Modern Development Practices in Banking

A PLAYBOOK

Flagsmith

2024

Contents

INTRODUCTION

Moving to Modern Software Development for Banks

INSIGHTS FROM ROMANO ROTH

Romano Roth
Chief of DevOps and Partner at Zühlke

zühlke

Welcome to the "Modern Banking eBook", a comprehensive guide for banking institutions embarking on the crucial journey of modernising their software development processes. I'm Romano Roth, and it's my privilege to walk you through the transformative steps that can reshape the way banks operate in the digital age.

In today's fiercely competitive banking landscape, the impetus for modernisation is stronger than ever. This drive is fueled by the need for faster time to market, enhanced value for money, and—paramountly—elevated customer satisfaction. These pillars are foundational to thriving in an environment where customer experience is not just a metric but the heart of differentiation. The modern banking ecosystem demands agility, innovation, and a relentless focus on delivering exceptional user experiences.

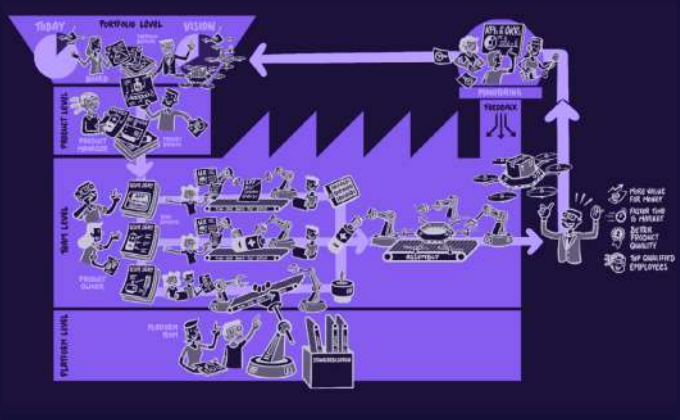
But what prompts banks to initiate this modernisation? The quest to deliver products to market more swiftly, efficiently, and with tailored customer experiences at the forefront is the primary catalyst. Modern software development practices, such as continuous deployment, feature flags, and test-driven development, are not merely technical upgrades; they are strategic enablers. These practices enable banks to deploy small, low-risk changes, release features on demand to specific locations or subsets of users, and conduct A/B testing. This approach not only significantly improves the developer experience but also accelerates the release cycle.

However, this journey is not devoid of challenges. Banks face hurdles like stringent compliance and regulatory requirements, traditionally longer development cycles, and the inertia of legacy systems. Overcoming these obstacles requires a thoughtful approach, starting with fostering a culture of modernisation, embracing cloud and hybrid infrastructures, and adopting practices like DevSecOps to shift left on quality assurance and security.

The first steps towards modernisation involve embracing strategies that allow for the decoupling of application landscapes and moving towards a composable and event-driven architecture. This approach,

underpinned by APIs and a separation from core banking systems, paves the way for a more agile, responsive banking environment.

The benefits of embarking on this modernisation journey are profound. Teams will experience heightened efficiency, a more collaborative development culture, and the ability to deliver innovative solutions that directly enhance the customer experience. Moreover, the strategic implementation of an Internal Developer Portal (IDP) and platform engineering principles will be instrumental. These elements serve as the bedrock for a banking institution's transformation into a digital factory, where everything with customer exposure is custom-built on a platform designed to enable and empower teams.



Looking ahead, the future of banking processes is bright with the promise of AI, multi-cloud strategies, distributed teams, and a shift towards thinking in products and value streams. The goal is a banking sector that is not just adaptable and efficient, but also acutely attuned to the evolving demands of the market and its customers.

This eBook lays out a roadmap for modernising banking software development, covering critical areas such as feature flags, observability, CI/CD, authorisation, and platform management. Through the lens of a case study on Komerční Banka and discussions on open source and modernisation strategies, we aim to provide you with the knowledge and tools to transform your banking operations.

Everything that has customer exposure is a differentiator and must be custom-built. To achieve this, banks need a platform that enables their teams to build it. In essence, banks are becoming digital factories—innovative, efficient, and customer-centric. Let us embark on this journey together, laying the foundations for a future where banks are not just financial institutions but digital powerhouses that drive economic growth and customer satisfaction.

ABOUT ROMANO

I'm Romano Roth, Chief of DevOps and Partner at Zühlke. My journey with Zühlke began 21 years ago. Over the years, I've evolved from an expert software engineer and software architect to a consultant. Throughout this journey, one question has always fuelled my passion: How can we continuously deliver value while ensuring quality and automation?

When the DevOps movement began to gain momentum, I was naturally drawn to it. Today, I'm one of the organisers of the monthly DevOps Meetup in Zürich and president of DevOps Days Zürich, an annual conference that's part of the global DevOps movement. DevOps isn't just a professional interest; it's my passion. That's why I have my own YouTube channel, where I've curated over 100 videos centred on DevOps, architecture, and leadership.

ABOUT ZÜHLKE

Zühlke is an innovation powerhouse specialising in turning visionary ideas into reality through product and software engineering as well as management consulting. With 50+ years of experience and a global footprint, we drive digital transformation in the finance, healthcare, industrial, and consumer goods sectors. Our approach emphasises excellence, agility, and collaboration, making us a catalyst for sustainable growth and innovation. As a leader in the DevOps community, Zühlke stands at the forefront of pioneering practices that significantly boost operational excellence and efficiency. We transform organisations for a sustainable future.

5 Tools Banks Can Adopt to Modernise Development

CHAPTER 1

Banking and Modern Observability

DYNATRACE INSIGHTS

Andreas Grabner
DevOps Activist at Dynatrace and CNCF Ambassador

dynatrace

CLOUD NATIVE COMPUTING FOUNDATION

INTRODUCTION

How Observability is Changing

Observability has been around for as long as software engineering has been around. But it has clearly changed over the years!

If we look back 15 years, the classic three-tier application stack (web server, app server, database) with application runtimes such as Java and .NET resulted in a new area of observability tools called Application Performance Monitoring (APM).

Those APM tools focused on monitoring rather static infrastructure and the dynamic behaviour of the application runtimes and business transactions. Financial organisations who connected those three-tier applications with their backend mainframe systems also started leveraging APM as some vendors expanded their support into z/OS, CICS or IMS to monitor the behaviour of end-to-end financial transactions.

As changes to those applications only happened occasionally, detecting problems and incidents got easier by analysing all the collected metrics, logs and distributed traces.

Fast forward to today: Infrastructure is no longer static. It runs on a multi-layer virtualised stack on VMs, Container Orchestration Platforms (such as Kubernetes or OpenShift), or even in Serverless PaaS (Platform as a Service) offerings distributed in global data centres. Applications are no longer three-tier, but rather a mesh of services spanning self-developed cloud services to any type of SaaS-hosted third-party services.

Why It's Time for a New Approach to Observability for Banks

Changes in applications (releasing new features) and adaptations to virtual infrastructure to change load or resource consumption behaviour happen continuously. Deployments are now decoupled from releases by applying progressive delivery techniques such as canary deployments or feature flagging, which reduce the risk of failure—especially for critical banking systems that power the globally connected financial transaction systems.

Services are becoming increasingly complex, though, and the shift towards dynamically releasing new features also means that the chances of becoming vulnerable to security problems and attacks are rising alarmingly!

This explosion of complexity (and the need to provide resilient and secure services) requires a new approach to observability. This new approach needs to be able to cope with constantly changing distributed virtual and cloud-based infrastructure. It also needs to handle continuously changing services and configurations, and the increased need to observe and mitigate any potential security vulnerabilities.

Without adopting modern observability practices, many organisations will not be able to:

- Identify and fix problems impacting end-user experience resulting in lost business
- Identify resource usage inefficiencies resulting in higher operational costs
- Identify and mitigate security threats jeopardising your business's integrity
- Identify and roll back progressive delivery changes threatening your systems' stability

What are the Hurdles to Modernising for Banks?

Like most industries, banks and financial institutions need to modernise their technology stack and software delivery processes for multiple reasons, including:

- Delivering modern and competitive digital experiences for end users
- Speeding up feature delivery to react faster to market changes
- Attracting new software engineering talent
- Providing and consuming APIs to connect with the larger financial ecosystem

Any type of change to an existing system comes with a certain risk, though, and there is understandable hesitancy to change at the organisational level. For example, the new cloud-native environments are complex, and microservice architectures and dependencies to third-party API providers are a blocker.

An adapted and modern approach to observability is still the way forward though. With modern observability, you can better understand the potential risk, impact and root cause of changes.

Shifting Left to Modern Observability

Traditionally, observability was implemented in production, providing operations teams insights into an unknown system (black box) to improve and identify potential problem areas in case issues arise.

With the increased complexity of modern cloud-native environments, observability must no longer be seen as an afterthought. Observability must be included as a software development requirement—hence the term "Shift-Left".

Shift-left means that developers (who know the critical components and code best) must define what level of observability they need to determine whether their software runs as expected. This can be done through agent-based observability solutions that automatically instrument code, or through developers leveraging standard frameworks like OpenTelemetry to emit traces, metrics or logs directly from their custom code.

A shift-left approach also means that observability data must be collected and analysed in every stage of your software delivery pipeline. Software quality gates must include checks on whether expected observability data is collected successfully and whether systems are behaving as expected. The relevant observability data must also automatically be forwarded to the right people and tools to make automated decisions in case of any anomalies.

How to Shift to Modern Observability

Some First Steps

Observability must now be a primary focus. It can no longer be seen as a cost centre or an afterthought. So how can you shift the approach to observability in your teams' processes?

First, it's important to recognise that observability is a business differentiator and has to become part of the software development process by adding it as a non-functional requirement to every newly created software component.

Some best practices for doing this include educating developers on the available observability signals (logs, metrics, traces, events, etc.) and the usage of modern observability frameworks (such as OpenTelemetry). This can be achieved in training sessions, development processes, and in shifting the culture and team mindset towards development.

Secondly, observability platforms that collect all observable data should be made available as self-service from the first developer environment up to production with minimal or no effort to access and analyse the data.

Additionally, when software gets tested as part of the software development process, observability data should also be validated as part of the software delivery process. Modern observability platforms integrate well with CI/CD solutions so you can validate that all logs, metrics, traces, and so on are captured and analysed the way the developers have coded or configured them. This ensures that all relevant observability data is available in production to ensure healthy operations or to support with troubleshooting of unexpected problems.

Many observability tools will have experts on hand who can work with you to run an assessment of your current development processes and help implement changes, too.

Observability in the Context of Progressive Delivery

How Can You Modernise With Observability and Feature Flags?

Progressive delivery—the technique used to decouple deployments from releases—is a key component in keeping your systems reliable, resilient and secure. Feature flags have become a very popular aspect of this in the last few years. The use cases range from deploying new end-user features to supporting Operations teams with auto-remediating tasks by dynamically changing the code behaviour without having to redeploy.

Using feature flags only works successfully when making the use and impact of those flags observable. Therefore, it's necessary to make feature flags observable by default.

Modern feature flagging and modern observability solutions can work together to provide out-of-the-box insights into things like:

- Which feature flags are used by which end user
- How the feature flags impact the users' experience
- Whether a new feature negatively impacts the underlying dynamic infrastructure

For compliance reasons, observability tools also keep track of when features were enabled and disabled to provide auditable tracing in case a feature had a non-desired impact.

What's more, modern observability can be applied to the feature flagging solutions themselves to ensure they are always able to deliver the right feature flagging configuration to the business app that is using them. Feature flagging tools and their backends are becoming critical system components and they have to be as resilient and available as all other components.

(This is where the Flagsmith-Dynatrace integration comes into play, letting you send flag change events from Flagsmith into your Dynatrace event stream.)

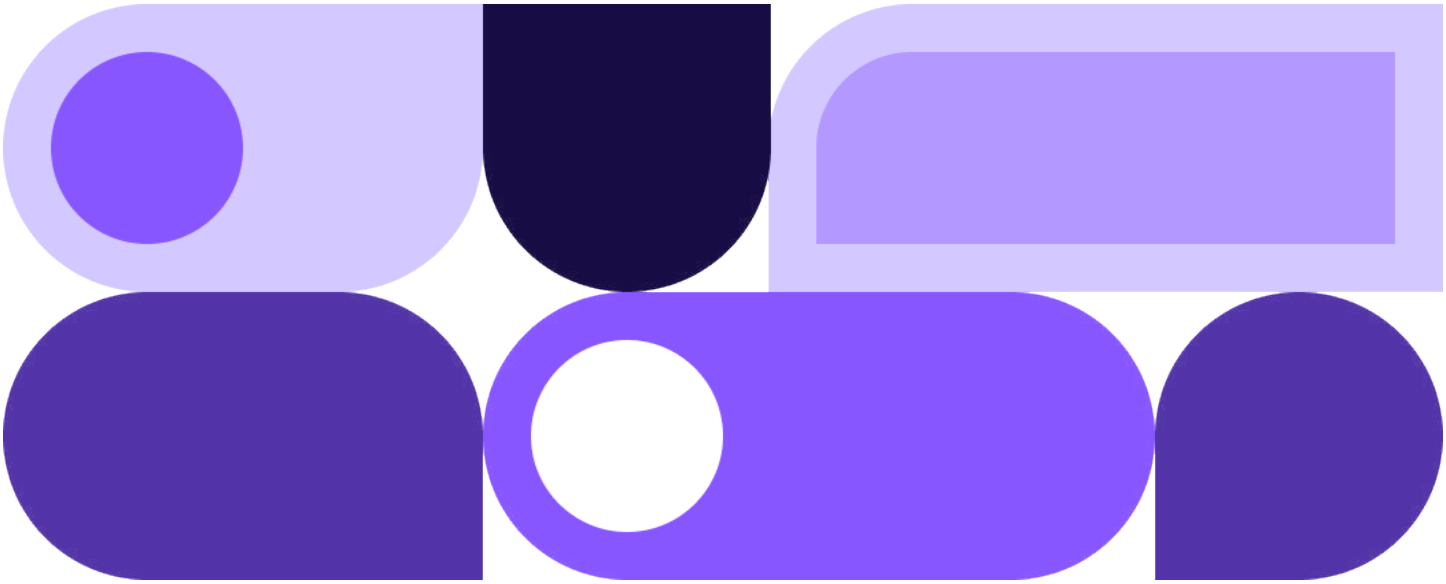
CONCLUSION

Modern Observability Leads to More Secure Development for Banks

Banking organisations that implement modern observability can make it part of their software delivery requirements and enable developers to think about observability right from the start. Those organisations end up delivering innovation faster, more reliably and more securely.

ABOUT DYNATRACE

Dynatrace has been a leader in modern observability and security for the past decade. Learn more about how Dynatrace can help your organisation innovate faster: https://www.dynatrace.com/



CHAPTER 2

Unlocking Efficiency

TRANSITIONING TO MODERN CI PLATFORMS

Tomas Fernandez
Technical Writer at Semaphore

semaphore

In the fast-paced world of technology, staying anchored to outdated continuous integration (CI) platforms can hinder an organisation's progress and stifle its growth.

Continuous integration is the beating heart of any software project. A poor CI experience impacts every aspect of the development process, from testing to deployment.

Legacy CI systems, once reliable, now struggle to keep pace with modern requirements, leading to frustrated developers and scalability bottlenecks. It's time for a change. It's time to transition to modern CI platforms that offer automated scalability, cost savings, and an improved developer experience. In this chapter, we delve into the challenges of legacy CI platforms and explore why organisations should embark on a journey toward modernisation.

Challenges of Legacy CI Platforms

Is your organisation hindered by legacy CI platforms? Have your engineers toiled to keep your legacy platform running? You're not alone. A CI platform is easy to take for granted until it starts holding your development team back. While your CI platform may have served you well in the past, it often struggles to keep pace with modern requirements, presenting challenges that can hurt productivity and growth.

Let's look at some common challenges legacy CI platforms present.

High Costs

One of the trickiest issues with legacy CI platforms is their hidden operating costs. While the product itself might be open-source and free, its maintenance is a major expense. Often, it requires dedicated personnel to manage server upkeep, conduct routine system checks, and implement periodic plugin updates. However, the costs don't stop there. During upgrades, these systems can experience significant downtime, halting progress and adding indirect costs from lost productivity.

To make matters worse, as organisations grow, so does their need for system resources. Legacy CI platforms may require costly hardware upgrades to keep pace with this growth.

Moreover, the onboarding process for these platforms is usually lengthy and complex. Training new team members can be time-consuming, adding to the financial strain.

Subpar Scalability

Scalability is a crucial factor in any CI platform. As organisations expand, products become more complex, their codebases grow, and their teams develop, the CI system must be capable of keeping pace. Legacy platforms, however, often falter when faced with scaling requirements.

When a CI platform fails to scale, developers are forced to wait for system resources, leading to bottlenecks that slow development and frustrate teams. This drains the team's vitality, making progress and innovation more challenging.

Poor Developer Experience

Speaking of vitality, legacy CI platforms are notorious for their complexity. They hail from a different era when the importance of a simple and streamlined interface wasn't yet a priority. This complexity can be off-putting to developers accustomed to more modern, user-friendly tools.

Legacy CI systems often require manual configuration and maintenance—tedious, mind-numbing work that diverts developers from their core work: creating code and delivering value to the business.

Limited scalability and general CI slowness compound the issue, leading to slower feedback loops. Slow feedback loops disrupt developers' train of thought because they either wait for the CI platform to finish its task or switch to another task and return later to check if their work has passed all the tests.

Security Vulnerabilities

As with any software system, security is a primary concern. However, older platforms are often riddled with vulnerabilities. Outdated software versions and unmaintained plugins can be exploited by malicious entities, exposing the organisation to potential breaches, data loss, and liabilities.

Keeping the system secure and compliant requires continuous efforts, adding another layer of complexity and workload for the maintenance team. This necessitates that organisations stay alert to the latest security threats and updates.

Lack of Support

Support is crucial to any software platform, and CI systems are no exception. When problems arise, having a support team standing by your side can be the difference between a minor hiccup and a significant delay.

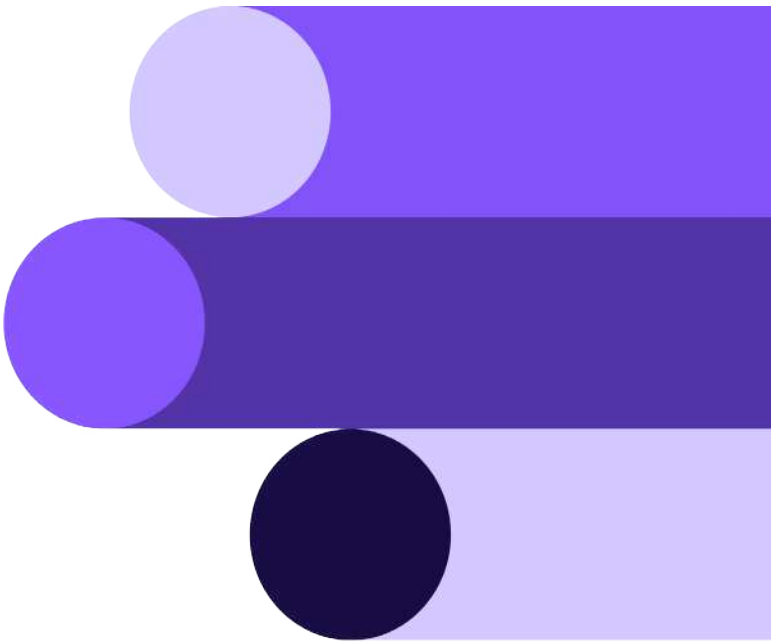
Legacy CI platforms often lack robust support systems. Without 24/7 support or service level agreements (SLAs), problems can seriously disrupt developers' work.

Lower Performance

Performance is another area where legacy CI platforms often fall short. The performance is limited to the power of the machine running the CI. Even powerful servers can be overwhelmed by the testing needs of a modest project, slowing the entire development process and negatively impacting delivery timelines.

Complex Configuration and Onboarding

The complexity of legacy CI platforms isn't limited to their interfaces. These platforms often require complex configurations for setup and management. Such a complex design discourages collaboration—when team members find a system difficult to use, they're less likely to fully engage with it.



The Need for Transitioning to Modern CI

Given all the challenges legacy CI platforms pose, the need for a transition to more modern solutions becomes apparent. Modern CI platforms, such as Semaphore CI/CD, are designed to address these pain points, offering features and capabilities that can significantly improve efficiency, productivity, and developer happiness.

Automated Scalability

Modern CI platforms are built with scalability in mind. They leverage both on-premise hardware and cloud capabilities, enabling them to scale to match any level of developer activity. In other words, the platform can always provision enough resources to meet demand.

This automated scalability eliminates one of the primary bottlenecks associated with legacy CI platforms—developers no longer need to wait for system resources.

Cost Savings

Transitioning to a modern CI platform can also lead to substantial cost savings. These platforms free up resources for more productive tasks by eliminating the need for extensive maintenance efforts, since there are no servers to maintain or patch. Additionally, with many of these tasks being offloaded to the CI provider, the need for dedicated maintenance personnel can be reduced.

Security

Ensuring that only the right people have access to the platform can be challenging in a large organisation. Modern CI platforms address this with features such as Single Sign On (SSO), GitHub-based authentication, or integrating with identity management platforms like Okta.

Increased Developer Happiness

One of the most significant benefits of modern CI platforms is the improvement in developer experience. These platforms are designed with developers in mind, offering user-friendly interfaces, comprehensive reports, and powerful tools to aid development.

Features like automated deployments, test reports, and project insights dashboards make the development process smoother and more enjoyable. Modern CI platforms provide an environment developers enjoy working in, leading to higher job satisfaction, lower turnover, and, ultimately, a more effective development team.

Enhanced Performance

Modern CI platforms are designed to optimse performance. They support parallelism, allowing multiple tasks to be executed simultaneously and speeding up the overall execution time.

A faster execution time means quicker feedback for developers, leading to a more agile development process. Additionally, these platforms often offer optimsed workflows, making the development process smoother and further improving performance.

Compliance

Compliance is critical for many organisations, particularly those in regulated industries. Modern CI platforms are designed with compliance in mind, providing features and controls to help organisations meet their compliance obligations.

CI platforms are certified with stringent compliance standards. Semaphore, for instance, is ISO 27001 certified, providing a high level of assurance for organisations handling sensitive data. Organisations can reduce their compliance risk and simplify their compliance efforts by using a CI platform that meets these compliance standards.

Hybrid Operation

The flexibility to run on the cloud or on-premises is another benefit of modern CI/CD platforms. This is called “hybrid operation”. It allows organisations to leverage the benefits of both deployment models: the cloud and owned, on-premise hardware. A hybrid

operation can provide the best of both worlds: cloud platform scalability and ease of use with the control and security of on-premise solutions.

Organisations can choose the model that best fits their needs and even switch between models as those needs change. This flexibility makes modern CI/CD platforms a robust solution for various organisations.

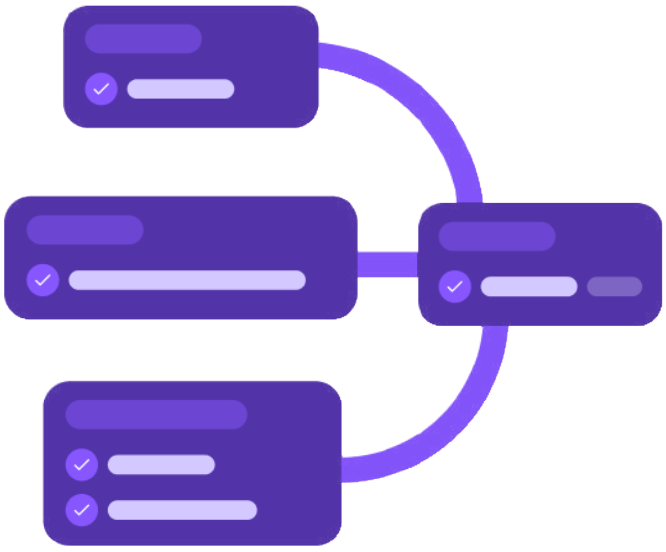
Maintenance-Free Operation

With modern CI platforms, the provider handles the bulk of the maintenance. This model reduces the burden on the organisation's internal teams and minimses downtime. Maintenance tasks are handled by professionals deeply familiar with the infrastructure.

By eliminating the need for internal maintenance efforts, organisations can focus their resources on their core business, shifting the focus to what matters: producing valuable software.

Comprehensive Support

Modern CI platform providers understand the importance of support and often offer robust support services. A non-functioning CI pipeline means developers cannot work. Therefore, CI services are often backed by service level agreements (SLAs), ensuring the system is available and issues are resolved promptly.



A robust support system reduces the stress and frustration associated with system issues. Knowing that help is just a call or email away can provide peace of mind and allow developers to focus on their work.

Transitioning to a Modern CI Platform: A Step-by-Step Approach

While the benefits of transitioning to a modern CI platform are clear, the transition is a project in itself. It requires careful planning, coordination, and execution.

We can split the transition project into six phases:

1. Assessment and planning phase
2. Establish a proof of concept
3. Setup and configuration
4. Migration
5. Training and documentation
6. Continuous improvement

Let's delve deeper into each of these phases.

1. Assessment and Planning Phase

The transition to a modern CI platform begins with a thorough assessment. During this phase, organisations should evaluate potential CI platforms, considering their scalability, performance, security features, support services, ease of use, and compatibility with existing tools.

Organisations should also examine their current CI/CD pipeline as part of this assessment. This examination should identify pain points, determine areas for improvement, and establish goals for the transition. By doing so, organisations can ensure that their new CI platform meets their needs and helps them overcome their current challenges.

Once the assessment is complete, the next step is to plan the transition. This plan should include a detailed timeline for the transition, resource allocation, and role definitions. Organisations can establish a clear plan to ensure the transition is well-coordinated, and everyone involved understands their role.

2. Establish a Proof of Concept

Before committing to a new CI platform, organisations should establish a proof of concept (PoC). During this phase, a small project or team is selected to test the new CI platform.

The PoC aims to validate the platform's capabilities and ensure it meets the organisation's needs. Success criteria should be defined, and the platform should be evaluated against these criteria. During the PoC, it's essential to work closely with the team, gathering feedback and addressing any issues that arise.

A successful PoC can provide valuable insights into the new platform's capabilities, help identify potential challenges, and provide a foundation for the full transition.

3. Setup and Configuration

Once the PoC is successful, the next step is to set up the necessary infrastructure and configure the CI platform. This step involves provisioning the required resources, setting up the system, and configuring it to meet the organisation's needs.

It's also important to implement security measures during this phase. This may include setting firewalls, configuring user access controls, and implementing encryption. By taking these steps, organisations can ensure that their new CI platform is secure and ready for use.

4. Migration of CI Workflows and Pipelines

With the new CI platform set up and configured, the next, and often longest, step is migrating workflows from the legacy system. This migration should be done incrementally, starting with less critical workflows and gradually moving to more critical ones.

During the migration, it's crucial to collaborate closely with the support teams of the CI platform provider. They can provide valuable insights and assistance, ensuring the migration goes smoothly. Each migrated workflow should be rigorously tested, with any problems addressed before proceeding further.

As more workflows are migrated, and the new system proves reliable, the migration can be expanded to include the entire codebase. It's important to continue monitoring the system and resolving any issues that arise during this phase to prevent problems down the road.

5. Training and Documentation

Training is a crucial part of the transition to a new CI platform. All system users should receive comprehensive training to ensure they are comfortable using the new platform. This training should cover all platform aspects, including its features, capabilities, and best practices.

In addition to training, detailed documentation is needed. This documentation can serve as a valuable reference for users, helping them navigate the new system and resolve any issues. A plan to promptly address user concerns and questions is also worthwhile to ensure everyone is confident using the new platform.

6. Continuous Improvement

Even after the transition is complete, the work is not over. As the needs of the organisation change, so must the CI workflows. Modern CI platforms provide productivity metrics to measure the effectiveness of the software development cycle.

In the last phase, the organisation should monitor these metrics to identify pain points and opportunities for optimisation. Then, take advantage of these opportunities and use the metrics provided to validate that productivity has increased.

CHAPTER 3

Transitioning to Modern Feature Management with Feature Flags

Ben Rometsch
Co-founder of Flagsmith and OpenFeature
Governance Board Member

Introduction

Developers working in banking and financial institutions often find themselves constrained by legacy systems that prevent them from deploying as frequently as they would like; instead, they are limited to monthly—or even quarterly—releases.

This is less than ideal for a number of reasons, but perhaps the biggest issue is that these deployments can become large and unwieldy, making them more error-prone. This is particularly stressful because banks can't risk outages or issues with critical features, so the less high-stakes the release, the better.

By transitioning to a more agile way of working that allows for more flexibility in the release process, developers can push features faster and more confidently. Feature flags are an important part of this toolkit, letting teams decouple deploy and release, remove bottlenecks, and increase release velocity. As banks begin to make the switch to modern feature management, finding tools that are built with data security in mind is essential. Look for tools that can increase agility while meeting your data security needs.

Challenges of Legacy Feature Management Processes

Complex Coordinated Releases

We built Flagsmith to solve the problems that often come with coordinated releases. For example, a developer is writing code and wants to push their feature but can't because there's a dependency on another team or another platform that's not ready. This blocks the developer, the build, and the release.

This can create a lot of frustration for the team and add pressure on release days. If the release gets pushed back because of a hard dependency, developers feel the strain. The next release gets bigger, there's increased potential for issues, and it can end in a Hail Mary release where everyone is sure something is going to break.

Feature flags and feature management tools let the developer wrap the feature in a flag and just push the code. Then, when the work they're waiting on is ready, they can go into their feature management tool and enable it. This lowers the number of release issues—like messy branch situations and monster pull requests—that are riddled with collisions.

Limited Scalability with Legacy Systems

Many banks run on legacy systems and release processes. Developers might use feature flags and feature management as part of this, but they've likely built homegrown solutions to try and meet their needs. Developers realise early on that they need to disable parts of the code, and will create feature flags in the database or using code configuration. This can work, but there is a breaking point. A few warning signs that you're reaching this point are:

- Needing feature flags to propagate across a microservice architecture
- Realising that you're tired of not knowing who changed what, when
- Getting tired of product owners or business people asking you to change a flag for them or activate a flag for a user

Let's take the final item on that list as an example. Engineers can manually go in and adjust the config file when the product team wants to test a feature, but this is often only sustainable for simple use cases (say, on/off switches for certain features). If the legacy system is deployed quarterly, these requests can mount, and it can become untenable to keep manually fulfilling these. The problem grows in tandem with team and organisational complexity.

Consider testing, too. The manual nature of legacy systems means that you might have a dedicated team to comb through your code to surface any issues before going live. But manual testing isn't feasible when you're releasing code every day. In addition, if there are even a few manual steps in your testing process, human error usually prevails, and it becomes increasingly likely that mistakes will be made.

Why Transition to Modern Feature Management Using Feature Flags?

Short answer? To ship better products to customers—faster! Long answer? Read on...

Release Velocity

Using feature flags, teams can decouple deploy and release, which means they can deploy when they want to and test in production. They're able to do gradual rollouts, automate releases, and move towards a continuous deployment process (no more stress-inducing monster pull requests).

As a bank, continuous deployment to production may be out of reach, but you can move towards a more regular release cadence that lowers the pressure on each sprint. For example, Flagsmith's customer Komerční Banka (KB) deploys every day, but to non-production environments. This has many of the same benefits and ensures developers can push out their code when it's ready, rather than being forced to wait on a release date.

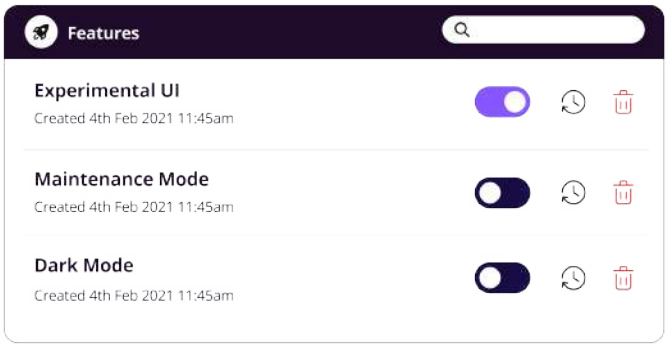
Developer Confidence

Apart from reducing cycle time, the number one benefit of introducing feature flags may be surprising: increased developer confidence and morale. We've worked with a lot of banks that have moved from massive quarterly releases to more frequent deployments, and it's always exciting to see a boost in developer confidence as they're able to push their code more easily.

If developers can release their code more frequently, they aren't as stifled by dependencies, and the pressure on each build decreases. This results in more successful deployments and features getting in front of customers faster.

Security and Compliance

Financial institutions are heavily regulated and bound to stringent data security requirements. With robust



feature management processes and tools, the banking sector can fast-track product development, while simultaneously meeting rigorous compliance standards.

In the banking development ecosystem, processes need to be agile but also secure and scalable. Feature management tools, for example, offer detailed access management through RBAC and are built with data sensitivity in mind, offering on-premise and private cloud deployment options, team roles, and remote authentication via SAML, LDAP, and more.

Built-in Safety Net

Though it might seem counter-intuitive, releasing code more rapidly using feature flags can create more stability and safety than adding a PR to your main branch when you're running production code with a high-traffic API.

Feature flags let you push code iteratively and with a safety net, giving you capabilities like:

- **Phased rollouts/canary releases:** Allowing you to test high-risk features by only showing them to a subset of users—mitigating widespread impact
- **Kill switches:** So you can quickly and universally disable features in an application

Best Practices for Modernising Release Processes

The most important thing to remember when transitioning to a modern release process is to take it one step at a time. If you're working on a large, high-impact product in a bank, completely transitioning to decoupled releases can take years. So start small, and get things right before trying to overhaul your whole release process.

Team Training and Documentation

Set up technical talks about things like best practices, what can go wrong if you don't set default flag values, how to set up A/B tests, how to debug and know which features are enabled for which users, and so on. Educate everyone in the team (not just engineers), and make sure they know how to integrate with the other tools in their stack. Remember to keep everyone looped in on releases, too. It's important to keep documenting releases as your velocity increases.

Find a Provider Who Can Help with Training

Some feature flag providers (like Flagsmith) will help with setup and training your teams on best practices, etc. This helps a lot and takes the onus off of you!

Start Small

Consider configuring the tool for a small team or a shovel-ready use case before rolling it out to a wider user base and other teams across the company. This way you can learn and iterate as you go.

Think about Best Practices for Your Company Specifically

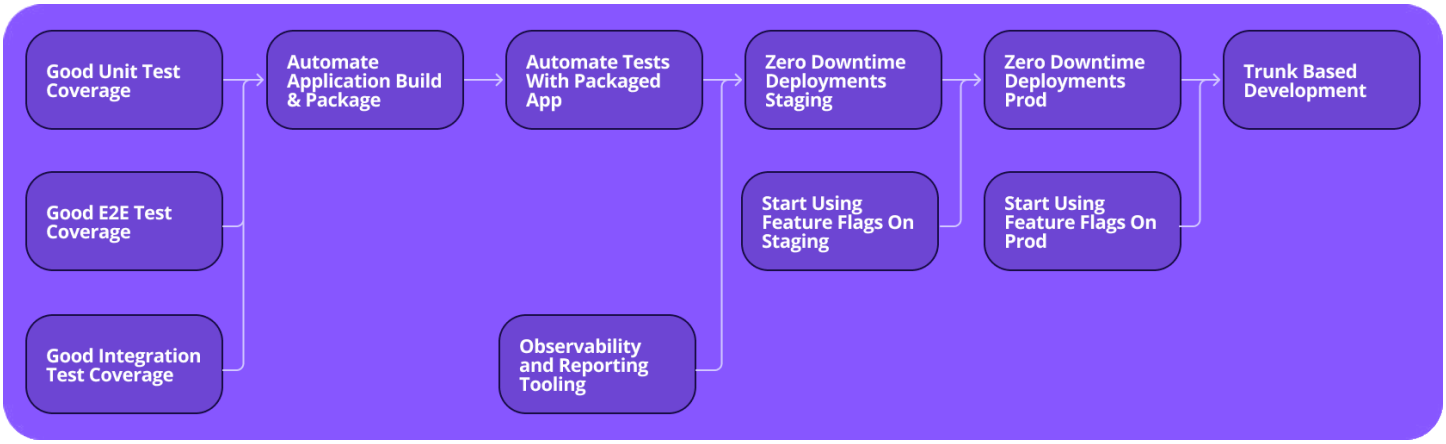
Operating in a regulated, data-sensitive industry means that you will need to be cognisant of privacy and user data. Implementing an on-premise solution, or deploying to your private cloud, will likely be the right choice over a SaaS offering.

Where Do Feature Flags Fit Into The Bigger Picture?

Teams often adopt feature flag management as part of a wider initiative around modernising the way they develop software. In financial services, this might mean transitioning from monolithic architecture and legacy banking systems to new infrastructure and ways of working (say, adopting things like the Spotify Model). If this is the case, it can be a massive project! It helps to adopt changes in the right order.

For example, I would recommend solidifying your automated testing before you introduce zero downtime deployment into your setup. Otherwise, you might just end up running zero downtime deployments that are rife with bugs.

Take the flowchart on the following page as an example of how you might structure your changes (and where feature flags fit in).



Break down the problem into the smallest possible units of work that you can. Then figure out what the correct logical order of those things might be. Some may be possible at the same time, while some may have dependencies.

You might split the changes into 12 tasks and the first three or four you can do concurrently, but then the fifth task depends on the fourth, which then unblocks the sixth, and so on. Build out a flow or diagram of the dependencies (like the one above) for each task.

If this feels too manual, feature management and observability tools can also look at your existing processes and help you figure out where to start.

Conclusion

Legacy systems can put a lot of pressure on developers for each build, especially when it comes to releases with hard dependencies. This slows down development velocity and also means developers can't release with confidence.

Feature flags and modern feature management practices let teams decouple deployments from releases and move towards a continuous deployment approach. This reduces the risk of bad builds, helps with developer morale, and gets new features in front of customers faster. Finding the right feature management tool can help with the transition, and also means that you aren't implementing new processes alone.

About Flagsmith

Flagsmith is an open source feature flag software that lets developers release with confidence. We work with banks across the world to help them transition to modern feature management and software development, offering on-premise deployments, security features, and technical support to cover your needs. We also partner with OpenFeature to support open standards and prevent vendor lock-in.

CHAPTER 4

Transitioning to Modern Authorisation Management

Alex Olivier
CPO and Co-founder at Cerbos

Navigating the ever-evolving landscape of authorisation management poses considerable challenges for companies. The dynamic nature of authorisation requirements, ranging from the addition of new roles to adapting to shifts in data security needs, demands a delicate and imperative approach. Legacy systems entrenched with hard-coded authorisation logic present hurdles in terms of scalability, developer experience, and responsiveness to changing business demands.

For regulated industries aiming for compliance benchmarks like ISO27001, SOC2, or PCI, the fragmented nature of authorisation audit logs can pose significant challenges during audits or security incidents. In this environment, the inability to efficiently handle complex roles and permissions can impede a company's growth potential.

Transitioning to modern authorisation, or decoupling authorisation management from core code, emerges as a strategic solution to the challenges posed by legacy systems. The paradigm shift involves centralising authorisation logic in audited, version-controlled policies, separating it from the application code. This decoupled authorisation approach brings about technical advantages, such as independent evolution, versioning, and testing of authorisation logic. Standardising decisions through a Policy Decision Point (PDP) not only enhances efficiency but also ensures a consistent audit trail across various components of the application architecture. The organisational impact is equally profound, as this approach fosters transparency, empowering a broader range of stakeholders to comprehend and modify the permissions structure without extensive coding knowledge.

The Challenges of Legacy Authorisation Management

Without fail, your authorisation requirements will change. Whether it is adding a new role, completely reworking your access model to suit a new product packaging framework, or enforcing a change in data security needs—this work is delicate, laborious and imperative to get right. If the thought of this immediately sounds like a three-month scope of work, or touching a ball of mud then it is a sign the authorisation mechanisms in an application need modernisation.

Consider a system where this logic has been deeply ingrained and hard-coded into application logic, this is going to be a time-consuming, expensive and

fragile process. Developers will need to go back into code bases which have gone long untouched, convert business requirements into code, update tests and redeploy the application stack. Changes to the authorisation logic become progressively more difficult to implement, often creating bottlenecks in business operations. For more heterogeneous microservice architectures, the business requirements may have to be translated into many different programming languages and frameworks, each with its idiosyncrasies and testing hoops to jump through.

In regulated industries, or for anyone looking to obtain and retain compliance such as ISO27001, SOC2 or PCI, being on top of the authorisation happening at every step of the request chain is critical. Ensuring that every microservice, function and worker job not only checks whether an action is authorised but also makes a concrete, consistent and verifiable audit log of the actions is a hard requirement. Approaches where each component creates and maintains its own authorisation audit logs, result in a fragmented set of formats, storage, and retrieval mechanisms. When it comes to demonstrating your approach to access control to your auditors or security, you may find yourself searching through disparate application logs and myriad request traces. The pain of trying to understand how actions were allowed is only amplified when being done under pressure due to a breach or data security incident.

Moreover, the inability to effectively manage complex roles and permissions can limit a company's growth potential. It can prevent businesses from selling to larger corporations, hinder operations in regulated industries, and slow down product innovation.

There is a better way to approach this which makes a system manageable, scalable and most importantly compliant.

The Case for Modern Decoupled Authorisation

Decoupled authorisation is a new approach which enables application permissions and access controls to be defined centrally and enforced locally for each application. The core promise of decoupled authorisation is to extract all the authorisation logic out of the application code base and define it in an audited, version-controlled and human-readable set of policies which can be stored as configuration rather than application code.

These policies are loaded into a Policy Decision Point (PDP) which is deployed alongside your application and then wherever permissions need to be checked, rather than hardcoding the logic, an API is made out to the PDP with all the relevant context, and a simple ALLOW or DENY decision is returned. The application code is now a simple boolean check of this response to decide whether to do the requested action or not.

This approach has two major technical advantages:

1. Authorisation logic is now independent of the application code. It can be evolved, versioned and tested in isolation. Whenever a change is made, all the PDPs receive the updated policies and immediately serve decisions based upon them without any need to touch the application code, saving time and deployment risk.

2. All decisions, regardless of where in the application architecture—frontend, backend, async jobs—are going through a standardised PDP service which will make the correct decision based on the policy but also create a standardised audit log of the decision made checking many compliance boxes.

A side effect of decoupling the logic out of the application code is more organisational. This approach allows both business stakeholders and non-expert engineers to understand the permissions structure

as policies are now written as configuration definitions rather than application code. Whilst there are some domain-specific language concepts to learn for any policy format, the effort to learn is far less than understanding the entire application code base structure and architectural decisions. This transparency empowers a broader range of stakeholders to make changes to the authorisation logic without needing extensive coding knowledge or risking the stability of the core application.

Without Cerbos

```
apiVersion: api.cerbos.dev/v1
resourcePolicy:
    resource: contact
    version: default
    rules:
      # admins can do all actions
      - actions: ['*']
        effect: EFFECT_ALLOW
        roles:
          - admin

      # user role can create and update contacts
      # if they are a member of the sales group
      - actions: ['create', 'update']
        effect: EFFECT_ALLOW
        roles:
          - user
        condition:
            match:
                expr: ('sales' in request.principal.attr.groups)

      #user role can delete a contact if they are one of:
      # - the owner of the contact
      # - member of the sales-manager group
      # - member of the product group
      - actions: ['delete']
        effect: EFFECT_ALLOW
        roles:
          - user
        condition:
            match:
                any:
                    of:
                        - expr: request.resource.attr.owner == request.principal.id
                        - expr: ('sales-manager' in request.principal.attr.groups)
                        - expr: ('product' in request. principal.attr.groups)
```

With Cerbos

```
if (await cerbos.isAllowed({ principal: user, resource, action: "edit" })) {
  //allowed
}
```

Addressing the Challenges in Shifting to Modern Authorisation

Decoupling authorisation means relying on a third party, so you need to consider the potential risks that come with it so that you can manage them well.

Firstly, vendor lock-in can feel like a loss of control since you can rely only on the features of the solution you've chosen. The language of your policies and their configuration structure will be imposed by the solution you use. If you find that a feature is missing, you may struggle to achieve your goals, and moving to another solution may be complex. It's therefore of paramount importance to choose your solution well.

That means making sure you select a solution that can support your workload and is highly available. Otherwise, the authorisation component may become a bottleneck or, worse, the single point of failure. For example, you may want to evaluate the scalability of the solution and its ability to handle your traffic load, including periods of increased traffic or sudden spikes in demand. You will need to evaluate outage risks and how to mitigate them.

Lastly, using an authorisation server means adding a call to another application, which means added latency to your application. So make sure the solution you select is performant. If you opt for a SaaS solution, consider the provider's geographic proximity to your application. The better alternative is, of course, to have a solution that can be deployed in your infrastructure.

Decoupling Authorisation: Best Practices

From our experience at Cerbos, 90% of the effort involved with migrating to a more modern authorisation solution isn't a technical one, but rather a business analysis one. Authorisation has stakeholders from across the organisation—be it security teams, customer support, product development, sales and DevOps. Getting all these parties around the table and agreeing on what the access model should be for an application is key.

We always recommend starting with a simple spreadsheet with a column for each user role, and all the actions as rows and simply fill in the boxes for which role can do each action. This quickly highlights the complexity of the permissions as the answer is not always a simple ALLOW or DENY but rather a conditional one.

Once everyone is on the same page, the actual implementation can seem rather trivial, but there are several aspects to consider. It's important to select a solution that meets your application's requirements and can scale with your needs. Consider factors such as ease of integration, performance, flexibility, and support for different access control models. Ideally, the solution should seamlessly integrate with your existing infrastructure, including identity providers, data stores, and messaging systems, to avoid introducing new vulnerabilities or dependencies.

There are various challenges that a development team may face when implementing decoupled authorisation:

Ensuring Policies Are Enforced Accurately and Efficiently

An authorisation system must be robust, accurate, and perform well under load. It's essential to validate policies before deployment and ensure that they are applied consistently across the application. For example, a development team for a large e-commerce platform must ensure that the decoupled authorisation system can handle a high volume of user requests during peak shopping times to protect sensitive user data and prevent unauthorised access to restricted resources such as payment information or order management.

Maintaining Policy Consistency Across Multiple Services and Microservices

Maintaining policy consistency across multiple services and microservices can become challenging as applications scale and evolve. Developers must ensure that authorisation rules are synchronised and that changes propagate correctly throughout the system. For example, a content streaming platform may use microservices for user management, content catalogue, and billing. As new features and services are added, the development team must ensure that authorisation policies remain consistent across all microservices.

Dealing With Policy Conflicts and Resolutions

Increasing the number of policies also increases the potential for conflicts. Developers must be able to detect and resolve these conflicts to ensure that the correct access permissions are applied in all scenarios. For example, in a healthcare application, a doctor may have access to their own patients' records, but access to records of patients outside their care may be restricted to emergency situations. In this case, the development team must implement a mechanism to detect and resolve policy conflicts, ensuring that the doctor can access critical information in emergencies but is restricted from accessing other patients' records in non-emergency situations.

Standardise Communication Across the Business

Communicating effectively will allow your business to implement authorisation quickly; use clear, concise policies that are easy to understand and maintain. Using comments and descriptive names for variables, functions, and classes helps provide context and improve readability. Periodically review and update access control policies to ensure they remain accurate and reflect the current state of your application; this helps to prevent stale or outdated policies from causing security issues.

Create Robust Testing

Before deploying policies to production environments, extensively test them. Automated testing tools and techniques like unit tests and integration tests can help validate policy behavior and ensure it meets requirements. In addition, implement robust logging and monitoring solutions, such as the ELK stack (Elasticsearch, Logstash, and Kibana) or Splunk, to track authorisation activity. Use this data to detect potential issues, analyse trends, and provide evidence for compliance audits.

Decoupled authorisation is essential to modern software development, offering flexibility, scalability, and improved security. However, a business must understand and address the technical challenges of decoupled authorisation to create robust and maintainable applications. Plan carefully and implement a solution that works for you and your organisation; this will help you adapt to changing requirements and regulations more effectively.

Conclusion

Having security concerns means your application stack likely has several layers of authorisation from the network level down to the resource levels and through the application itself. The issue with this approach to authorisation is its complexity and lack of visibility. For instance, at the application level, developers may not realise that they're writing the same code repeatedly to check who can do what.

This is where the idea of decoupling authorisation logic from the main application comes into play. It allows developers to outsource repetitive tasks and offers increased visibility on your policies. This approach reduces the overhead of managing authorisation logic and allows you to focus on implementing business logic. Decoupling authorisation also enables policy frameworks to be tested independently, increases visibility, and provides comprehensive audit trails.

Decoupling authorisation is a powerful tool that can help organisations improve their security and reliability. However, you need to carefully evaluate potential risks before selecting your solution provider. This process includes considering some downsides, such as being aware of vendor lock-in and the need to select a solution that is both performant and highly available.

About Cerbos

If you are interested in a decoupled authorisation solution, consider Cerbos to easily implement and manage fine-grained access control in your applications.

CHAPTER 5

Banking's Legacy Infrastructure

EMBRACING MODERNISATION THROUGH PLATFORM ENGINEERING

Benjamin Brial
Founder, Cycloid

cycloid

Introduction

In today's dynamic banking landscape, managing legacy infrastructure has become increasingly complex. Ops are often more busy putting out fires than innovating solutions, and lack of centralised oversight leads to resource and budget drain. In addition to that, legacy infrastructure is harder to scale to the ever-evolving demands of customers and more complex security regulations. This is why many banks are looking towards embracing modern platform solutions.

Platforms offer the promise of regulatory compliance, streamlined operations, and the ability to harness the latest technologies—everything that banks are currently struggling with. What are the benefits of self-hosted SaaS platforms? Can a GitOps approach bring about a real transformation? And what are the cultural challenges of implementing modern infrastructure management?

Challenges of Legacy Platform Infrastructure

- **Difficult to manage.** Legacy infrastructure is often complex and outdated, making it harder to maintain and troubleshoot effectively.
- **Doesn't scale.** Legacy infrastructure lacks the flexibility to accommodate growing demands, making it challenging to expand or adapt to changing workloads and user needs.
- **Ops are busy putting out fires.** With legacy systems, operations teams frequently find themselves dealing with unexpected issues and constant maintenance, leaving them little time for innovation.
- **Weak security.** Legacy infrastructure is prone to security vulnerabilities as it may not receive regular updates or patches, making it a potential target for cyberattacks and data breaches.
- **Wasted resources due to lack of oversight.** Organisations may struggle to monitor and optimise resource usage efficiently, resulting in unnecessary expenses and underutilised hardware.

Why Banks Need to Transition to Modern Platform Engineering

Platforms offer regulatory compliance and greater scalability for on-prem infrastructure.

Most banks are trying to embrace SaaS solutions, especially for non-core functions, to benefit from greater scalability, agility, and access to the latest technology without the burden of maintaining complex on-premises infrastructure.

Regulatory Compliance

Banks operate in a highly regulated environment, and compliance becomes more complex when dealing with legacy systems. Older systems might lack the necessary capabilities to quickly adapt to changing regulations, potentially resulting in non-compliance and legal consequences.

Modern platforms offer advantages in this regard. They can be designed with compliance in mind, incorporating features to track and manage regulatory requirements effectively. Automation and advanced data analytics can aid in generating the required reports and ensuring transparency.

Furthermore, modern platforms can facilitate better audit trails, making it easier for banks to demonstrate adherence to regulations and handle compliance audits efficiently.

Legacy Systems and Integration

Legacy systems, while once cutting-edge, can create challenges when integrating with newer technologies. The mismatch in architecture, data formats, and protocols can lead to complexity and inefficiency during integration efforts.

Transitioning to modern platforms enables smoother integration due to their modular and standardsed design. Application Programming Interfaces (APIs) play a crucial role, allowing legacy systems to connect with modern applications while preserving data integrity.

Risks of Transitioning to Modern Platform Engineering

The Complexity of a GitOps-based Approach

GitOps is a great approach to start optimising and modernising your infra management. It uses Git as a source of truth to deliver Infra-as-Code and applies DevOps best practices natively to improve collaboration, compliance, security, and CI/CD. However, getting started with GitOps requires time and technical expertise, which can be a barrier in traditional organisations like banks. It also usually leaves no opportunity for upscaling and therefore cannot support your long-term business goals.

Platform engineering can make your GitOps approach more user-friendly and accessible. Therefore easier to adapt to. With Git as the source of truth, the deployment process is automated and streamlined, allowing developers to focus on what they do best - writing code.

Cultural Change

Moving to platform engineering requires a cultural shift within the organisation and may require employees to adapt to new processes and workflows. Banks are traditionally very siloed in their organisational structure making change management essential for success, but lengthy to make the move.

This can be remedied by modelled behaviour—let your platform team handle the tools and specifics, but don't forget developer and end-user experience (DevX) in the process. If you build your platform with end-users in mind, your intended audience will welcome it easier.

The best hope is for gradual change—it will build incrementally, attract like-minded new employees, and gradually become the norm in your organisation.

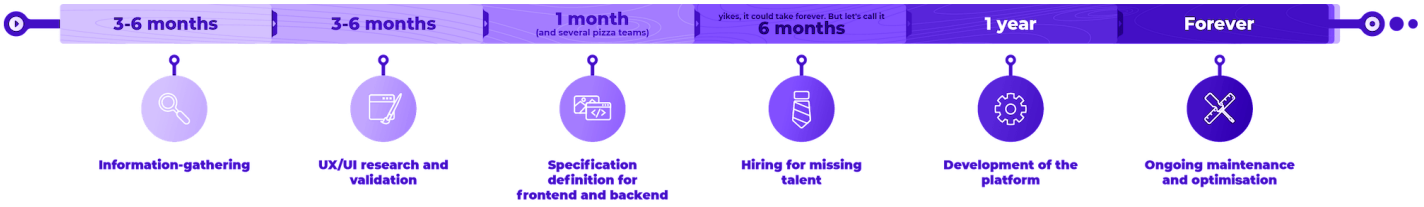
Conclusion

The world of banking infrastructure management is undergoing a profound transformation, and embracing platform engineering emerges as a definitive solution to its multifaceted challenges. As we have seen, platforms offer banks the assurance of regulatory compliance and a new level of scalability. Beyond this, the power of platforms extends to modernising and centralising legacy infrastructure, allowing banks to streamline operations and optimise performance.

However, the journey toward Platform Engineering is not just about technology; it necessitates a cultural shift within banking organisations. With siloed structures being the norm, change management becomes paramount to ensure a successful transition.

Moreover, the introduction of platforms paves the way for a more user-friendly and accessible GitOps-based approach. The combination of Git as the source of truth and platforms as the facilitator empowers banks to leverage Infrastructure-as-Code with greater ease, thereby boosting collaboration, compliance, security, and continuous integration and deployment.

It's clear that from regulatory compliance and modernisation of legacy systems to overcoming cultural barriers and simplifying GitOps, platforms bring forth a comprehensive solution. Embracing Platform Engineering will propel banks toward a future of unparalleled efficiency, innovation, and success.



About Cycloid

Cycloid aims to promote efficient infrastructure & software delivery alongside digital sobriety. We do this by optimizing platform engineering, alleviating the cognitive load on IT teams, and advocating for Green IT practices.

Our mission is to enable a future where technology and sustainability can coexist harmoniously, leaving a lasting positive legacy.

How Banks are Modernising to Approach Open Source

Rob Moffat
Senior Technical Architect at FINOS

FINOS

Professional Open Source

When you think of open source, do you see visions of coders hacking away in an attic late at night, headlines about vulnerabilities being discovered (like Log4Shell), or hordes of zealous individuals at conventions such as FOSDEM or KubeCon?

The fact is, open source software infiltrated the firm long ago—the Synopsis OSSRA report (2023) estimates that 96% of all codebases contain some open source software and that 76% of all code is open source. Projects like Linux and Kubernetes are now key components in most firms' cloud strategies.

It is perhaps surprising to learn that a lot of open source software development is also now happening within the enterprise. Corporations are rolling up their sleeves and assigning staff to work on these projects, freely donating their time and resources to further the "common good" of open source projects that they deem to be "too important to fail".

Big Players

Datamation's 2023 report gives some idea of the main movers and shakers in professional open source: Amazon, Apache, IBM, Google, Intel, Microsoft, RedHat, Meta, Oracle, and so on. These are the big tech names you would expect to see here.

Google is of particular note; their strategic approach to open source is outstanding. When faced with a competitor they want to challenge, their strategy is "let's build an open source alternative to that". This is a strategy called "Commoditizing Your Complement", attributed to Joel Spolksy back in 2002. When considering how to compete against Microsoft's Internet Explorer, Google started Chromium (an open source browser). Faced with Apple's iPhone, Google started Android, and when it wished to compete against Amazon's AWS, which had a considerable head-start, Google open-sourced Kubernetes.

Financial Services

At the time of writing (early 2024), Big Tech is at its biggest ever. The valuations of the top firms are staggering—and increasing. However, the financial services industry is no small fry either. According to Statista, financial services contributed just under 50% of the total net income generated by global financial services, fintech, and Big Tech in 2018.

Financial services companies and Big Tech—as well as many others—are growing increasingly similar all the time; they are becoming software companies. In the words of Marc Andreesen, *Software is eating the world*. Netflix is a software company that happens to be in the business of showing TV and movies. Spotify is a software company that happens to be in the business of playing music, and financial services firms are software companies that happen to be in the business of money.

Financial services firms have plenty of software know-how: consider the software you interact with every day, such as Internet banking, ATMs, card payments and so on. There's also a lot going on behind the scenes—settlements, balance sheets, electronic trading, risk management and know-your-customer systems. Because of this, financial services firms are as reliant on and invested in open source as anyone else.

Strangely, financial services firms don't appear in the list of open source "Big Players" we looked at earlier. Where are they?

Three Problems

Financial Services firms are the "Hotel California" of open source: code goes in, but (at least until recently) nothing comes back out. Why is this? Here are three main reasons:

1. Culture

The first and most obvious thing that blocks open source contributions is culture. Banks are famously secretive. The more secretive your bank is, the better! Internally, there is a culture of caution—information is provided on a need-to-know basis. Staff are taught to question why someone might be calling and asking for the name of a manager. Access control systems abound for controlling data resources or processes. This is not an accident; these systems have grown in response to the risks faced in the financial services environment. However, this is in stark contrast to the culture of open source.

2. Regulations

Many financial services firms are regulated. Market forces and competition can erode systems of trust, so it is up to governments and regulators to set and enforce rules to make sure our financial systems are stable.

For example, regulated firms are required to keep detailed logs of messages to clients. In 2022 the SEC fined several major firms for un-monitored communications using WhatsApp, to the tune of $1.1bn. These are important regulations to ensure the fairness of markets and avoid insider dealing. The size of the fines underscores this.

Thus, a large part of the effort of running a financial services organisation goes into following regulations to avoid these kinds of fines. Whether they are related to anti-money laundering, counter-terrorism, security or the safety of personal information, avoidance is key.

In protecting bank IP and customers' personal information, it is clearly better to err on the side of caution, and in doing so, often open source is the casualty.

3. Rules

To meet regulations and control risks around confidential information and reputation, firms enact policies, procedures and rules. For example: it would be easy to accidentally or deliberately divulge confidential information in a Facebook post or a Google Doc. So, broadly, sites which could be involved in this are "firewalled" or blocked internally.

Likewise, the penalties for sharing confidential information (source code included) via email are strict.

From this perspective, a site like GitHub (commonly used for open source development) is just as likely a vector as Facebook, WhatsApp or Google Docs. Usually, these are locked down tightly.

In summary, there are lots of things standing in the way of open source participation in financial services. It's worth looking at what is lost as a result of this.

What Is Lost

It would be very easy at this point for financial services firms to throw in the towel and decide that there are too many barriers standing in the way of open source contribution. But it's worth reflecting on what is lost by doing this. Let's look at four of the most obvious problems:

1. Dependencies and Vulnerabilities

Financial Services firms are auditing their open source dependencies and discovering that sometimes, key projects are maintained by "that one guy in Nebraska" to quote the XKCD comic strip.

This is a driver for open source contribution, as staff can argue: "We have a dependency risk issue here. These are some of the threats that we have to figure out. We should be investing in open source and we should be adding maintainers to these projects to keep track of these things."

It makes sense for financial services firms to help maintain code and keep projects up-to-date. This way, the code is there for the future when they need it, rather than it getting abandoned by its maintainers. Abandoned code creates more work for firms since they then have the costs of migrating to new solutions.

2. Excessive Internal Forks

The alternative to the abandonment problem is *forking projects internally* and maintaining those. At first glance, this seems like a way to mitigate risk and keep control of your codebase—but then you have to maintain all the forks. Every time the upstream, open source version changes, you have to update the internal versions and you're in a massive maintenance hell.

Typically this happens when an engineer discovers a bug or missing feature in an open source project and can't supply the fix. They then fork internally. Worse still, this can happen multiple times within the same firm. Imagine having development staff spending their time maintaining multiple, different out-of-date versions of (say) the React project just because each developer discovered a different bug that needed fixing.

3. Strategically Compromised Positions

As discussed earlier, major companies like Google are using open source as a strategic lever. There is always a risk that another firm might build a vital open source piece of software that changes the finance industry. For example, what if a major bank introduced something along the same scale as Kubernetes, but for finance instead of cloud computing?

Any firm that doesn't have a way for its teams to engage with that open source project would be unable to influence its future. This would limit the value they could get from it, which could be strategically damaging. A firm that doesn't "do open source" can't help steer the ship—they're either stuck as a passenger or they're stuck watching the ship go by without them.

Another analogy which is helpful when thinking about open source as a strategy is to consider the Peloton in cycling: rather than every cyclist battling air resistance, they work together and slipstream each other. They are all competing with one another, but at the same time, they're working together. If a firm isn't engaged in an open source effort, they're battling the wind all by themselves. A single firm is unlikely to be able to commit the resources to compete with a "Peloton" of other firms in the industry.

Given that "software is eating the world" this is a strategically compromised position.

4. Talent

Consider the case where a firm builds a product on an open source project. As discussed before, if the project is critical enough to that product, the firm should appreciate the key dependency risk involved in the open source project.

One strategy they might adopt to mitigate that risk is to hire one of the project's maintainers to come and work for them. That way, they have the talent on hand should issues arise with the dependency.

There is a critical flaw in this plan: if the company isn't set up to allow open source, that new staff member won't be able to engage with the project they maintain! So, they're faced with a choice: either leave the firm or leave the open source project. Furthermore, if they choose the latter, this is also bad for the firm, since it *increases* the dependency risk on that open source project. The project now has one less maintainer!

A further argument in favour of open source in this area is that talented developers are attracted to open source. They want to work on quality projects that enhance their profile and open source is a great way to do this. Allowing open source contributions is an employment benefit.

FINOS

FINOS is the FinTech Open Source Foundation and part of the Linux Foundation. Its membership (around 80 entities at the time of writing) comprises various firms across the financial services industry and its suppliers. FINOS has Platinum, Gold and Silver members, all corresponding to the different membership fee levels and benefits. Many of the platinum and gold-level members are exactly the sort of heavily regulated firms I've described above: they are all-in on the potential value of open source software. They see the risks and issues posed above by not contributing to open source and want FINOS to help resolve them.

But that's not all: FINOS hosts many open source projects that financial services firms want to take advantage of. As well as trying to fix the issues described above, there are some huge prizes for the industry if it can collaborate on open source projects. Let's look at some:

1. Open RegTech

According to International Banker, financial institutions spend $270 billion per year on complying with regulations. When a regulatory agency imposes a new requirement, the firms all have to comply or face fines. Largely, the work building systems to meet new regulations is repeated in every firm.

The idea of firms collaborating on open source implementations of regulations represents a huge opportunity for cross-industry cost-saving.

2. Cloud Computing Standards

FINOS has two projects, Common Cloud Controls and Compliant Financial Infrastructure aimed at allowing financial services firms to get the most out of the cloud. Building a cloud platform that complies with worldwide financial regulations is a big ask, so why not collaborate with your peers to share the cost?

Additionally, "given enough eyeballs, all bugs are shallow" (to quote Linus Torvalds), the more cross-industry collaboration that occurs on these projects, the better the quality of them will be.

This is not just true for FINOS' cloud computing standards but for other standards such as FDC3 (financial desktop interoperability) and CDM (a financial services Common Domain Model).

3. Emerging Technologies

The technology landscape is not only changing extremely rapidly but the rate is not slowing down. With this in mind, organisational strategy needs to be constantly on the lookout for external technological disruption and innovation.

FINOS' Zenith project is all about looking at innovation and trying to map it back to the financial services industry. How does AI intersect with finance? Where is it best deployed? What about blockchain, augmented reality or quantum computing?

4. Open Source Best Practices

FINOS runs Open Source Readiness, which gets financial services' open source advocates together every couple of weeks. FINOS helps document the path to adoption and excellence in open source and provides a way for firms to share best practices and approaches. In 2024, FINOS are starting to look at best practices around open source AI and focus more closely on supply chain security measures.

Baby Steps

An attitude shift has to happen before change can pick up pace. Financial Services firms are heavily using standards such as FDC3 behind their firewalls. But to maximise their benefit and improvements to the standard, they need to contribute back.

While there's a long path ahead before most internal teams can make contributions and truly adopt open source, many firms are taking the first steps. For example:

1. Trial Projects

The organisation will grant a board-level agreement for a project. Teams can use open source on it and run a risk analysis. Instead of doing it entirely in-house or going to a third party and getting a proprietary solution, they try open source and see how that works and take it from there.

2. Tackling Policies and Regulations Head-On

This means working with all of the organisation's policies to understand them and try to chart a way through them to allow for open source. Often, it's hard for firms to even craft a policy around open source which doesn't either infringe on existing policy or overstep regulations. For this reason, FINOS has worked with its members to produce a sample policy (open sourced from one of its member banks) and annotations, explaining why the policy has been crafted the way it has.

3. Technology

FINOS projects such as GitProxy allow a compliant, DLP-aware workflow for pushing code to public repositories.

4. Training

FINOS has just released LFD137 - Open Source Contribution in Finance. A free Linux Foundation training course aimed at teaching developers in financial services firms the best practices for contributing to open source. This is starting to see acceptance within regulated firms.

5. Certification

FINOS (and the Linux Foundation) have just released FSOSD - Financial Services Certified Open Source Developer, a proctored, on-line certification allowing candidates to prove that they understand the risks associated with open source contribution and the best industry practices for mitigating them.

Conclusion

Every project is not going to become an open source project. This includes projects which help integrate internal systems, projects which contain the algorithmic "secret sauce" of the business and projects which might be vital but aren't going to be of interest to the open source community in general. These "in-house" projects will always exist.

There is now compelling evidence that in-house by default is being challenged, however. In the future, we should expect to see financial services choose between in-house and open source based on merit.

If you're working in financial services and are interested in understanding your firm's open source journey, why not get in touch with FINOS via their email help@finos.org. They would be happy to hear from you and put you in touch with like-minded individuals within your organisation.

CASE STUDY

How Komerční Banka Modernised Its Banking Infrastructure Using Feature Flags

Meet Komerční Banka

Komerční Banka (KB) is a leading Czech Republic banking institution with over 1.6 million customers and nearly 400 branches. The bank was built on legacy systems with huge, distributed monolithic architecture. To stay competitive, KB felt the need to modernise its infrastructure and software development approach. Due to its large-scale operations, there was an urgent need to modernise its infrastructure and software development to stay competitive.

Initial Challenges

KB had over 400 developers (out of 600) managing its legacy system built on monolithic architecture. The problem? The system consisted of heavily interdependent components that led to longer development cycles, higher bug risks, and a lack of system scalability.

These issues prompted KB's COE (Center of Expertise) to improve efficiency. As a result, they launched "The New Digital Bank" initiative—a five-year project migrating their infrastructure to a microservices-based architecture.

The goal was to modernise KB's banking system and adopt an **Agile 2.0 development mindset**—resulting in better service delivery.

KB's Modernisation Hurdles

KB developed its development framework using:

- Java SDK to build cloud applications
- Jenkins and Argo CD for CI/CD pipelines
- Kubernetes to deploy and run applications

As the infrastructure was built on monolithic architecture, every feature release, update, or product launch moved at snail's pace.

For example, it took the team days to get the system back up and running whenever there was a new Kubernetes update. There was no scope for automation of updates or independent development cycles.

Plus, the current system was a massive drain on costs. They were already maintaining systems on-premise while subscribing to expensive tools like IBM. To overcome this issue, they were looking for open-source technologies as they're free and have minimal upkeep costs.

How Feature Flags Helped

The use of feature flags is common in the banking industry. Many companies, including KB, developed their own systems to manage them. KB's homegrown tool handled simple tasks like managing the configuration files because the legacy system was deployed only three times a year.

However, KB was taking an Agile approach to development. So, they needed a tool that supported the approach and fit into the new microservices architecture.

When evaluating a feature flag solution for their modernisation effort, KB tried three options: continuing with their own system, adopting Flagsmith, and adopting LaunchDarkly.

Ultimately, they chose to go with Flagsmith's solution because of its:

- Open-source software
- System's flexibility
- Excellent documentation
- Support capabilities

The Results

Before adopting Flagsmith, KB's engineers had to wait months to ship new features or product releases. Their legacy system was deployed only three times per year. But now, they deploy daily to two non-production environments. One environment is used for testing during the build and the other is used for staging.

The goal was to get to production as fast as possible by keeping these environments close to each other. By doing so they can ship as fast as they need to—inching closer to completing the New Digital Bank initiative.

From the time they've implemented Flagsmith, over 85 engineers have begun working with it in production. That's over half of the team dedicated to the program.

Flagsmith passed all their internal security checks with flying colours. When you consider how sensitive the banking industry is to such requirements, having a SOC 2-certified tool on their side only smoothened the transition.

Going forward, they can deploy features and products with confidence in a controlled environment without impacting the rest of the system.

In the future, KB plans on expanding Flagsmith's usage within the company. They will start by integrating the API with JIRA to align QA teams with product owners. It'll help them move flagging processes upstream, improving efficiency in the long run.

Find the full interview with KB here.

Conclusion

This eBook is a collaborative project between different leaders in software development, banking, and open source. Each contributor sees modernisation through a different lens, just as each financial services company chooses to modernise and release software in a different way. We started this project with transparency and collaboration in mind because there's so much we can learn from one another in the open source community, the developer community, and the banking community.

This eBook is designed to be a roadmap for modernising banking software development, covering critical areas such as feature flags, observability and open source. Using the tools and approaches laid on in each chapter, our hope is that you can unlock different ways your teams release and develop software.

Contributors

- Romano Roth from Zühlke
- Andreas Grabner from Dynatrace
- Tomas Fernandez from Semaphore
- Ben Rometsch from Flagsmith
- Alex Olivier from Cerbos
- Benjamin Brial from Cycloid
- Rob Moffat from FINOS

More Information

If you'd like to find out more about any of the companies or contributors from this eBook, follow the links above.

If you have any questions about how your company can modernise their approach to software development, contact us and we can answer any questions or point you in the right direction for who to speak to.

We also invite you to stay connected with Flagsmith and each contributor on social media for updates, discussions, and additional resources.

Thank You

Thank you to each of the individual contributors and companies for your collaboration and commitment to this eBook project. Your insights and work on this project has been immensely valuable for learning more about the way the financial services industry can release software. Thank you, reader, for your interest in the eBook. We hope you found it valuable and look forward to creating more resources to help you release with confidence.

Flagsmith

Get in Touch

Flagsmith is an open source feature flag software that lets developers release with confidence. We work with banks and financial institutions across the world to help them transition to modern feature management and software development, offering on-premise deployments, security features, and technical support to cover your needs. We also partner with OpenFeature to support open standards and prevent vendor lock-in.

"Our development speed and velocity have increased. Mainly, though, I just feel good about releases. I know when I ship something to production it's going to be safe and I won't have to do a thousand tests to make sure I don't miss something. When things are behind a feature flag, I know what is and isn't enabled in production and I have the visibility I need."

Vontobel

Globally active investment firm with Swiss roots

Flagsmith.com