GRC Updates instantly?

It's Sunday. A good day for reflection! It reminds me that in cloud architecture, Governance, Risk, and Compliance (GRC) form a crucial foundation that is often underestimated.

For years, we've treated GRC as a static blueprint. We'd spend months with consultants, draft extensive documentation, and manually configure hundreds of settings and policies. We'd build the perfect, compliant house, take a step back, and hope for the best.

But the **cloud is not a static environment**; it's a perpetual motion machine. Microsoft Azure is "evergreen," meaning it's constantly evolving with new services, features, and security updates. A GRC framework that is perfect today is clearly obsolete in 18 months if left untouched.

This constant change creates a problem I call the **governance deficit** — a slow, insidious accumulation of risk and a decay of control. What started as strict, well-defined guardrails slowly erodes into a set of mere suggestions. A temporary exemption is made for a project, a manual change is pushed through the portal to meet a deadline, and slowly, your compliant state decays[4]. Your security posture weakens, not from a dramatic attack, but from a thousand small, undocumented deviations that create a massive backlog of security gaps and configuration debt. This is where automation isn't just a "nice-to-have"; it's a fundamental necessity.

New Standards: From Manual Blueprints to Living Code

The industry standard is shifting beneath our feet. The idea of a 200-page Word document defining your GRC is outdated. Today, the best practice is to define “everything as code”, as defined by frameworks like the **Microsoft Cloud Adoption Framework (CAF).** Your governance isn't a policy document; it's a set of policy definitions, infrastructure templates, and configurations stored in a version-controlled repository.

This code-first approach transforms GRC from a one-time project into a living, breathing part of your operations. It becomes **transparent**, **repeatable**, and, most importantly, **updatable**. When a new threat emerges or a compliance standard evolves, you don't schedule a series of manual changes across dozens of subscriptions. You **update the code, test it, and deploy it systematically**. This is the only sustainable way to keep pace with the cloud and pay down the governance deficit.

However, delivering these updates is where the real challenge lies. How do you roll out a critical change to your entire Azure environment without breaking something? This brings us to a critical point: the **anatomy of a failed update**.

CTO Sunday Thoughts #7

When the Safety Net Breaks

Imagine pushing out an updated security policy that, due to an unforeseen interaction, accidentally revokes access for a critical application's managed identity. Suddenly, your production environment grinds to a halt. The frantic scramble to identify the cause, the emergency rollbacks, the late-night calls — it's a scenario we've all feared.

A failed GRC update can have catastrophic consequences:

Security Vulnerabilities A poorly tested policy might inadvertently open a security hole, creating an exposure that is far worse than the one it was intended to fix.

Operational Disruption An update can break production workloads, leading to downtime, lost revenue, and a loss of trust from your users.

Compliance Breaches Ironically, an update meant to enhance compliance can, if it fails, leave you in a non-compliant state, exposed to audit failures and potential fines.

The risk of failure is precisely why many organizations hesitate to update their GRC framework, allowing the governance deficit to grow. They choose the slow decay they know over the potential for acute disaster. But what if we could deliver continuous updates while rigorously managing the risk?

Our Approach: Deployment Rings for GRC

This is a challenge we've spent a great deal of time solving at **MyPlatform**. If you're going to provide an "evergreen," always-updated GRC framework as a service, you must have an exceptionally robust and safe way of delivering those updates. The answer lies in a concept used by hyperscalers like Microsoft itself: **deployment rings**.

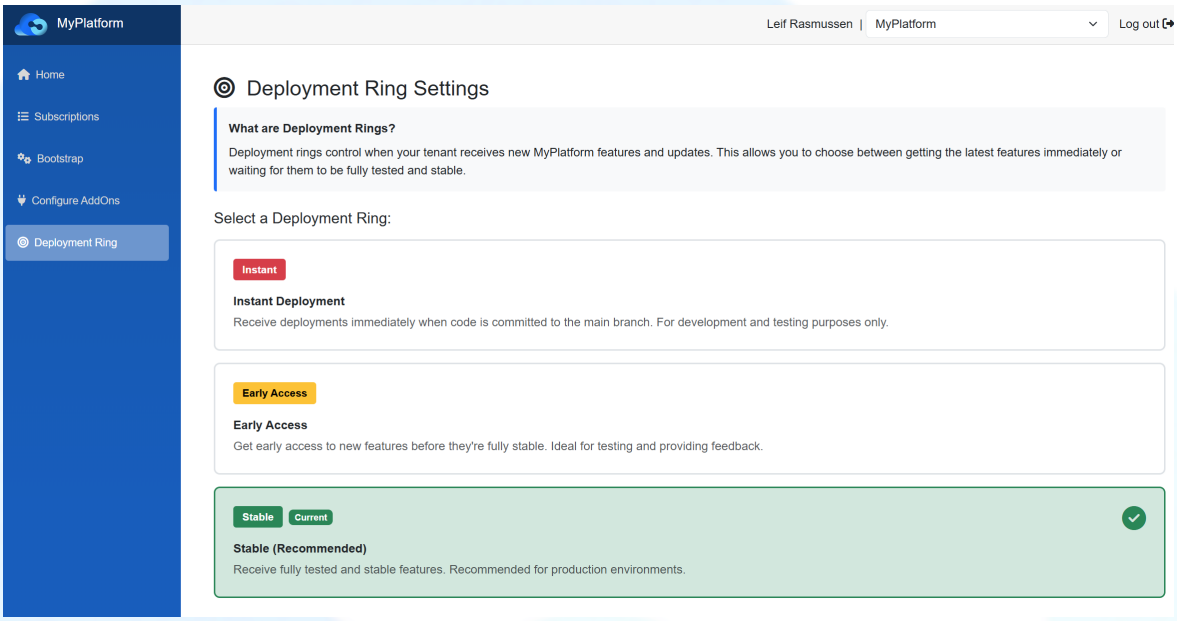
Instead of a single "big bang" release, we roll out changes in controlled stages. We've just implemented this model, which allows us to manage how and when updates are deployed to our customers' tenants. It works like this:

The "Instant" Ring - When our team merges a new feature or policy update into our main branch, it automatically deploys to a set of tenants designated as "Instant." These are typically our own internal environments and those of customers who have opted-in to get the latest updates immediately. This ring acts as our first line of real-world validation.

The "Stable" Ring - This is the default for the vast majority of our customers. Updates are only pushed to this ring after they have been thoroughly vetted in the "Instant" ring. The deployment to the "Stable" ring is a deliberate, manual process. We'll merge the now-tested code from main to our release branch and run the pipeline on a planned schedule.

This ensures that by the time an update reaches a production environment, it's not just code that passed automated tests, but code that has proven itself in live, operational tenants.

This ring-based deployment model provides a crucial safety buffer. It allows us to innovate and respond to new threats rapidly in the "Instant" ring while providing the assurance of rock-solid stability in the "Stable" ring. It's a methodical, risk-managed approach that turns the terrifying prospect of a GRC update into a controlled, predictable, and routine process.



The future of cloud is not just about adopting new services. It's about maturing our operational models to manage the inherent complexity of this evergreen world. Automated, code-driven GRC is the baseline, but a safe, structured deployment strategy is what separates sustainable success from an ever-growing governance deficit. **It's about having an autopilot that is constantly learning, supported by a safety net that you can always trust.**

And that's a thought worth having this Sunday.