CSCI 5409 Cloud Computing – Fall, 2023
Week 9 – Lecture 2 (Nov 3, 2023)

DevOps and Release Management

Dr. Lu Yang
Faculty of Computer Science
Dalhousie University
luyang@dal.ca

Housekeeping and Feedback

- Start recording
- Start working on your term project.
- Try to ask TAs questions first. Remind me if I missed your messages.

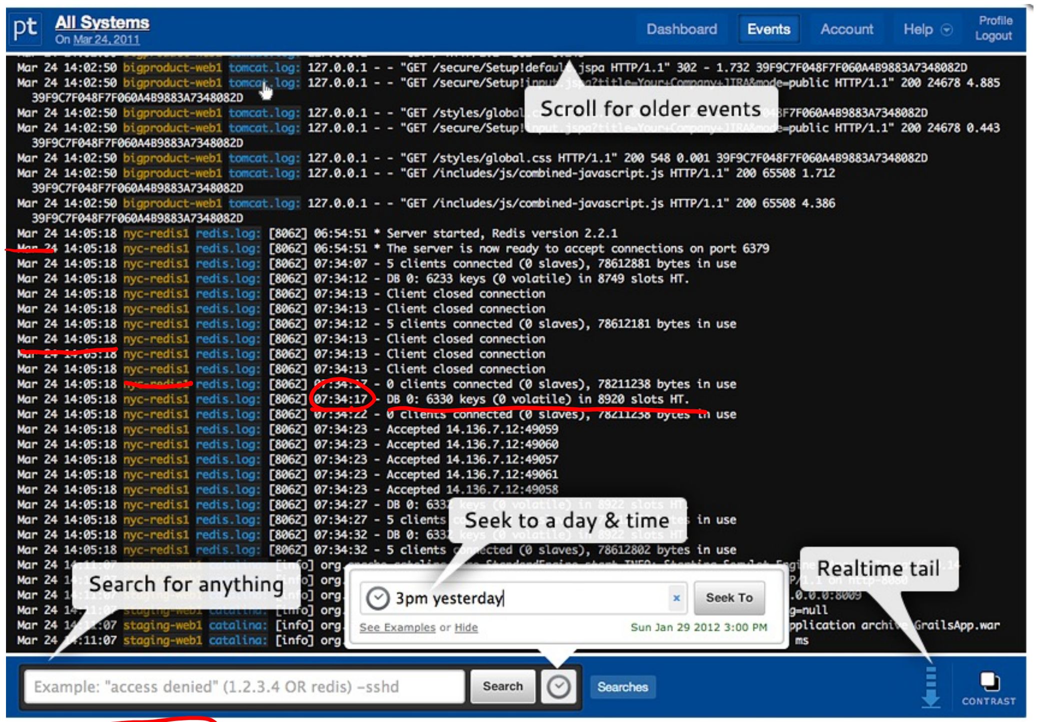
The Second Way: The Principles of Feedback

- "Enabling reciprocal and constant feedback from right to left to create an ever safer and more resilient system" [1]
- Just like catching a bug in an IDE with a stack trace, detecting and remediating problems while they are small and easier to fix prevents catastrophes
- **When failures occur, we treat them as learning opportunities as opposed to a cause for punishment and blame**
- "Our goal is to increase information flow in our system from as many areas as possible, sooner, faster, cheaper, and with as much clarity between cause and effect as possible. The more assumptions we can invalidate, the faster we can find and fix problems, increasing our resilience, agility and ability to learn and innovate." [1]
  - Achieved through the creation of feedback and feedforward loops in our system

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Pages 27 - 29.

Principles of Feedback (Continued)

- Telemetry:
  - "When all levels of our application stack have monitoring and logging, we enable other important capabilities, such as graphing and visualizing our metrics, anomaly detection, proactive alerting and escalation, etc." [1]
  - **Log everything**
  - **Add metrics for business logic events, DB ops, etc…**
  - Start with the most important things, then build to **everything**



```
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /secure/Setup!defaul  jspa HTTP/1.1" 302 - 1.732 39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /secure/Setup!input.jspa?title=Your+Company+JIRA&mode=public HTTP/1.1" 200 24678 4.885
  39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /styles/global                        F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /secure/Setup!input.jspa?title=Your+Company+JIRA&mode=public HTTP/1.1" 200 24678 0.443
  39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /styles/global.css HTTP/1.1" 200 548 0.001 39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /includes/js/combined-javascript.js HTTP/1.1" 200 65508 1.712
  39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:02:50 bigproduct-web1 tomcat.log: 127.0.0.1 - - "GET /includes/js/combined-javascript.js HTTP/1.1" 200 65508 4.386
  39F9C7F048F7F060A4B9883A7348082D
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 06:54:51 * Server started, Redis version 2.2.1
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 06:54:51 * The server is now ready to accept connections on port 6379
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:07 - 5 clients connected (0 slaves), 78612881 bytes in use
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:12 - DB 0: 6233 keys (0 volatile) in 8749 slots HT.
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:13 - Client closed connection
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:13 - Client closed connection
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:12 - 5 clients connected (0 slaves), 78612181 bytes in use
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:13 - Client closed connection
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:13 - Client closed connection
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:13 - Client closed connection
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:17 - 0 clients connected (0 slaves), 78211238 bytes in use
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:17 - DB 0: 6330 keys (0 volatile) in 8920 slots HT.
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:22 - 0 clients connected (0 slaves), 78211238 bytes in use
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:23 - Accepted 14.136.7.12:49059
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:23 - Accepted 14.136.7.12:49060
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:23 - Accepted 14.136.7.12:49057
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:23 - Accepted 14.136.7.12:49061
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:23 - Accepted 14.136.7.12:49058
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:27 - DB 0: 633  keys (0 volatile) in 8922 slots HT.
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:27 - 5 clients                                 in use
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:32 - DB 0: 633  keys (0 volatile) in 8922 slots HT.
Mar 24 14:05:18 nyc-redis1 redis.log: [8062] 07:34:32 - 5 clients connected (0 slaves), 78612802 bytes in use
Mar 24 14:11:07 staging-web1 catalina: [info] org
Mar 24 14:11:07 staging-web1 catalina: [info] org                              P/1.1 on http-8080
Mar 24 14:11:07 staging-web1 catalina: [info] org                              .0.0.0:8009
Mar 24 14:11:07 staging-web1 catalina: [info] org                              g=null
Mar 24 14:11:07 staging-web1 catalina: [info] org                              pplication archi   GrailsApp.war
Mar 24 14:11:07 staging-web1 catalina: [info] org                              ms
```

3pm yesterday

Sun Jan 29 2012 3:00 PM

Example: "access denied" (1.2.3.4 OR redis) -sshd

Papertrail – A high performance advanced logging system.
There are **many** tools and technologies in this area.
Image - https://www.getapp.com/it-management-software/a/papertrail/

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Page 198.

Principles of Feedback (Continued)

- **The Andon Cord: [1]**
  - Toyota manufacturing plant practice: every worker and manager is trained to pull the cord when something goes wrong; when the cord is pulled the team leader investigates, within a specified time if no solution is found the production line is halted so that the entire organization can be mobilized to assist.
- Swarming problems:
  - Prevents problems from progressing downstream where cost and effort to fix increases exponentially
  - Prevents introducing new work which may introduce new errors, obfuscating existing errors
  - If not addressed we could have the same problem next time, requiring more workarounds and wasted work (increasing lead time)
- Preventing the introduction of new work enables powerful continuous integration and deployment automation. Anything that passes our tests automatically goes out, anything that fails automatically pulls the Andon cord.

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Page 31.

The Third Way: The Principles of Continual Learning and Experimentation

- In organizations with systemic quality problems, work is typically rigidly defined and enforced
- "In these environments, there is also often a culture of fear and low trust, where workers who make mistakes are punished, and those who make suggestions or point out problems are viewed as whistle-blowers and troublemakers. When this occurs, leadership is actively suppressing, even punishing, learning and improvement, perpetuating quality problems."[1]
- "By removing blame, you remove fear; by removing fear, you enable honesty; and honesty enables prevention." [2]

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Page 37.
[2] Bethany Macri, engineer at Etsy and lead creator of the Morgue tool to assist with postmortems.

The Principles of Continual Learning and Experimentation (Continued) [1]

- We **MUST** reserve time for the improvement of daily work and to further accelerate and ensure learning:
  - **If we don't, technical debt accrues and cement shoes form**
  - Flip the script! Inform superiors that this work is part of being a professional programmer, it is **how** you create quality architectures and systems, don't let it be optional or seen as wasted work.
  - The more we do this, the more agile we become
- We consistently introduce stress into our systems to force continual improvement
- We even simulate and inject failures in our production services under controlled conditions to increase our resilience (e.g. Netflix "Chaos Monkey")

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Pages 37 – 38.

The Importance Of Organizational Culture [1]

Pathological: Characterized by large amounts of fear and threat. People hoard information, withhold it for political reasons, or distort it to make themselves look better. Failure is hidden.

Bureaucratic: Characterized by rules and processes, often to help individual departments maintain their "turf". Failure is processed through a system of judgement, resulting in punishment or justice and mercy.

Generative: Characterized by actively seeking and sharing information to better enable the organization to achieve its mission. Responsibilities are shared throughout. Failure results in reflection and genuine inquiry.

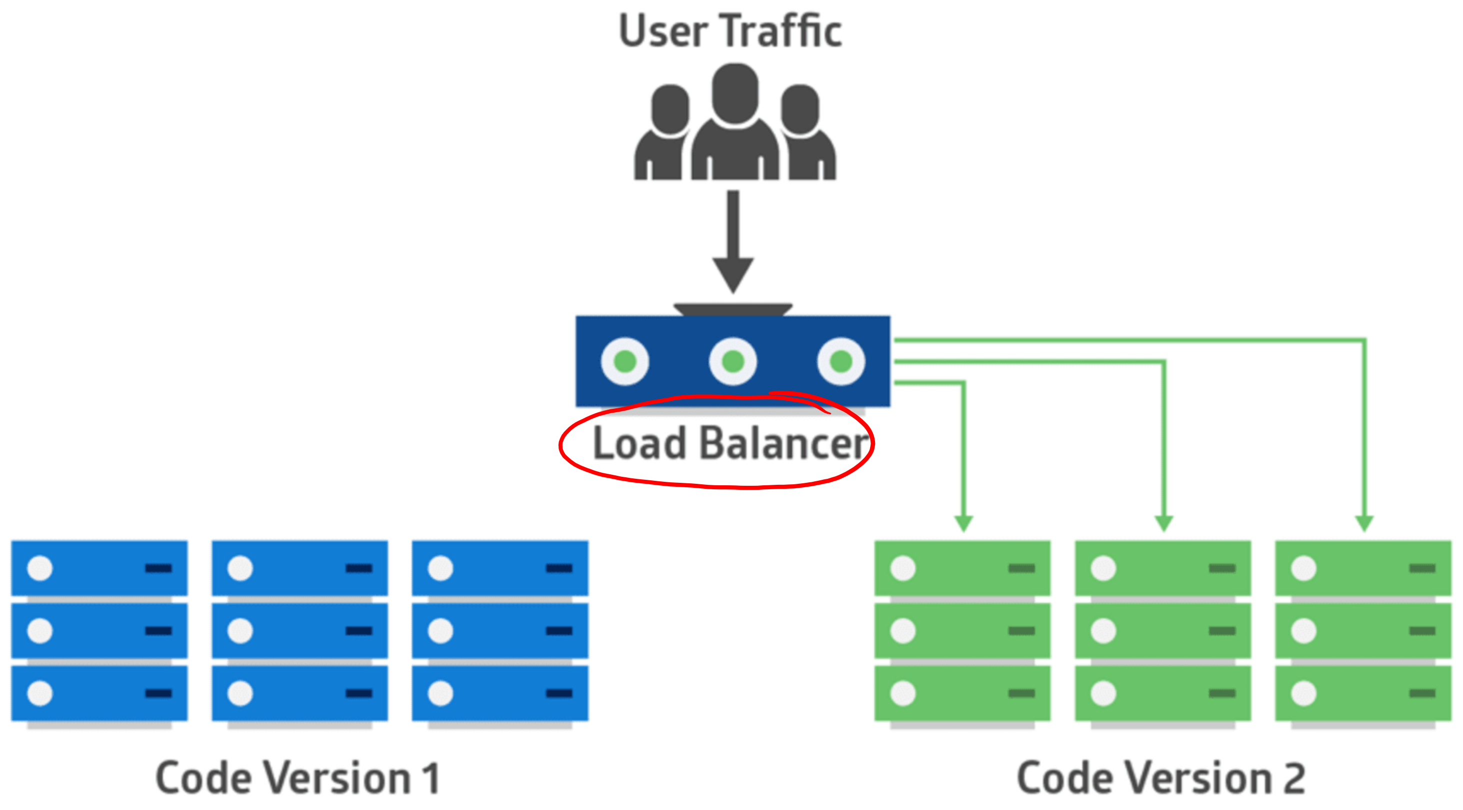
Pathological	Bureaucratic	Generative
Information is hidden	Information may be ignored	Information is actively sought
Messengers are "shot"	Messengers are tolerated	Messengers are **trained**
Responsibilities are shirked	Responsibilities are compartmented	Responsibilities are shared
Bridging between teams is discouraged	Bridging between teams is allowed but discouraged or blocked	Bridging between teams is **rewarded**
Failure is covered up	Organization is just and merciful	Failure causes inquiry
New ideas are crushed	New ideas create problems	New ideas are welcomed

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Page 39.

Release Management

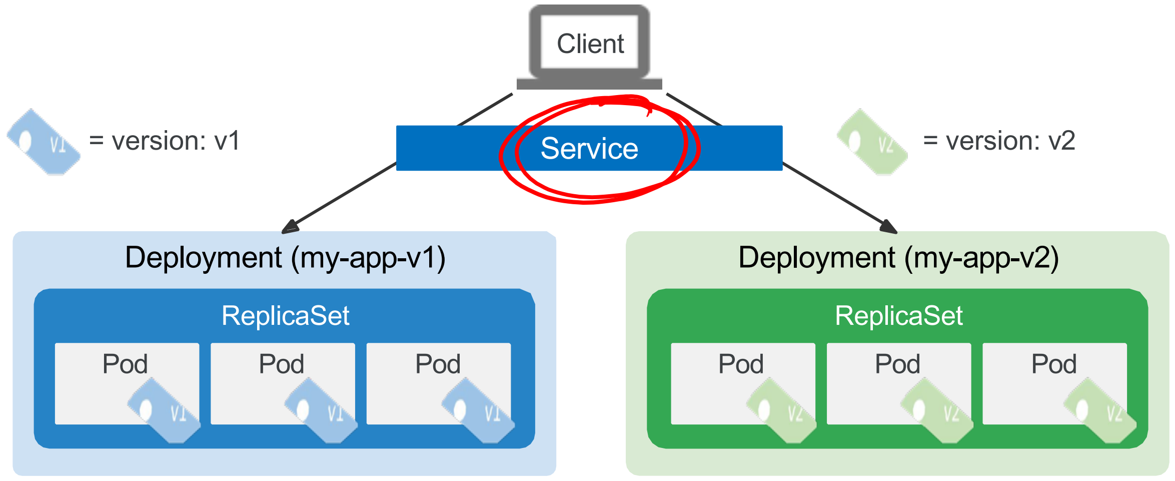
- **Ideally** everything about our release should be automated, a developer commits to version control and:
  - Continuous integration builds and runs unit tests
  - A production-like environment is automatically provisioned
  - Code is deployed
  - Automated tests do everything manual testers used to do
  - Automated info security / compliance checks on codebase
  - Code is deployed to production
- High performing organizations have the most automation, some releasing thousands of times per day
- This is extremely difficult to achieve, there is a chicken and egg problem here (how does an automated test cover code that was just written?)

The Blue/Green Deployment Pattern



Review at the blue/green deployment strategy in K8s

A blue/green deployment strategy ensures app services remain available



Microservices

Dealing With Database Changes

- Releasing code is complicated when it is tightly coupled to the database
  - If the schema changes the database cannot support both versions of the code
- Options for avoidance:
  - First, reduce coupling with mechanisms like stored procedures that hide schema, this solves most problems
  - Create two databases (i.e. blue/green database), at release time put blue into read-only mode, backup, restore onto the green and then apply DB changes
    - What happens if you need to rollback? This is not an easy problem to solve.
  - Decouple DB changes from application changes:
    - Make only additive changes to the DB
    - Use a version table, deprecate old tables, migrate to new ones
    - Make no assumptions in code about which DB version is in production
    - Con: Data is duplicated

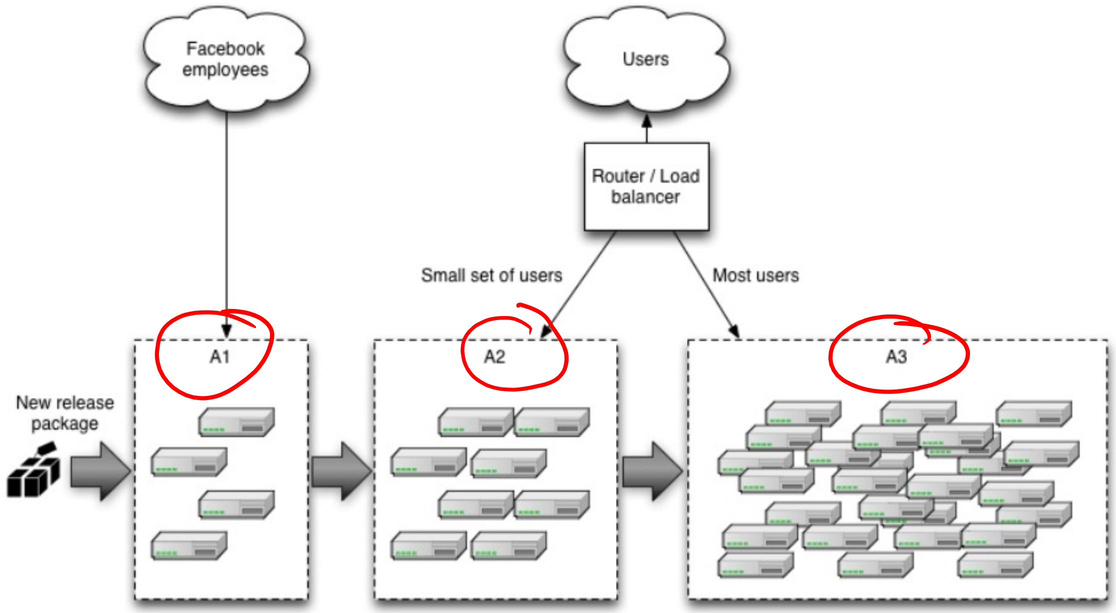
The Canary and Cluster Immune System Release Patterns

- Automates the release process by promoting code to successively larger and more critical environments as code quality is confirmed

The Facebook release pattern:

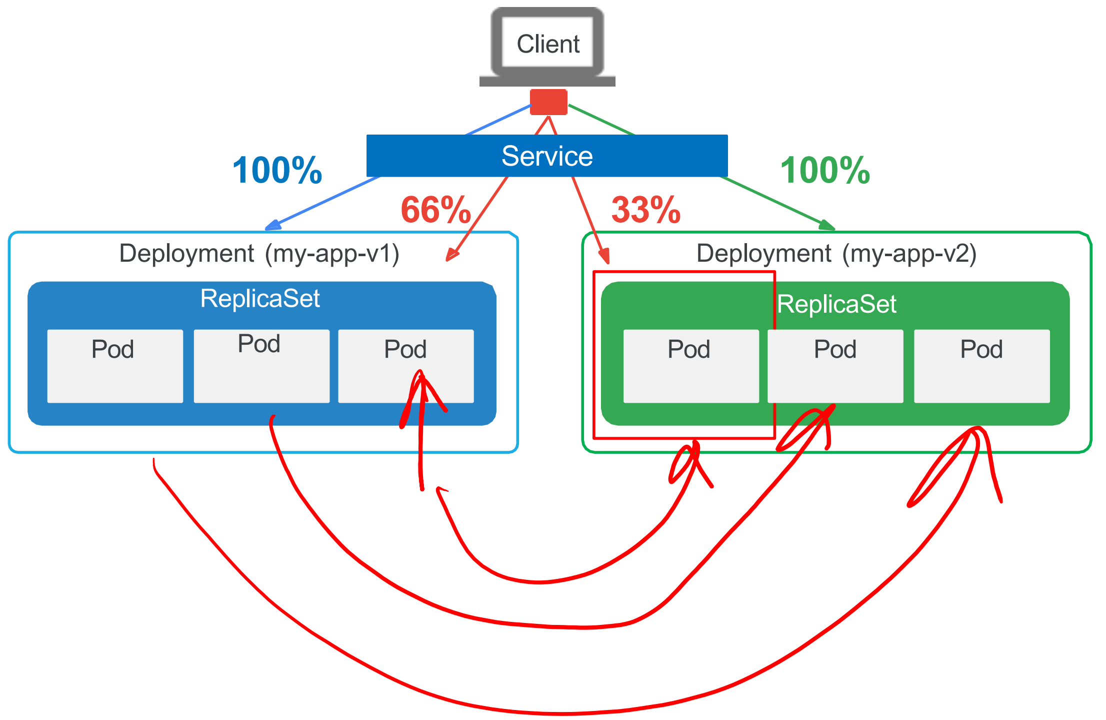
1. First code is released internally and exposed only to employees (A1)
2. Then the load balancer is configured to send a small set of users to a subset of servers running the new code (A2)
3. Then all users get the new code (A3)

Image - https://martinfowler.com/bliki/CanaryRelease.html



Review at the canary deployment strategy in K8s

Canary deployment is an update strategy where traffic is gradually shifted to the new version



Feature Toggles and Dark Launches

- Feature toggles are implemented by wrapping application logic or UI elements with a conditional statement, where the feature is enabled or disabled based on a configuration setting stored somewhere [1]
- Enables:
  - Easy rollback. If your feature breaks something or has defects, you just turn it off until fixed.
  - Gracefully degrading performance: If your service is experiencing high loads, turn off less valuable features
  - Increase resilience by enabling service-oriented architecture: You can release the feature even if the service isn't ready yet
  - **Dark Launches:**
    - Deploy features to production without making them accessible
    - Perform testing directly in production while it is still invisible to customers (potentially for weeks even)
    - Slowly enable it for some users
    - Finally enable it for all users

[1] Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. The DevOps handbook. Page 171.