Building Scalable Developer Platforms

Engineering Excellence Through Integrated Tooling and Infrastructure

By: Kommireddy Veera Venkata Subrahmanyam | Micron Technologies

The Current Challenge

40%	60%	3x
Engineering Time	Reduction	Faster Delivery
Spent on infrastructure tasks instead of core product development	Possible decrease in deployment complexity with proper platform engineering	Potential increase in feature delivery speed with Internal Developer Platforms (IDPs)

These statistics highlight the critical need for streamlined developer platforms that reduce infrastructure overhead and accelerate development cycles.

Key Challenges in Platform Engineering

Tool Sprawl & Fragmentation

Disconnected tools create silos and inefficiencies across the development lifecycle

Lack of Standardization

Inconsistent practices lead to quality issues and knowledge gaps

Manual Toil

Repetitive tasks consume valuable engineering resources

Cultural Resistance

Teams may resist adopting new platforms and workflows

Skill Gaps

Specialized knowledge required for modern infrastructure

Scaling Pressures

Growing teams and systems demand robust, flexible platforms

The Platform Engineering Imperative

Complexity of Modern Software

- Microservices architectures
- Multi-cloud deployments
- Complex dependency management
- Distributed systems challenges

Fragmented Developer Experience

- Inconsistent workflows
- Infrastructure toil
- Lack of standardization
- Cognitive overhead

These challenges create significant friction in the development process, reducing productivity and increasing time-to-market.

The Platform Engineering Solution

Internal Developer Platforms (IDPs)

Unified interfaces that abstract infrastructure complexity

Developer-First Approach

Designed with developer experience as the primary focus

Relentless Automation

Eliminate manual tasks through comprehensive automation

Clear APIs & Interfaces

Well-defined contracts between platform and applications

Metrics-Driven Success

Measure adoption, efficiency, and developer satisfaction

Core Components: Self-Service Infrastructure

On-demand provisioning replaces ticket-based model

Self-service infrastructure empowers developers to provision resources without waiting for operations teams, dramatically reducing bottlenecks.

Key Benefits:

- Faster delivery cycles
- Reduced wait times
- Developer empowerment
- Consistent environments
- Standardized configurations

Core Components: Integrated Toolchain Automation

Single Source of Truth

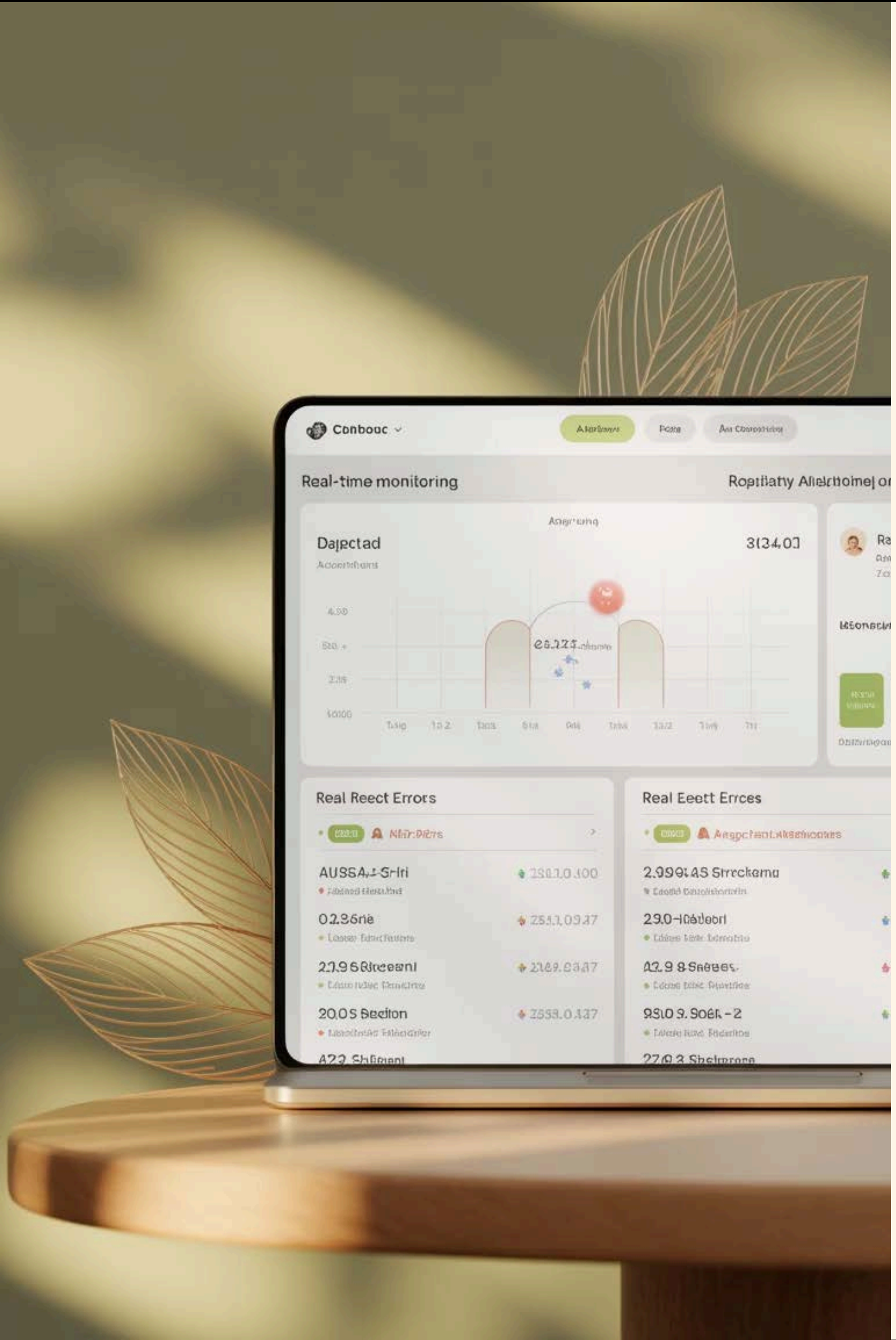
Version control + configuration management ensures consistency across environments

CI/CD Pipelines

Automated build, test, and deployment with security & compliance built-in

Deployment Automation

Consistent, repeatable deployments across environments with rollback capabilities



Core Components: Observability & Monitoring Frameworks

Comprehensive Visibility

- Logging: Centralized log aggregation
- Metrics: Performance and business KPIs
- Traces: Distributed transaction tracking

Advanced Capabilities

- Anomaly detection
- Interactive visualization
- Proactive optimization
- Automated alerting

Effective observability transforms reactive troubleshooting into proactive optimization, reducing MTTR and improving system reliability.

Industry Best Platform Engineering Tools

Self-Service Infrastructure

Kubernetes, Terraform, Pulumi, Crossplane

CI/CD

ArgoCD, Tekton, GitHub Actions, Jenkins X

Observability

Prometheus, Grafana, OpenTelemetry, Datadog

Service Management

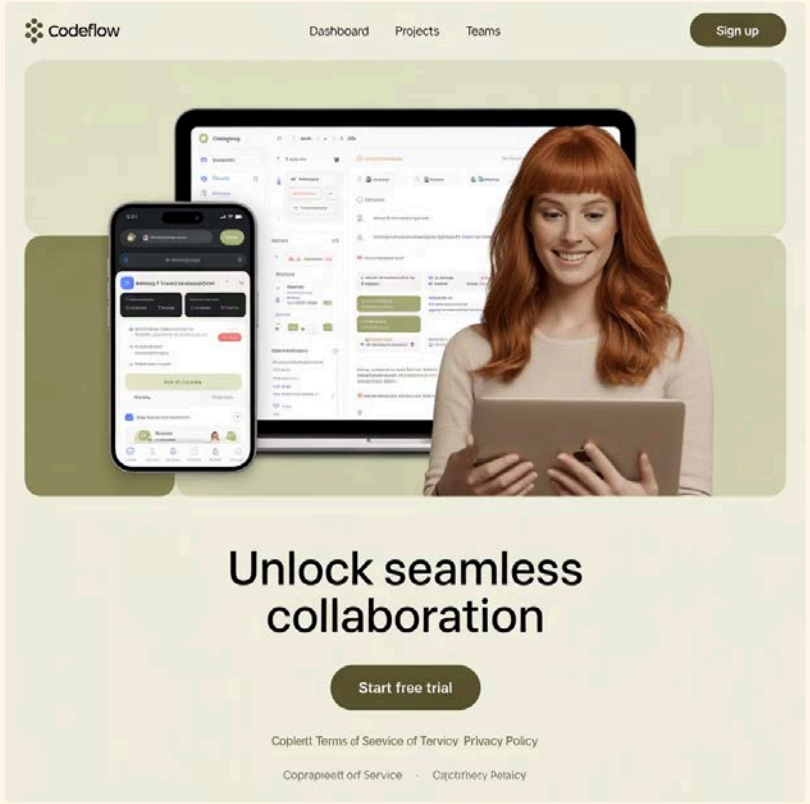
Backstage, Humanitec, Qovery

Security

Vault, OPA, Snyk

These tools form the foundation of modern platform engineering stacks, with integration capabilities that enable seamless developer experiences.

Implementation Strategies: Developer-Centric APIs & Interfaces


codeflow
Dashboard
Projects
Teams
Sign up
Unlock seamless collaboration
Start free trial
Coplett Terms of Seevice of Terviсy Privacy Policy
Coprapeett orf Service · Cajcbrnecy Petaicy

API Design Principles

- REST, GraphQL with consistent patterns
- Clear documentation and examples
- Versioning strategy
- Error handling standards

Multiple Interface Options

- Command-line tools for automation
- Web portals for visualization
- APIs for integration
- UX-first design approach

Implementation Strategies: Establishing Adoption Metrics

Adoption Metrics

- Active users
- Time-to-first-deployment
- Platform feature usage
- Migration from legacy systems

Productivity Metrics

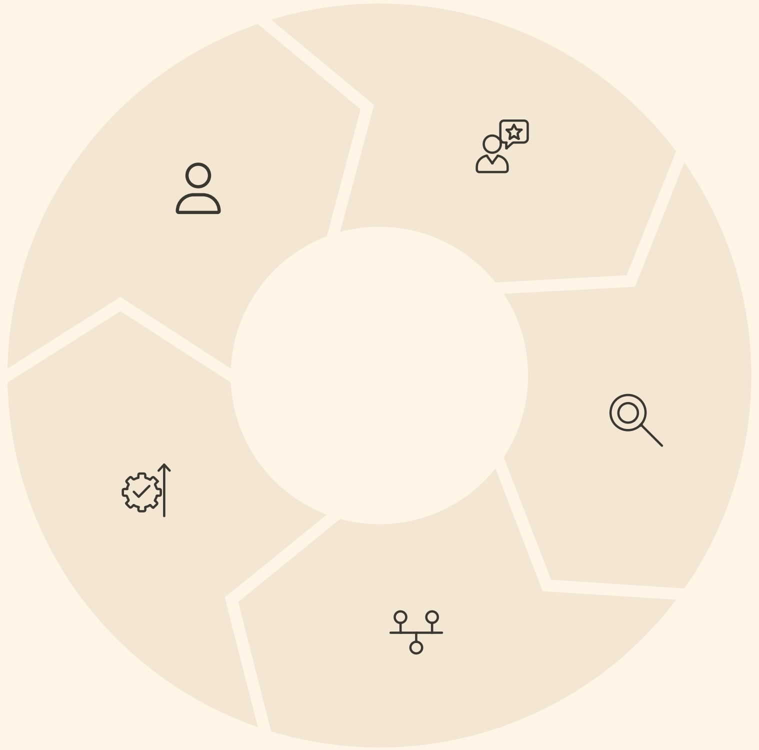
- Deployment frequency
- Mean Time to Recovery (MTTR)
- DORA metrics
- Lead time for changes

Satisfaction Metrics

- Developer surveys
- Net Promoter Score (NPS)
- Support ticket volumes
- Qualitative feedback

Measuring these metrics provides visibility into platform effectiveness and guides continuous improvement efforts.

Implementation Strategies: Creating Feedback Loops



Telemetry

Collect usage data, error rates, and performance metrics

Implementation

Deliver improvements based on feedback priorities

Structured Feedback

Regular surveys, interviews, and feedback sessions

Analysis

Identify patterns, pain points, and improvement opportunities

Roadmap Communication

Share plans and progress to build trust and alignment

Continuous feedback loops ensure the platform evolves to meet developer needs and organizational goals.

Real-World Success Story 1: E-commerce Giant

Platform Initiatives

- Standardized environments across development, testing, and production
- Automated CI/CD pipelines with integrated security scanning
- Centralized observability platform with custom dashboards

Impressive Results

- 3x increase in feature velocity
- 75% reduction in production bugs
- 60% fewer deployment failures
- Improved developer satisfaction

Real-World Success Story 2: Financial Services Firm

Embedded Compliance

Automated security checks and policy-as-code implementation

Self-Service Provisioning

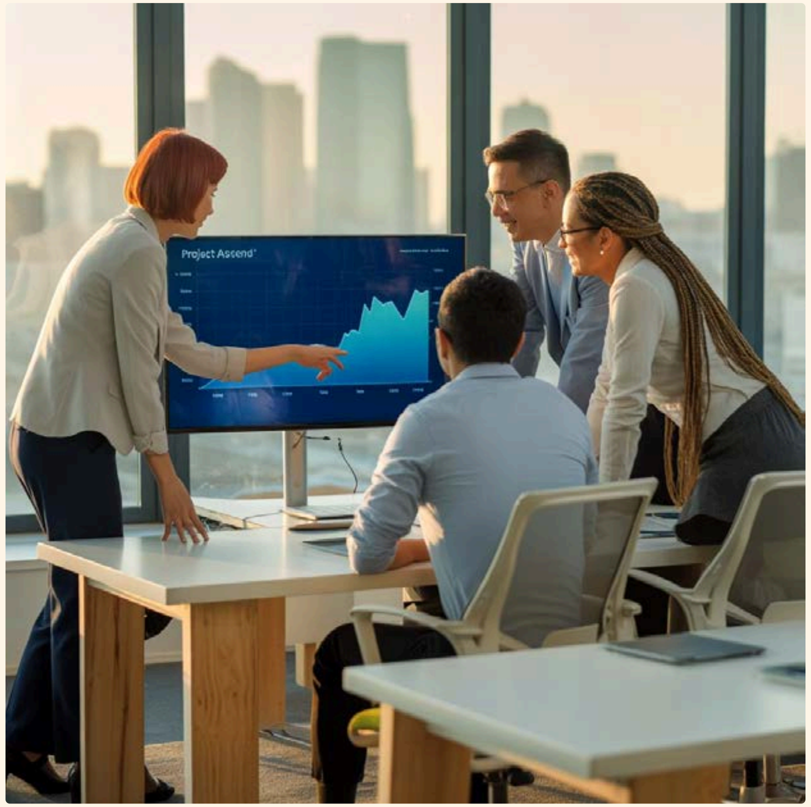
Compliant environments available on-demand

Transformative Results

50% reduced overhead, deployment time from weeks to days, 70% fewer incidents

By embedding compliance and security into their platform, this financial services firm achieved both regulatory compliance and development agility—previously considered mutually exclusive goals.

Real-World Success Story 3: Technology Startup



Platform Approach

- "Golden paths" with flexibility for innovation
- Automated onboarding for new engineers
- Scalable infrastructure templates

Growth Impact

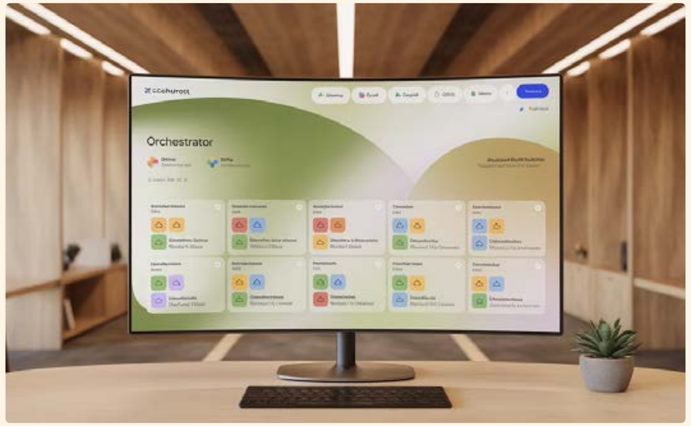
Successfully scaled from 10 to 100 engineers while maintaining productivity and code quality. New engineers became productive in days rather than weeks.

Platform Engineering vs DevOps vs SRE

Aspect	Platform Engineering	DevOps	SRE
Goal	Build Internal Developer Platforms	Bridge Dev & Ops	Ensure reliability
Focus	Abstraction, automation, DX	CI/CD pipelines, culture	SLIs, SLOs, error budgets
Users	Developers	Dev + Ops teams	Reliability engineers
Tools	Backstage, ArgoCD, Terraform	Jenkins, GitLab, Docker	Prometheus, Grafana
Metrics	Productivity, adoption	MTTR, deploy frequency	Uptime, latency

While these disciplines overlap, they have distinct focuses. Platform Engineering creates the foundation that enables both DevOps practices and SRE principles to thrive in an organization.

Emerging Trends & Future Directions



Multi-Cloud Orchestration

Seamless workload distribution and management across cloud providers, with abstracted infrastructure differences

AI-Powered Dev Tools

LLMs for code generation, intelligent automation, and predictive analytics to enhance developer productivity

Sustainable Operations

Balancing efficiency and reliability with environmental impact considerations in platform design

Building Your Platform Engineering Strategy

Assessment & Planning

- Evaluate current developer workflows
- Identify pain points and bottlenecks
- Define platform vision and principles
- Prioritize capabilities based on impact

Building the Right Team

- Blend infrastructure, development, and UX skills
- Foster product mindset for platform development
- Establish clear ownership and responsibilities

Measuring & Iterating

- Implement metrics from day one
- Create tight feedback loops with users
- Continuously evolve based on data and feedback
- Celebrate and communicate wins

A successful platform engineering initiative requires thoughtful planning, the right team composition, and a commitment to continuous improvement.

Key Takeaways

1 Platform engineering is essential for modern software development

The complexity of today's software ecosystems demands abstraction and automation to maintain developer productivity.

2 Developer experience should be the north star

Successful platforms prioritize developer needs, creating intuitive interfaces that abstract complexity without limiting capability.

3 Measure what matters

Establish clear metrics for adoption, productivity, and satisfaction to guide platform evolution and demonstrate value.

4 Start small, iterate quickly

Begin with high-impact capabilities, gather feedback, and continuously improve rather than attempting to build everything at once.

"Platform engineering is not optional — it's the foundation for software excellence."

Thank You | Q&A

Kommireddy Veera Venkata Subrahmanyam | Micron Technologies