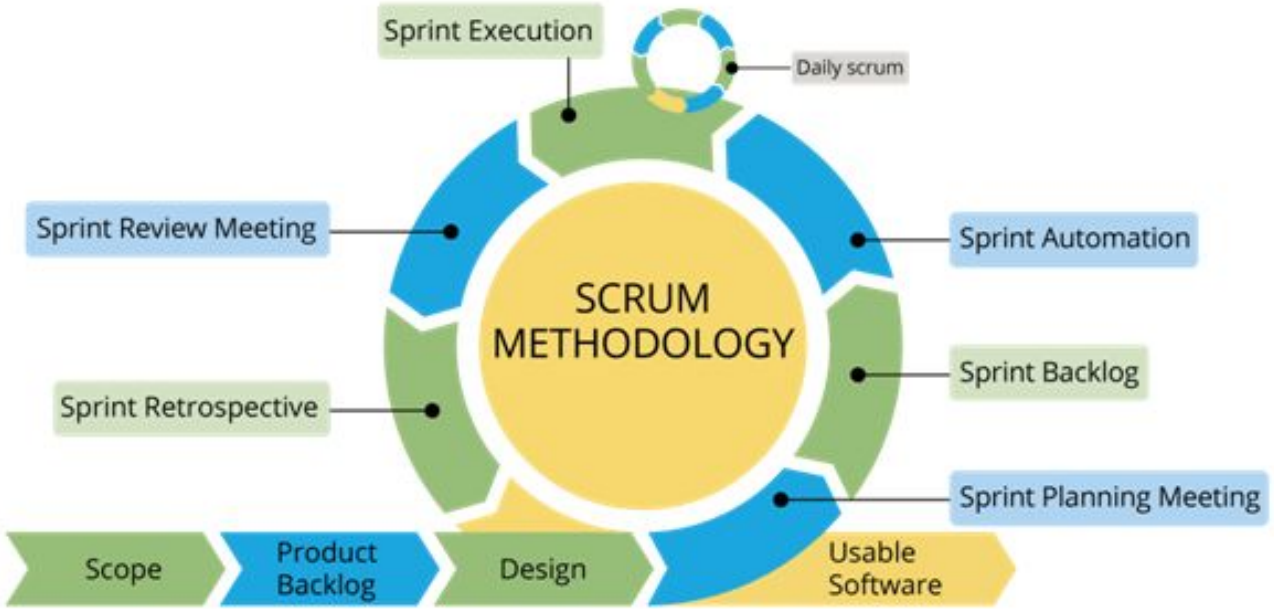
Sprint Planning - Laying out the work to be performed for the sprint. (4 to 8 hours)

Daily Scrum - Inspect progress toward the Sprint Goal. (15 mins)

Sprint Review - Inspect the outcome of the Sprint. (1 to 4 hours)

Sprint Retrospective - Plan ways to increase quality and effectiveness. (1 to 3 hours)

SCRUM


Sprint Execution
Daily scrum
Sprint Review Meeting
Sprint Automation
SCRUM METHODOLOGY
Sprint Backlog
Sprint Retrospective
Sprint Planning Meeting
Scope
Product Backlog
Design
Usable Software

Optional Class Activity

Action	Duration	Outcome
Learners self Reading for this link.	10 mins	Learners have a brief understanding of what agile user stories are.
Instructor facilitate students to answer three questions by getting students to type in chat. - What are user stories? - Why user stories? - Think & Provide examples of user stories	20 mins	Summarizes and highlight the good inputs from student.

Assignment

1. Discuss within your group about what the project scenario is e.g. “We are working for an ride-hailing company”.
2. Discuss within your group about what methodology you will use e.g. Agile/ Waterfall.
3. When it comes to deployment, discuss what software deployment strategy you will use and why.
4. Discuss what CI/CD Pipeline tools that you will use.
5. Discuss a simple pipeline that you will use and describe it on diagram.

Activity

Learner:

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.

Instructor

- Clean up AWS.
- Remove/delete/terminate all service/ resources that created.
- Check the AWS account after learner clean up.

What’s Next?