CTO Sunday Thoughts #8

GRC Updates: The Risky Business of "Later"

It’s Sunday. A perfect day to think about the things we put off. In our personal lives, it might be cleaning the garage. In IT, it's often GRC.

Last Sunday I did focus on Instant upgrades made me think on next level of this topic.

Do I need to be on the newest version? Or do we still embrace ‘don’t rock the boat’?

We spend months designing a perfect, secure, and compliant Azure foundation. We build it, get the "pass" from the auditors, and breathe a sigh of relief. The problem is that the cloud is **evergreen**. It doesn't wait for your next audit cycle. Microsoft pushes updates, new services, and new security recommendations *daily*.

Your "perfect" GRC foundation is obsolete the moment you finish building it. And every day you wait to update it, the **"governance deficit"** grows. This is the "silent drift" I've talked about before - a slow, quiet decay of your security posture.

So, if continuous updates are the answer, why doesn't everyone do it?

It comes down to fear.

Patch Tuesday vs. The Evergreen Cloud

In the on-prem world, we lived by **Patch Tuesday**. It was a predictable, if painful, monthly ritual. We'd batch up all the changes, test them in a staging environment for a week, and then hold our breath during a "big bang" production deployment.

This model is fundamentally broken in the cloud.

The pace of change is too fast. A new zero-day vulnerability doesn't wait for the second Tuesday of the month. Relying on a monthly or quarterly GRC update process is like leaving your front door unlocked because you only check the locks on the first of the month.

This creates a core tension:

> **The Need for Speed:** Security teams want updates pushed *instantly* to close vulnerabilities.
>
> **The Need for Stability:** Application owners fear that any change—even a security policy—could break their production workload.

This fear leads to a desire for "control," which usually manifests as an **opt-in** model. "Don't update my environment until I say I'm ready."

This sounds responsible, but in reality, it's the most dangerous option.

The Paralyzing Effect of "Opt-In"

An "opt-in" model, where you must actively choose to apply updates, makes "later" the default.

That "I'll get to it next week" becomes next month, and then next quarter. The GRC update backlog becomes a form of **configuration debt**. It grows so large and complex that applying it becomes a massive, high-risk project that no one wants to touch. Your organization is now running on an insecure, decaying foundation, all in the name of "stability."

A modern GRC service *must* flip the script. The default state must be **secure, compliant, and up-to-date**.

This is the advantage of an **opt-out** model. You get all the updates, all the time, *unless* you have a specific, justifiable, and *temporary* reason not to. It makes security the path of least resistance. The burden is no longer on the team to find time to *apply* security; the burden is on them to justify *not* being secure.

How We Solved the Update Dilemma

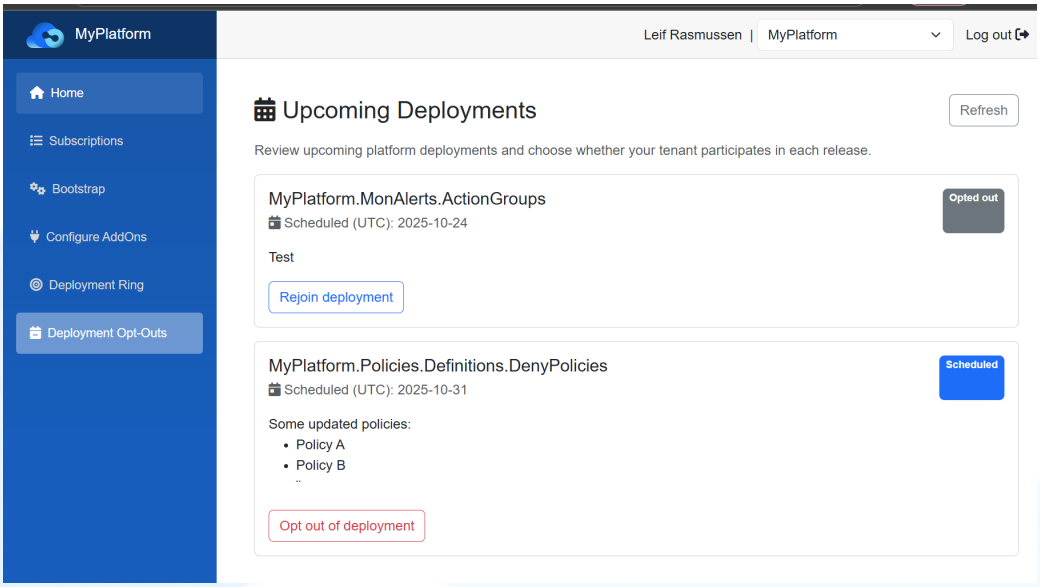
At **MyPlatform**, we had to solve this exact problem. Our promise is an **evergreen GRC platform**, but we have to deliver it without causing the very disruption our customers fear.

We've done it by combining two key concepts:

Deployment Rings (The Safety Net): We don't just push new code to every customer at once. As I've mentioned before, we use deployment rings. Our own internal tenants run on the "Instant" ring, getting every change immediately. Only after code is battle-tested there do we *manually* promote it to the "Stable" ring that our customers use. This isn't just automated testing; it's proven-in-production validation.

The "Opt-Out" Valve (The Control): We've just built a new "ControlCenter" for our platform. Here's how it works: We notify customers of an upcoming deployment to the "Stable" ring. By default, they are scheduled to receive it. However, if a customer is in the middle of a critical production freeze, they can go into their admin portal and request to "opt-out" of that *specific* deployment.

This is the critical part: it's a **snooze button, not an off switch**.



They are only skipping that single deployment stack. The *next* time a deployment runs, it will include all the changes from the one they skipped. This gives customers the control they need to manage critical change windows but makes it impossible for them to accumulate the massive, long-term governance debt that leaves them at risk.

Continuous GRC isn't just about automation; it's about building a trusted, reliable operational model. It's about giving customers a safety net *and* a safety valve, so they can stay evergreen without the fear of breaking what's most important.

"How does your team balance the need for stability with the risk of governance drift? Let's discuss how to shift from 'opt-in' to 'opt-out' without the fear."