linkedin.com/rashidwassan

What is DevOps?
What is DevOps culture?
What is software development, and lifecycle models?
What is the difference between Waterfall and Agile models?
A brief history of DevOps.
DevOpsDays Conference, Arrested DevOps podcast (Google Podcast).
Download a Book: The DevOps Handbook and start reading it.

Pair Programming:
- Two programmers on one workstation.
- The driver is typing.
- The navigator is reviewing.
- Every 20 minutes they switch.

Benefits of Pair Programming:
- Higher code quality when you explain code to someone.
- Defects found earlier.
- Lower maintenance code.
- Skills transfer.
- Two sets of eyes on each line of code.
- A broader understanding of the codebase.

Git Repository Guidelines:
- Create a separate repo for every component.
- Repos with multiple microservices are called mono repo.
- Create a new branch for every issue.
- Use pull request to merge to master.
- Avoid merging your PR, have them look at your code.

Working in batches:
- Learn from lean manufacturing.
- Faster feedback.
- Supports experimentation.
- Minimizes waste.
- Deliver faster.

Measuring the size of the batches
- Feature size supports frequent releases.
- Features should be completed in a sprint.
- Features are a step towards the goal, keep them small.

Minimum Viable Product (MVP):

- MVP is not the first phase or first beta of the project.
- MVP is an experiment to test your value hypothesis and learn.
- At the end of each MVP, you will decide whether to pivot or persevere.
- An experiment may fail but gives a lesson.

TDD

What is test-driven development (TDD)?

- Test cases drive the design.
- You write test cases first and then write code to pass the test.
- This keeps you focused on the purpose of the code.
- Code is of no use if your client can't call it.

Why developers don't test?

- I already know my code works!
- I don't write broken code.
- I have no time. Testing saves a tonne of time by allowing you to write solid code.
- Basic TDD workflow: **Red -> Green -> Refactor** and the cycle repeats.

Why is TDD important for DevOps?

- It saves time while developing.
- You can code faster with more confidence.
- It ensures the code is working as expected.
- It ensures future changes will not break the code.
- To create a CI/CD pipeline, all testing must be automated.
- You cannot have continuous integration and CD without automated testing.

Behavior-Driven Development (BDD):

- Describe the behavior of the system from the outside.
- Great for integration testing.
- Uses a syntax that developers and stakeholders can easily understand.

TDD vs BDD:

- BDD ensures that you are building the **right thing**.
- TDD ensures that you are building the **thing right**.
- **Gherkin:** an easy-to-text natural language syntax.
- Understandable by everyone (given, when, then, and).

Cloud Native Microservices:

- Cloud-native means born on the cloud.
- The Twelve-Factor App
- A collection of stateless microservices.
- Each service maintains its database.
- Resilience through horizontal scaling.

- Failing instances are killed and respawned.
- Continuous delivery.
- **Microservices:** Developing an application with a suite of small services, each running in its process. The term was coined by Martin Fowler and James Lewis.

Designing for failure:
- Retry pattern. Keep retrying until we get the expected data from the server.
- Circuit breaker pattern. Return something useful in case of a service failure to avoid cascading failures.
- Bulkhead pattern.
- Chaos engineering aka Monkey testing: Simian Army tools from Netflix.

Working DevOps
- Culture of teaming and collaboration.
- Agile development as a shared discipline.
- Automate relentlessly.
- Push smaller releases faster.

Taylorism
- Adoption of command and control management.
- Organizations are divided into functional silos.
- Decision-making is separated from work.

Required DevOps Behaviors:
- Enterprises see “new” as complex and time-consuming.
- Continual series of small changes.
- These changes cannot survive traditional overheads.

Infrastructure as Code
- Describing infrastructure in a textual format that can be executed by configuration management tools like Puppet, Ansible, and Chef.
- Never perform configurations manually.
- Use version control systems.
- **Ephemeral immutable infrastructure:**
    - Server drift is a major cause of failure.
    - Servers are cattle, not pets.
    - Infrastructure is transient.
    - Build through parallel infrastructure.
    - Applications are packaged in containers.
    - You never make changes to running containers but images.

Continuous Integration:

- CI/CD are two separate things.
- **Continuous Integration** is the process of continuously building, testing and merging to master.
  - Developers integrate code often unlike traditional development.
  - Devs work on short-lived feature branches.
  - Each check-in is verified by an automated build.
  - Changes are kept small.
  - Working in small batches.
  - Committing regularly.
  - Using pull requests.
  - Committing all changes regularly.
  - CI automation is building and testing every pull request. Use CI tools to monitor version control. Tests should run after each build.
  - CI reduces code integration risks.
  - The master branch should always be deployable.

Continuous Delivery is continuously deploying to a production-like environment (may not be the production environment itself).

- The master branch should always be deployable.
- A CI/CD pipeline needs:
  - Code repository.
  - Build server.
  - An integration server.
  - An artifact repository.
- **Five Key Principles:**
  - Build quality in.
  - Work in small batches.
  - Computers perform repetitive tasks, people solve problems.
  - Relentlessly pursue continuous improvement.
  - Everyone is responsible.

Measure What Matters:

- Instead of creating competition among employees through promotions, the company can be using social engagements as the metric for promotions to foster collaborations.

Vanity Metrics:

- A vague metric like 10k hits on the website, 10k hits might be from a single person or 10k individual people, we do not know.

Actionable Metrics (examples):

- Reduce time to market.
- Increase overall availability.
- Reduce time to deploy.
- Defects detected before production.

- More efficient use of infrastructure.
- Quicker performance feedback.
- **Top four actionable metrics:**
    - Mean lead time.
    - Release frequency.
    - Change failure rate.
    - Mean time to recovery (MTTR).

DevOps Vs Site Reliability Engineering (SRE):

- … what happens when a software engineer is tasked with what used to be called operations.
- Only software engineers are hired in SRE.
- SRE teams are different from development teams.
- Stability is controlled through error budgets.
- Developers rotate through operations.
- SRE provides Platform, and DevOps uses the platform.

Summary of Organizational Impacts of DevOps:

Measure and reward what you want to improve.

- People seek information on what is rewarded and then seek to do that.
- Measuring social metrics leads to improved teamwork and measuring DevOps metrics allows you to see the progression toward your goals.
- If you want people to be social, then measure them being social.
- DevOps changes the objective of problem resolution from failure prevention to failure recovery.
- Vanity metrics may be appealing at first but offer limited actionable insights.
- Actionable metrics provide meaningful ways to measure your processes and take action toward goals.
- DevOps actionable metrics include mean lead time, release frequency, change failure rate, and mean time to recovery.
- You can rate statements developed by Dr. Nicole Forsgren to measure your team's culture, including statements about information, failures, collaboration, and new ideas.
- Mean lead time is the measure of how long it takes for an idea to get to production.
- The change failure rate is the rate of failure from pushing new releases out.
- The mean time to recovery is how long it takes to recover from a failure.
- Failures are learning opportunities that should not be punished.
- Dr. Nicole Forsgren developed cultural statements for measuring team culture.

Introduction to Cloud Computing:

Definition According to NIST:

- A model for enabling convenient, on-demand network access to shared configurable computing resources that can be rapidly provisioned and released with minimal management or service provider interaction.

5 Essential Characteristics of Cloud Computing:

- **On-demand Self-service.**
- **Broad network access.**
- **Resource pooling.**
- **Rapid elasticity.**
- **Measured service.**

Cloud Deployment Models:

- **Public**
- **Hybrid**
- **Private**

Service Models:

- **Infrastructure (IaaS)**
    - User: System Admin
    - Like leasing a car
    - Provides computing, network, and storage on-demand over the internet and pay-as-you-go basis.
    - Users can deploy VMs usually with OS pre-installed.
    - Object storage is the most common storage in the cloud as it is distributed and resilient.
    - **Benefits:**
        - Enables teams to set up test and deployment environments faster.
        - Helping developers focus more on business logic instead of infrastructure management.
        - Business continuity and disaster recovery.
        - Faster deployments and scaling.
        - High-performance computing.
        - Big data analysis.
    - IaaS concerns a lack of transparency and dependency on a third party.
- **Platform (PaaS)**
    - User: Dev
    - Like renting a car
    - **PaaS provides** Servers, networks, storage, operating system, application runtimes, APIs, middleware, and Databases.
    - The user is only responsible for the application code.
    - Provide a high level of abstraction.

    - Provide support service and APIs and runtime environments.
    - Rapid deployment mechanisms, and middleware capabilities.
    - **Use cases of PaaS**
        - API development and management.
        - Internet of things.
        - Business analytics/intelligence.
        - Business project management.
        - Master data management.
    - **Advantages:**
        - Scalability
        - Faster time to market
        - Greater agility and innovation
    - PaaS offerings: AWS Elastic Beanstalk, CloudFoundry, IBM CloudPaks, Azure, RedHat OpenShift, and Magento.
    - **Risks:**
        - Information security threats
        - Dependency on the service provider's infrastructure
        - Customers lack control over changes in strategy, service offerings, or tools.
- **Application (SaaS)**
    - User: Anyone
    - Like getting a taxi
    - Provides and maintains Servers, databases, application code, and security.
    - **SaaS Supports:**
        - Email and Collaboration
        - Customer relationship management
        - Human resource management
        - Financial management
    - **Key Characteristics of SaaS:**
        - Multitenant architecture
        - Manage privileges and monitor data
        - Security, compliance, and maintenance
        - Customize applications
        - Subscription model
    - **Concerns:** Data ownership and safety, a third party manages business-critical data, and needs a good internet connection.

Lesson summary of Cloud Computing Service Models:

- Cloud computing allows us to utilize technology as a service, leveraging remote resources on-demand, on a pay-as-you-model. There are three main service models available on the cloud—Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).
- IaaS provides the fundamental Compute, network, and storage resources for customers on-demand.

- PaaS provides customers with the hardware, software, and infrastructure to develop, deploy, manage, and run applications created by them or acquired from a third party.
- SaaS provides access to users to a service provider's cloud-based software. Users simply access the applications on Cloud while the Cloud provider maintains the infrastructure, platform, data, application code, security, availability, and performance of the application.

Key Drivers for Moving to the Cloud:

- **Agility**
- **Flexibility**
- **Competitiveness**

Cloud Deployment Models:

- **Public Cloud**
- **Private Cloud**
- **Hybrid Cloud**
    - **Cloud bursting** is the process of combining a company's private cloud with a 3rd party public cloud to meet the spike in resource demand.
    - Three tenants of hybrid cloud: Interoperability, portability, and scalability.
    - **Hybrid Cloud Models:**
        - Hybrid Monocloud: one cloud provider.
        - Hybrid Multicloud: can be deployed on any public cloud infrastructure.
        - Composite Multicloud: greater flexibility.

Lesson summary: Cloud Deployment Models:

- Deployment models indicate where the infrastructure resides, who owns and manages it, and how cloud resources and services are made available to users. There are three main deployment models available on the cloud—Public, Private, and Hybrid.
- In the Public cloud model, the service provider owns, manages, provisions, and maintains the physical infrastructure such as data centers, servers, networking equipment, and storage, with users accessing virtualized Compute, networking, and storage resources as services.
- In the Private cloud model, the provider provisions the cloud infrastructure for exclusive use by a single organization. The private cloud infrastructure can be internal to the organization and run on-premises. Or, it can be on a public cloud, as in the case of Virtual Private Clouds (VPC), and be owned, managed, and operated by the cloud provider.
- In the Hybrid cloud model, an organization's on-premise private cloud and third-party, public cloud are connected as a single, flexible infrastructure leveraging the features and benefits of both Public and Private clouds.

Cloud Infrastructure

Regions:

- Geographic areas where the cloud provider's infrastructure is clustered.
- Cloud regions are isolated from one another.
- Can have multiple **zones**.
- Zones are also isolated for fault tolerance but are connected with other AZs and data centers using high bandwidth internet.
- **Computing resources:**
  - Virtual Servers (VMs)
  - Bare metal servers (Physical servers)
  - Serverless (Abstraction layer on virtual machines)
- **Storage**
  - Block Storage
  - File Storage
  - Object Storage (most common)

Virtualization & Virtual Machines:

- **Virtual machines**
  - **Single tenant**
  - **Multi-tenant (shared).**
    - A physical server is virtualized.
    - Different sizes and configurations are available.
    - Can be charged on an hourly or monthly basis, sometimes even seconds are also charged.
  - **Transient or Spot VMs**
    - Better for not production uses like development and testing.
    - They are very cheap.
    - Best for temporary high computing cheap workloads.
  - **Reserved Virtual Server Instances:**
    - Can be reserved for a certain period.
  - **Dedicated Hosts:**
    - Single tenant isolation.
    - You will have to specify the data center and pod.
    - This allows maximum control over workload placement.
- **Hypervisor:**
  - Type 1: Bare metal. Most common.
  - Type 2: Host OS required.

- **Bare Metal Servers:**
  - The cloud provider manages the server up to the OS.
  - Everything else is managed by the customer.
  - Can be preconfigured or custom configured as per customer specifications.

  - GPUs can also be added for Data Analytics and rendering.
  - Take longer to provision i.e 30 min to 4hrs.
  - More expensive than VMs.
  - Only offered by a few providers.
  - Highly secure and controlled environment.
  - Complete flexibility and transparency.

Cybersecurity Threats:

- In cloud instances, we use logical or software-based networking solutions instead of physical ones.
- In zones, VMs, storages, and networks are distributed using subnets.
- Subnets allow users to deploy enterprise applications using the same multi-tier concepts uses in on-premises environments.
- Subnets are the main area where security is implemented in the cloud.
- Every subnet is controlled using Access Control Lists i.e subnet firewall.
- VSIs lie in subnets.

Containers

- What are containers?

Basics of Storage on Cloud:

- Certain data storage units should be attached to the node before computing gets started.
- Cost is per gegabytes.
- **Types**
  - **Direct Attached:**
    - Local storage.
    - And is present in the same chassis of the server or the same rack.
    - It is ephemeral.
    - Not shared and not resilient.
  - **File Storage:**
    - Uses NFS (Network File System).
    - Connected with remote database using ethernet.
    - It is a bit slower than direct-attached storage.
    - It is of low cost, however.
    - Attached to multiple servers.
    - More resilient for failure.
    - Provides both types of encryption (in transit and at rest).
    - Speeds may be volatile.
  - **Block Storage:**
    - High-speed fiber connections.
    - Faster and more reliable than file storage.
    - IOPS: Input/Output Operations Per Second.
    - Can be persistent and can be mounted to another compute node.

        - Snapshots can be taken for backups with incremental change writing using metadata. They can not be used to recover individual files.
        - They can cost more.
        - Usually connected to one only compute noted.
- **Object Storage:**
    - Cheapest and slowest of all.
    - Infinite in size.
    - You pay for what you use.
    - It is better for all types of data like docs and IoT data.
    - It can be used without connecting it to any compute node, via APIs.
    - It is effectively infinite.
    - Good for storing large amounts of unstructured data since there is no hierarchy.
    - Data is stored in the form of a bucket and each bucket contains objects.
    - There is not any specified size of a bucket.
    - Objects are static.
    - **Data Tiers:**
        - Also called classes that are classified based on how frequently data is accessed.
        - **Standard Tier**
            - Stores objects that are frequently accessed and cost more.
        - **Vault/Archive Tier:**
            - Stores data that is accessed about 1-2 times a month and is relatively cheap.
        - **Cold Vault Tier:**
            - Stores data that gets accessed once or twice a year.
            - Costs just a fraction of the US cent per gigabyte per month.
    - Automatic archive rules can be defined to move less frequently used data to cheaper storage.
    - **SPEED:**
        - Does not come with IOPS options.
        - Slower than block and file storage.
        - Data in ‘Cold Vault’ buckets can take hours to retrieve.
    - **APIs**
        - S3 API is most common to retrieve data in standard object storage (AWS).
        - HTTP-based REST API.
    - Can be used as a backup & disaster recovery solution.

Content Delivery Network

A content delivery network, or content distribution network, is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and performance by distributing the service spatially relative to end users.

Summary: Cloud Storage and Content Delivery Network:

- Cloud storage is available in four main types–Direct Attached, File, Block, and Object Storage. These storage types differ in how they can be accessed, the capacity they offer, how much they cost, the types of data they are best suited to store, and their read-write speed.
- Direct Attached (or Local) Storage is storage that is presented directly to a cloud-based server and is effectively either within the host server chassis or within the same rack.
- File Storage is typically presented to compute nodes as a Network File System (NFS), which means that the storage is connected to compute nodes over a standard ethernet network.
- Block Storage is presented to compute nodes using high-speed fiber connections, typically provisioned in volumes, which are mounted onto a compute node.
- Object Storage is accessed via an API and doesn't need an underlying compute node. Object Storage offers infinite capacity as you can keep adding files to it and just pay for what you use. Compared to the other storage types, object storage is the slowest in terms of read and write speeds.
- A Content Delivery Network (CDN) is a distributed server network that accelerates internet content delivery by delivering temporarily stored or cached copies of website or media content to users based on their geographic location.

Multi-Cloud

- Using multiple services from different cloud providers.
- Hybrid Multi-Cloud means using different cloud models by different cloud providers for optimized use.

Microservices

- Loosely coupled and independently deployable services.
- They have their stack running in their container and working in collaboration using service discovery.
- Allows multiple developers to work independently.
- Use of different stacks and runtime environments can be made.
- Allow independent scaling.

Serverless

- Offloads responsibility for common infrastructure management tasks such as scaling, scheduling, patching, and provisioning.

- Serverless doesn't mean that there are no servers, but the management is being taken care of so the user doesn't have to do any of that.
- In serverless computing, code runs on-demand, scaling as needed.
- Pay only when invoked and used.
- Abstracts the infrastructure away from developers.
- Code executed as individual functions.
- No prior execution context is required.
- AWS Lambda is an example of serverless computing.
- Serverless may cause vendor locking and cause latency issues.

Cloud-Native Applications

- Applications that are developed to only work in cloud environments.
- Or an existing app that has been reconfigured with cloud-native principles.
- Comprises many microservices.
- **Development Principles for Cloud Native Applications:**
    - Microservices Architecture
    - Rely on containers
    - Adopt agile methods

DevOps on Cloud

Application Modernization

- Monolithic Legacy Applications:
    - Siloed Systems
    - Difficult to Update
    - Expensive to Maintain
- Modernized Applications:
    - Accelerate Digital Transformation
    - Leverage New Tech & Services
    - Respond Faster to Change

- Cloud-native applications are applications that are built or refactored to work in the cloud environment. These applications, developed using DevOps methodologies, consist of microservices packaged in containers that can run in any environment—making it possible to create and update features in quick iterative cycles.
- DevOps is a collaborative approach that enables development and operations teams to continuously deliver software in quick iterative cycles while reducing overhead, duplication, and rework. DevOps' tools, practices, and processes help tackle the complexities and challenges posed by the cloud, allowing solutions to be delivered and updated —quickly and reliably.
- Application Modernization helps organizations accelerate their digital transformation, take advantage of new technologies and services, and become more responsive to

changing market dynamics. Cloud computing is one of the key enablers of application modernization.

What is Cloud Security?

- Make sure your data is encrypted.
- Key management is for professionals.
- Make sure to have good key management.

SecDevOps
DevSecOps

Identity and Access Management

- **Cloud Security Concerns:**
    - Data loss and leakage
    - Unauthorized access
    - Insecure interfaces and APIs
- The first line of defense.
- Authenticate and authorize users.
- Provide user-specific access.
- **Main Types of Users:**
    - **Administrative Users**
        - Cloud platform administrators, moderators, and managers.
    - **Developer Users**
        - Application developers, platform developers, and application publishers.
    - **Application Users**
- **Risk Mitigating Strategies:**
    - Provisioning users by specifying roles on resources for each user.
    - Password policies control the usage of special characters, minimum password lengths, and similar things.
    - Multifactor authentication is like time-based one-time passwords.
    - Immediate de-provisioning of access when users leave or change roles.
- **Access groups.**
- **Access policies** define how users, service IDs, and access groups, in the account are permitted to access resources.

Cloud Encryption

- Encrypts data.
- Data access control.
- Key management.
- Certificate management.
- **Data Protection States:** Rest, Transit (SSL, TLS), and in Use.
- **Server-Side Encryption:** Create and manage your keys or generate and manage keys on Cloud.

- **Client-Side Encryption:** This occurs before data is sent to the cloud; cloud providers cannot decrypt the data.

Multi-Cloud Data Encryption
- Data access management
- Integrated key management
- Sophisticated encryption

Key Management
- Encryption does not eliminate security risks.
- It separates security risk from the data itself.
- Keys need to be managed and protected against threats.

Key Management Services
- They enable customers to encrypt sensitive data at rest, easily create and manage the entire lifecycle of cryptographic keys, and protect data from cloud service providers.

Key Management Best Practices
- Storing encryption keys separate from the encrypted data.
- Taking backups offsite and auditing them regularly.
- Refreshing the keys periodically.
- Implementing multi-factor authentication for both master and recovery keys.

Cloud Monitoring
- Assess data, application, and infrastructure.
- **It helps in:**
    - Accelerate the diagnosis and resolution of performance incidents.
    - Control the cost of your monitoring infrastructure.
    - Mitigate the impact of abnormal situations with proactive notifications.
    - Get critical Kubernetes and container insights for dynamic microservice monitoring.
    - Troubleshoot your applications and infrastructure.
- **Best Practices**
    - Leverage end-user experience monitoring solutions.
    - Move all aspects of infrastructure under one monitoring platform.
    - Use monitoring tools that help us track usage and cost.
    - Increase cloud monitoring automation.
    - Simulate outages and breach scenarios.

Lesson Summary: Cloud Security and Monitoring
- Cloud security refers to the policies, technological procedures, services, and solutions designed to secure enterprise applications and data on the cloud against insider threats, data breaches, compliance issues, and organized security threats.

- Cloud security is a shared responsibility between the cloud provider and the user organization.
- Security architecture and methods for achieving continuous security need to be embedded through the life cycle of an application to ensure that the application runs on a safe platform, the code is free from vulnerabilities, and the operational risks are understood.
- Identity and Access Management, also known as access control, helps authenticate and authorize users and provide user-specific access to cloud resources, services, and applications.
- As part of their Identity and Access Management services, most cloud providers offer users the ability to define access groups and create access policies that define permissions for users on account resources.
- Cloud encryption, often referred to as the last line of defense, not only encrypts data, but also provides robust data access control, key management, and certificate management.
- Data needs encryption in three states -
  - Encryption at rest; protecting data while it is stored
  - Encryption in transit; protecting data while it is transmitted from one location to another
  - Encryption in use; protecting data when it is in use in memory
- There needs to be active monitoring of all connected systems and cloud-based services to maintain visibility of all data exchanges between public, private, and hybrid cloud environments. This ensures that the cloud provides a trusted platform that can securely integrate with your enterprise data centers.

Businesses all over the world are realizing tangible benefits from the use of cloud technologies and services.

- The Weather Company migrating to the cloud to reliably deliver critical weather data at high speed, especially during major weather events such as hurricanes and tornadoes
- American Airlines uses the cloud platform and technologies to deliver digital self-service tools and customer value more rapidly across its enterprise
- Cementos Pacasmayo achieving operational excellence and insight to help drive strategic transformation and reach new markets using cloud services
- Welch chose cloud storage to drive business value from hybrid cloud
- LiquidPower uses cloud-based SAP applications to fuel business growth

The market size of the cloud services industry is nearly three times the growth of overall IT services, escalating the need for qualified cloud computing professionals. Some of the common job roles that are available in this domain include Cloud Software Engineers, Cloud Integration Specialists, Cloud Data Engineers, Cloud Security Engineers, Cloud DevOps Engineers, and Cloud Solution Architects

Introduction to Agile Philosophy

What is agile?

- Iterative approach.
- **Agile emphasizes:**
    - Adaptive planning.
    - Evolutionary development.
    - Early delivery.
    - Continual improvement.
    - Responsiveness to change.

Agile Manifesto

- Manifesto for Agile Software Development (agilemanifesto.org)

Agile Software Development

- An iterative approach towards Software Development consistent with the agile manifesto.
- Emphasizes flexibility, interactivity, and a high level of transparency.
- Uses small, co-located, cross-functional, and self-organizing teams.
- Build what is needed, not what was planned.

Waterfall Model Revisited

- Requirements -> Design -> Code -> Integration -> Test
- **Problems with the Waterfall Approach:**
    - No provisions for changing requirements.
    - No idea if it works until the end.
    - Each step ends when the next one begins.
    - Mistakes found later on are expensive to fix.
    - Long lead times.
    - Teams are working separately, unaware of their impact on each other.

Extreme Programming

- One of the first agile methods as it is an iterative approach.
- **Extreme Programming Values:**
    - Simplicity
    - Communication
    - Feedback
    - Respect
    - Courage

What is Kanban?

- A Japanese manufacturing system in which the supply of components is regulated through the use of an instructions card sent along the production line.

- A Kanban system is characterized by visualizing workflow, limiting work in progress, managing and enhancing flow, making process policies explicit, and continuously improving a process
- **Core Principles of Kanban:**
    - Visualize the workflow.
    - Limit Work in Progress.
    - Manage and enhance the flow.
    - Make process policies explicit.
    - Continuously improve.

Introduction to Scrum Methodology

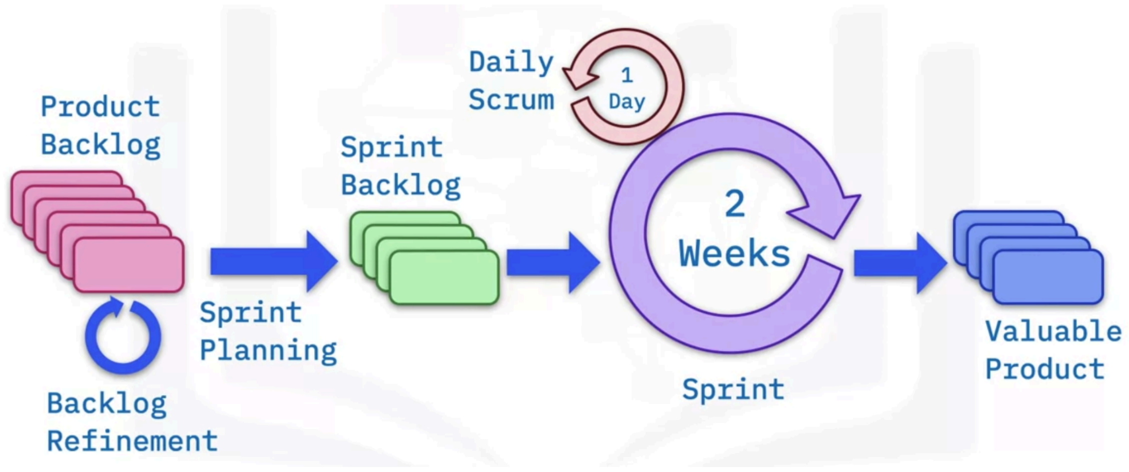
Scrum Overview

- Agile and scrum are not the same.
- Agile is a philosophy of doing work.
- Scrum is a methodology for doing work in an agile manner.
- **What is scrum?**
    - It is a management framework for incremental product development.
    - It emphasizes working in small, cross-functional teams.
    - Provides a structure of rules, roles, and artifacts.
    - Uses fixed-length iterations called sprints.
    - Has a goal to build a potentially shippable product increment with every iteration.

Sprint

- A sprint is one iteration through the design, code, test, and deployment cycle.
- Every sprint should have a goal.
- Sprints are usually 2 weeks in duration.

Steps in the Scrum Process



Roles in Scrum

- **Product Owner**
    - Represents the stakeholder interests.
    - Articulate product vision.
    - Is the final arbiter of requirements questions.
    - Constantly re-prioritizes the product backlog, adjusting any expectations.
    - Accepts or rejects each product increment.
    - Decides whether to ship.
    - Decides whether to continue development.
- **Scrum Master**
    - Facilitates the Scrum process.
    - Coaches the team.
    - Creates an environment to allow the team to be self-organizing.
    - Shields the team from external interference to keep it in the zone.
    - Help resolve impediments.
    - Enforces sprint timeboxes.
    - Captures empirical data to adjust forecasts.
    - Has no management authority over the team.
- **Scrum Team**
    - A cross-functional team consisting of developers, testers, business analysts, domain experts, and others.
    - Self-organizing team with no externally assigned roles.
    - Self-managing as they self-assign their work.
    - Usually consists of 5-9 collaborative members.
    - Co-located.
    - Dedicated.
    - Negotiate commitments with the product owner.

Planning to be Agile

- **Destination Unknown:**
    - Plan iteratively.
    - Don’t decide everything at the point when you know the least.
    - Plan for what you know.
    - Adjust as you know more.
    - Your estimates will be more accurate.
- **Agile Roles and Need for Training**
    - Formulas of failure: assigning people new roles without training them.
    - Product Manager vs Product Owner
    - Project Manager vs Scrum Master
    - Development Team vs Scrum Team
- **Kanban and Agile Planning Tools**

User Stories:

- **Story Contents**
    - Brief description of need and business value.
    - Any assumptions or details?
    - The definition of “Done”.
    - Given some conditions.
    - Gherkin syntax is used.
    - **INVEST Story**
        - Independent
        - Negotiable
        - Valuable
        - Estimable
        - Small
        - Testable
    - **Epic**
        - A big idea
        - A user story that is bigger than a single sprint
        - Backlogs usually start as epics and become stories when defined
        - For sprint planning, Epics need to be broken into stories
- **Effectively Using Story Points**
    - The story point is an abstract measure of the overall effort.
    - Used to measure the difficulty of implementing a user story.
    - Story points acknowledge that humans are bad at estimating time-to-completion.
    - Story points are like T-Shirt sizes.
    - Most tools use the Fibonacci series.
    - Since story points are relative, you will have to agree on what a particular size is.
    - **Story Size**
        - A story should be small enough to be coded and tested in a single sprint iteration, ideally, just a few days.
        - Large stories should be broken down into smaller ones.
    - **Story point anti-pattern**
        - Evaluating a story to wall-clock time.
        - Humans are bad at estimating wall clock time.

- **Building the Product Backlog**
    - The product backlog contains all the unimplemented stories not in a sprint.
    - Stories are ranked in order of importance or business value.
    - Series are more detailed at the top, and less detailed at the bottom.
    - **Sample requirements**
        - **As a …**
        - **I need …**
        - **So that …**
- **Lesson Summary - User Stories:**

● A user story documents a person requesting a function to achieve a goal.

- Using a template helps ensure that stories are complete.
- Defining "done" helps minimize misunderstandings.
- Use the INVEST acronym to remember the qualities of a good user story: independent, negotiable, valuable, estimable, small, and testable.
- Epics can be used to capture big ideas.
- Story points are a metric used to estimate the difficulty of implementing a given user story.
- Story points are relative, like T-Shirt sizes.
- You must agree on what "average" means.
- You should never equate story points with wall clock time.
- A product backlog is a ranked list of all unimplemented stories.
- Stories high in the ranking should have more detail than those that are lower.
- Create stories using the "As a", "I need", and "So that" template to ensure everyone understands who it benefits and the business value it provides.

Backlog Refinement

- Keep important stories on top.
- Break large stories near the top into smaller ones.
- Make sure that stories near the top of the backlog are groomed and complete.
- **Backlog refinement meeting:**
    - Who should attend?
        - Product owner.
        - Scrum master.
        - Development team (optional and only one of them would be enough)
    - What is the goal?
        - Groom the backlog by ranking the stories in order of importance.
        - Make sure the story contains enough information for a developer to start working on it.
- **New Issue Triage**
    - Start with new issue triage.
    - Goal: At the end of backlog refinement, the New Issues column is empty.
    - Take stories from new issues and move them into product backlog or icebox if of less priority. You can also reject them.
- **Backlog refinement workflow**
    - The product owner sorts the product log in order of importance.
    - The team may provide estimates and other technical information.
    - Large vague items are split and clarified.
    - The goal is to make stories “sprint ready”.
- **Labels**
    - Labels in GitHub
    - The yellow label should be for the technical dept as well.
    - **Examples of Technical Debt:**
        - Code refactoring.

        - Set up and maintain environments.
        - Changing technology, like databases.
        - Updating vulnerable libraries.
    - **Backlog refinement tips**
        - Refine backlog every sprint to ensure the priorities are correct.
        - Have at least two sprints' worth of stories groomed.

Sprint Planning

- Used to define what can be delivered in the sprint and how that work will be achieved.
- This is accomplished by producing a sprint backlog.
- **Sprint Planning Meeting**
    - Should be attended by Product Owner, Scrum Master, and the Development Team.
- **Goals:** Sprint goals should be clear. The product owner describes the goal and and product backlog items supporting.
- **Mechanics of Sprint Planning**
    - **The development team** takes stories from the product backlog and assigns them to the sprint backlog. It also assigns story points and labels. It also ensures each story has enough information for a developer to start working on it. It stops adding stories when team velocity is reached.
    - **Team Velocity** is the number of story points a team can complete in a single sprint. It can be changed over a period and it is unique to a team and cannot be compared to others.
- **Create a Sprint Milestone**
    - Create a sprint milestone to start the sprint.
    - The milestone title should be short.
    - The description should document the milestone goal.
    - Duration should be 2 weeks.

Lesson Summary: The Planning Process

- It is the product owner's responsibility to maintain a groomed backlog
- Backlog refinement is used to order the product backlog and make stories sprint ready
- You start refinement by triaging new issues
- Large stories should be broken down until they are small enough to fit in a sprint
- The goal of backlog refinement is to get the backlog ready for the sprint planning meeting
- It is the product owner's responsibility to present the sprint goal
- It is the development team's responsibility to create a sprint plan
- A sprint plan is created by moving stories from the product backlog into the sprint backlog until the team's velocity is reached

Daily Workflow:

- **Daily Execution:**

    - Take the next highest priority item from the sprint backlog.
    - Assign it to yourself.
    - Move it in progress/process.
    - No one should have more than one story assigned to them unless they are blocked.
    - When you are finished, you create a pull request and move story to QA.
    - When the PR is merged, move the story to 'done' column.

Daily Stand-up

- Occurs every day at the same time and place.
- AKA Daily Scrum.
- Each member briefly report on their work.
- Timeboxed to 15 minutes.
- No one is allowed to sit.
- This is not a project status meeting.
- **Who should attend?**
    - Scrum master.
    - Development team.
    - Product onwer (optional).
- **Daily stand-up questions:**
    - What did I accomplish the previous day?
    - What will I work on today?
    - What blockers or impediments are in my way?
- **Impediments and blockers**
    - Impediments identified by the team should be unblocked by the scrum master.
    - Developers that are blocked should work on the next story.
- **Tabled topics**
    - Topics that are raised during the daily stand-up should be held until the meeting has ended.
    - Anyone interested in those topics can stay to discuss.

Lesson Summary: Executing the Plan:
You need to keep the kanban board updated so that everyone knows what you are working on

- It is important to always work on the story with the highest priority that you have skills for
- Working on more than one story at a time may lead to neither story being finished at the end of the sprint
- The daily stand-up occurs every day for 15 minutes
- Topics not related to the stand-up should be addressed after the meeting
  Each person should be prepared to answer the three stand-up questions:
- What did I accomplish the previous day?
  What will I work on today?
  What blockers or impediments are in my way?

Completing the Sprint

- **Milestones and Burndowns**
  - Milestones can be created for anything in your project.
  - Burndown charts are used to measure progress against any milestone.
- **Burndown Charts**
  - The measure of story points completed vs story points remaining for a sprint.
  - Over time remaining story points should go down, hence the name: burndown.

The Sprint Review

- It is a demo time.
- Live demonstration of implemented stories.
- The product owner evaluates if stories are done right.
- Done stories are closed.
- **Who should attend?**
  - Anybody can join (including optional customers).
- Feedback gets converted into new product backlog stories.
- This is where iterative development allows the creation of products that couldn't have been specified up-front in a plan-driven approach.
- **Rejected Stories**
  - What about stories that are not done?
  - Add a label to indicate this and close them.
  - Write a new story with new acceptance criteria.
  - This will keep the velocity more accurate since the work done on incomplete stories will also be counted.

The Sprint Retrospective

- A meeting to reflect on the sprint.
- Measures the health of the process.
- The development team must be comfortable speaking freely.
- **Who should attend The Sprint Retrospective Meeting?**
  - Scrum master.
  - Development team.
- **Three questions are asked:**
  - What went well?
  - What did not go well?
  - What should be changed for the next sprint?
- **The goal is the improvement**

Lesson Summary: Completing the Sprint

- A burndown chart shows the measurement of story points completed vs story points remaining for a sprint
- Burndown charts can be used to show progress and forecast the team's probability of achieving the sprint goal

- A sprint review is a demonstration of the features that have been implemented during the sprint
- Feedback from stakeholders is critical to help shape the future of the product
- The backlog is updated based on feedback
- A sprint retrospective is a time to reflect on how the sprint went
- The sprint retrospective is attended by the scrum master and the development team
- The team must feel comfortable speaking freely
- A sprint retrospective must result in changes to improve the next sprint
- Three questions are answered on what went right or wrong:

- What went well? (keep doing)
- What did not go well? (stop doing)
- What should we change for the next sprint?

Measuring Success

- Vanity metrics
- Actionable metrics

Getting Ready for Next Sprint

- End of sprint activities:
    - Move stories from done to close.
    - Close the current milestone.
    - Create a new sprint milestone.
    - Adjust unfinished work.
- **Handling Untouched Stories**
    - These stories can be moved to the top of the backlog.
    - Resist the urge to move them into the next sprint.
- **Handling Unfinished Stories**
    - Do not move unfinished stories into your next sprint.
    - Give the developer credit for the work they did.
    - This will keep the velocity more accurate.
    - Adjust the description and add an unfinished label and move it to the done.
    - Write a new story for the remaining work.
    - Assign the remaining story points and move them to the next sprint.
- **Ready for the Next Sprint**
    - All stories assigned to the current sprint are closed.
    - All unfinished stories are reassigned.
    - The sprint milestone is closed.
    - A new sprint milestone is created.

Agile Anti-Patterns and Health Check

- No real product owner.
- Multiple owners.
- Teams are too large ideal team should be less than 10.
- Teams are not dedicated.

- Teams are too geographically distributed.
- Teams are siloed.
- Teams are not self-managing.
- **Scrum Health Check**
    - Proper accountability.
    - Work is organized in consecutive sprints of 2-4 weeks or fewer.
    - There is an ordered product backlog.
    - Sprint backlog with visualization of remaining work for the sprint.
    - At sprint planning, a forecast, a sprint backlog, and a sprint goal should be created.
    - The result of the daily scrum is work being re-planned for the next day.
    - Stakeholders provide feedback as a result of inspecting the increment at the sprint review.
    - The product log is updated as a result of the sprint review.

Introduction to Software Engineering

Different Roles:

- **Developer Advocate**
  - Developer advocates help software developers be successful.
  - They create a lot of content and attend conferences.
- **Junior Software Engineer**
  - Creates new features, fixes bugs, and addresses customer concerns.
- **Software Engineer**
  - Works on the suite of products.

What is Software Engineering?

- Application of scientific principles to the design and creation of software.
- A systematic approach to software development.
- Initially, software development lacked formal development.
- The software crisis began in the 1960s when software developers ran over the budget and behind schedule with buggy code.
- Sometimes, solutions could not scale to larger projects.
- Now, this issue has been minimized due to the consistent applications of engineering principles.

Software Engineer vs Software Developer

- Software Engineers are also developers.
- The developer is a narrower scope than a software engineer.
- Software engineers have broad and big-picture knowledge bases.
- Developers can have creative approaches.
- Software engineer follows a systematic development process.
- Software engineer focuses on structure rather than solving a small problem.

Software Development Lifecycle (SDLC)

- A systematic way to develop high-quality software.
- A scientific approach to software development.
- Guides the software development process.
- Identifies discrete steps needed to develop software.
- Aims to produce software that meets requirements.
- Defined phases with their processes and deliverables.
- Minimizes development risks and costs.
- **Advantages of SDLC**
  - Improves efficiency and reduces risks.
  - Team members know what they should be working on and when.
  - Facilitates communication among stakeholders.
  - Team members know when development can move to the next phase.

- Respond to changing requirements.
- Problems are solved earlier.
- Reduces overlapping responsibilities.
- **Phases of SDLC**
    - Planning, Design, Development, Testing, Deployment, and Maintainance.
    - These names along with their count can be different.

Building Quality Software

- **Requirement Gathering**
    - SRS and use cases used.
    - **Requirement Categories:**
        - **Functional**
        - **External & User Interface**
        - **System Features**
        - **Non Functional**
- **Design**
    - Transforming requirements into code structure.
    - Breaking down requirements into sets of related components.
    - Communicating business rules and application logic.
- **Coding for Quality**
    - Quality code must fulfill the requirements of the software without defects.
- **Testing**
    - Unit testing
    - Integration testing
    - System testing
    - User acceptance test (beta testing)
- **Testing Categories:**
    - Functional
    - Nonfunctional
    - Regression
- **Releases**
    - **Alpha:** Selected stakeholders, may contain errors, and design changes may occur.
    - **Beta:** All stakeholders, and user testing, meet requirements.
    - **General Availability (GA)**
        - Stable and for all users.
- **Documenting**
    - System documentation (Technical)
    - User documentation (Non-Technical)

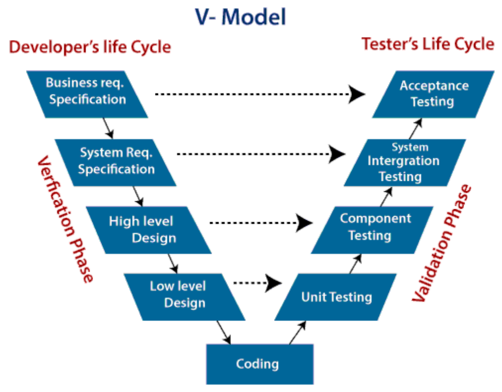
Requirements

- **Steps to gathering requirements**
    - **Identifying stakeholders**

        - Decision makers, end-users, system administrators, engineering, marketing, sales, and customer support.
    - **Establishing goals and objectives**
        - **Goals:** Broad, long-term achievable outcomes.
        - **Objectives:** Actionable and measurable actions that achieve the goal.
    - **Eliciting requirements from stakeholders**
        - Surveys, questionnaires, and interviews.
    - **Documenting the requirements**
        - Align with goals and objectives
        - Easily understood
        - **Three documents can be formed from this process:**
            - **Software requirements specification (SRS)**
                - Captures the functionalities that software should perform.
                - Establishes benchmarks/service labels for performance.
                - The document contains purpose and scope, constraints, assumptions, and dependencies.
                - Contains requirements sorted into four categories.
            - **User requirements specification (URS)**
                - Contains user stories.
            - **System requirements specification (SysRS)**
    - **Analyzing and confirming the requirements**
        - Consistency
        - Clarity
        - Completeness
    - **Prioritizing the requirements**
        - Labels like: must-have, highly desired, and nice to have, are helpful.

Common Development Methodologies

- **Waterfall:**
- **V-shape Model**
    - Easy to use but rigid along with efficient testing.



- **Agile**

Software Versions

- Software versions are identified by version numbers.
- Used by developers to keep track of changes.
- Some version numbers have four parts separated by a period.
- **Version number breakdown:**
  - The first number indicated major changes to the software.
  - The second indicates minor change.
  - Third for patches and bug fixes.
  - Fourth for build number, build dates, and less significant changes made.
- Version compatibility is an issue.

Software Testing

- **Test Cases**
  - To verify functionality and requirements.
  - Test cases contain Steps, data, input, and expected outputs.
  - Test cases are written after the requirements are finalized.
- **Three types of testing**
  - **Functional**
  - **Non Functional**
  - **Regression**

Introduction to Application Development Tools

- The build tool is used to transform your source code into binaries for installation. Build tools can automate the tasks of downloading dependencies, compiling source code into binary code; packaging that binary code, running tests; and deployments.
- Packages make the app easy to install.

Introduction to Software Stacks

- Combination of technologies.
- Consists of: Presentation layer, logic layer, and data layer.
- Complex stacks may consist of a Presentation, logic layer, data layer, security, virtualization, and orchestration.
- LAMP stack: Linux, Apache, PostgreSQL, and Python.

Interpreted & Compiled Languages

- Interpreted languages are also called scripting languages.
  - Python, JavaScript, Lua, and HTML.
- Compiled programs are usually compiled into one file.
- Java is a compiled language.

Choosing a Programming Language

- Trust, Experience, and Efficiency.

Query and Assembly Programming Language

Two Ways of Organizing the Code
- **Flowchart**
- **Pseudo Code**

Software Architecture
- Addresses nonfunctional aspects.

Software Design
- Process of documenting:
    - Structural components
    - Behavioral attributes
- Models express software design using:
    - Diagrams and flowcharts.
    - UML
- The behavioral model describes what a system does, but does not explain how it does it.
- Behavioral UML diagrams: state transition, interaction.

Component
- A component is an individual unit of encapsulated functionality.

What does a Software Engineer do?
-

Hands-on Introduction to Linux

Operating System
- A software that manages computer hardware resources
- Allows interaction with hardware to perform useful tasks.

Unix
- Unix is a family of operating systems like Oracle Solaris, FreeBSD, HP - UX, IBM AIX, and Apple MacOS.
- MacOS was derived from BSD.

What is a shell?
- The user interface for running commands
- Interactive language
- Scripting language

A sea of shells

- The default shell is usually bash.
- Many other shells include **sh, ksh, tcsh, zsh**, and fish.

Shell Scripting

Script:

- List of commands interpreted by a scripting language.
- Scripting languages are interpreted at runtime.
- Shell script - executive text file with an interpreter directive.
- The interpreter directive is also known as the “shebang” directive.

```
#!interpreter [optional-arg]
```

- ‘interpreter’ - path to an executable file.
- ‘optional-arg’ - single argument string.

Shell script directives:

```
#! /bin/sh
```

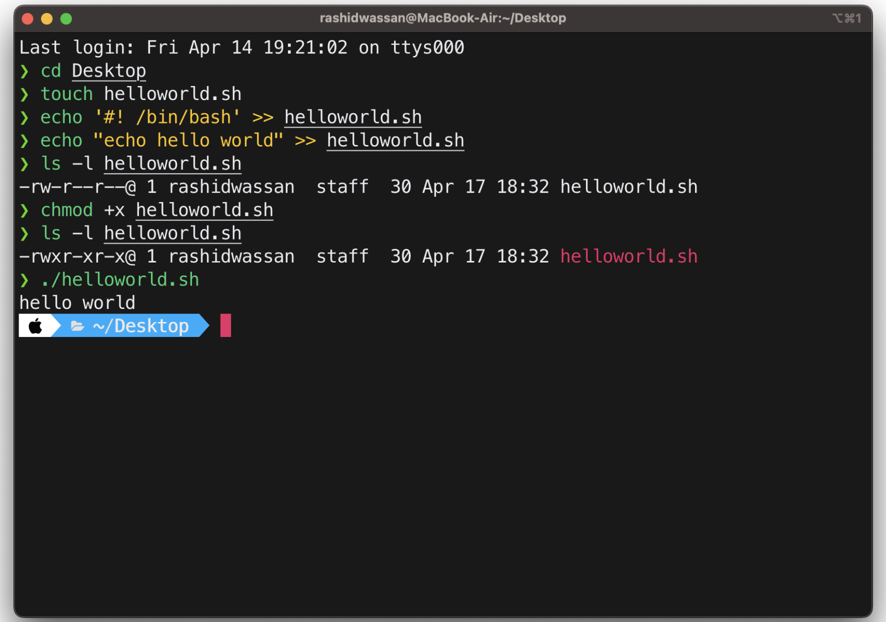
- Invokes Bourn or another compatible shell from the bin directory.

```
#! /bin/bash
```

Python Script Directive:

```
#! /usr/bin/env python3
```

Hello World example shell script:



```
Last login: Fri Apr 14 19:21:02 on ttys000
❯ cd Desktop
❯ touch helloworld.sh
❯ echo '#! /bin/bash' >> helloworld.sh
❯ echo "echo hello world" >> helloworld.sh
❯ ls -l helloworld.sh
-rw-r--r--@ 1 rashidwassan  staff  30 Apr 17 18:32 helloworld.sh
❯ chmod +x helloworld.sh
❯ ls -l helloworld.sh
-rwxr-xr-x@ 1 rashidwassan  staff  30 Apr 17 18:32 helloworld.sh
❯ ./helloworld.sh
hello world
 ~/Desktop
```

Useful Features of Bash

- **Metacharacters:**
  - # - prescedes a comment.
  - ; - command separator
  - * - filename
  - \ - escape special character interpretation
  - “ ” - interpreted literally but considers special characters
  - ‘ ‘ - interpreted literally
  - > - redirect output to a file
  - >> - appends the output to any existing content
  - 2> - redirect standard error to a file
  - 2>> - append standard error to a file
  - < - redirect file contents to a standard input
  - $(command) or `command`, can be used for command substitution like
    - var = $(command)

- **Batch vs Concurrent Modes**
  - **Batch Mode:** Commands are executed sequentially.
    command1; command2
  - **Concurrent Mode:** Commands are executed concurrently
    command1 & command2

Scheduling Jobs Using Cron