Software Delivery

DevOps

• Misunderstood as a hybrid role of developer and sys admin
• Should be perceived as a new way to develop and release software
  – Communication and collaboration between development, operations, quality assurance, product, and management

Developing the DevOps mindset

• In the past, different teams working on software lacked effective communications
  – Developers lacked the environment in which software is used
  – Operations received software to support but had no input in development
  – Led to fragile systems
  – Deployments are complex and error prone, leading to longer release cycles and creating even more risk
  – Systems even harder to maintain with each release

• Unplanned work
  – Resources may get pulled off of planned work
  – Project schedules are impact and due dates slip
  – This leads developers to take shortcuts, resulting in a lack of sound architecture, delaying nonfunctional requirements such as security and supportability, and other critical stability features, leading to even more issues
  – Quality, reliability, morale, and customer satisfaction degrade over time

• DevOps focusing on system thinking
  – CAMS – Culture, Automation, Measurement, Sharing
  – Build systems with a mind set that the needs of development, operations, and quality assurance are all related
  – Collaborative process
    * Developers, testers, and operations responsible for entire system
    * Every actor needs to understand each aspect of the system

• Four principles of DevOps
  1. Understand the flow of work
  2. Always seek to increase flow
  3. Don’t pass defects downstream
  4. Achieve a profound understanding of the system

• Influenced by lean manufacturing principles

• Maximize the flow of software creation from concept to development to release, with focus on six practices
  1. Automate infrastructure
    – Abstraction of infrastructure as an API allows infrastructure to be treated as code
    – Capability of scripting provisioning and deprovisioning of infrastructure leads to automating the creation of environments
    – Build code and environments at the same time
Every sprint with a complete set of code should include the corresponding environment
- User stories in sprint should include the necessary development operations and quality assurance requirements
- Separation of development, quality assurance, and operations required a lot of back and forth meetings
  - Leads to bottlenecks and environmental issues
  - Different development and operations environments introduce new problems
  - Finding bugs late in the life cycle leads to prioritizing those bugs
    - High priority bugs get fixed while others linger on
- Make sure that self-service infrastructure does not lead to chaos, inconsistent environments, non-optimized costs, and other bad side effects
  - Create standard set of machine images that can be requested on demand with appropriate access privilege
  - Ensure that the developers work with discipline and do not modify their environment to cause conflict
  - Apply patches at regular intervals to all the VMs

2. Automate deployments
- Code, configuration files, and environment scripts should share a single repository
- Decreases cycle times by removing the human error from deployment
- More frequent deployment leads to smaller change sets reducing the risk of failure

3. Design for feature flags
- Allow features to be configured by turning on or off
- If a feature has issues, it can be quickly configured to be turned off during deployment
  - Rest of the deployed features remain running in production
  - Gives team time to fix the issue and redeploy when convenient
- Allows features to be tested by a select group before rolling out to all users

4. Measure, monitor, experiment
- Leverage feature flags to run experiments to gather information about system and users
- Complexity of a new registration form
- Test a feature in a geographic area
- Test features in production against real production loads

Continuous integration and continuous delivery

- Continuous integration
  - Practice of building and testing applications on every check-in
  - Every big or small change gets checked in

- Continuous delivery
  - Adds automated testing and automated deployment to continuous integration
  - Testing performed throughout the life cycle rather than towards the end
    - Build process fails if any automated test fails
    - Prevents defects from being introduced into the build
  - Software always works and every change that is successfully integrated into the build becomes part of a release candidate

- Old model
  - Software was assumed to be incorrect until validated by dedicated quality assurance professionals
– Testing was its own phase performed after development
– Developers met deadlines by giving poor-quality code to testers
– Quality assurance cut corners to get the code to operations in time to release the software
– Allowed known bugs to flow into production systems

● DevOps model
– Software assumed to be correct unless automation tells otherwise
– Quality is everyone’s responsibility and testing is performed throughout the life cycle
– High level of communication and collaboration along with a sense of trust and ownership throughout the team

● Applicable to all development, cloud or otherwise