Jacob Cannamela

CSD380

6/28/2025

**Module 6 Assignment**

In Chapter 13 of The DevOps Handbook, the author covers something that sounds very technical, “Architecting for Low-Risk Releases” but to me it makes sense. It is not only deploying faster or writing cleaner code, it is about reducing the chances of things blowing up in production and making the whole release process smoother. I think one of the most important concepts that stood out to me was with respect to old school architectures and tightly coupled systems and how they hinder an organization from doing frequent or safe deployments. This chapter largely makes the case that when your system consists of everything depending on everything else, changing one small thing can potentially break ten other things. Reminds me of so many projects I've worked where we didn’t want to touch anything because we had no clue what would break.

To fix this issue, the book stresses creating independently deployable systems which are loosely coupled. Microservices get mentioned a lot here (not surprising), but it’s not pushing a one size fits all approach. The focus is more towards helping developers be able to operate at speed without needing some sort of full team call for every change they wish to make after a change is deployed. I admired how they linked the discussion on architecture to culture and process. This is not an issue of writing different code, it’s about implementing feature toggles, blue/green deployments, and canary releases. These methods allow for changes to go live gradually or beneath the surface. This allows teams to monitor a real-time impact and if necessary, back out without rolling back a full deployment.

Another significant lesson learned was around making failures safe and reversible. It emphasized the need for systems capable of safely managing temporary lapses in productivity while maintaining swift self-recovery with minimal supervision. That’s terrifying at first glance but when you think about it, if there are mechanisms to catch issues early enough just like air traffic controllers in airports managing low visibility landings, one can maintain a limited soft-landing approach without constant chaos or alarm. In this case, for me, it was less about technical tools and more around designing systems that can handle changes. Starting from assuming the system will not fall apart when change is introduced up to trusting a fast recovery mechanism if things fail and honestly trusting leadership does not lurk in corners fuming, ready to blame or point the finger after failures occur.

Summing up, Chapter 13 seemed to me like the connection of theory with practical work. It is the section in DevOps which ensures that your architecture does not undermine all of that progress. This caused me to reflect more deeply on the systems I deal with and consider how much technological debt alongside tight coupling is stifling our ability with respect to working in a safer, faster, better manner.

**References:**

*DevOps Certifications -* (n.d.). International DevOps Academy. https://www.devops-certification.org/How\_Do\_You\_Enable\_Low\_Risk\_DevOps\_Code\_Deployments\_In\_Your\_Production.php

LaunchDarkly. (2025b, June 27). *5 Strategies to De-Risk Software Releases in Financial Services | LaunchDarkly*. LaunchDarkly. https://launchdarkly.com/blog/5-strategies-de-risk-releases-financial-services/

*Low-risk releases: 5 steps to building a leading DevOps practice*. (n.d.). https://www.pluralsight.com/resources/blog/business-and-leadership/5-steps-leading-devops