Jay Blanchard

Principal Platform / DevOps Engineer
Cloud Infrastructure | Developer Experience | Automation | Technical Leadership

https://jayblanchard.net
https://www.linkedin.com/in/jay-blanchard-53b4632/

> ***I got tired of pretending a 30-year engineering career is best explained as a chronological collection of bullet points.***
>
> ***It's better explained by the problems solved.***

I build and modernize the machinery behind software delivery: cloud infrastructure, CI/CD platforms, deployment automation, and the paved roads that let engineers spend more time engineering.

My career has moved between hands-on engineering and technical leadership, often while changing critical systems that cannot simply be turned off. The recurring goal is straightforward: **reduce operational friction without sacrificing reliability, security, or engineering ownership.**

Build the New Machinery Beside the Old

At **Constant Contact**, Payments-as-a-Service was a multi-application payments platform running on-premises. We needed to move it to AWS without interrupting the business it supported.

We built reproducible AWS environments using **CloudFormation**, progressively migrated applications through controlled DEV, STAGE, and PROD environments, and deliberately left the largest internal workload until last. When networking issues disrupted the first major cutover, we rolled back, corrected them, and completed the migration successfully a few days later.

That pattern has followed me throughout my career: build beside production, automate what can be reproduced, validate with real workloads, make rollback ordinary, and move only when the replacement has earned the right to become production.

Earlier, at **Pocket Communications**, I used the same approach to replace a third-party billing platform approaching a **$1 million renewal**. A small cross-functional team built and pressure-tested the replacement while the existing system continued operating. A Day 95 cutover failed its data checks and was rolled back. The Day 99 attempt succeeded.

Approximately **450,000 customers moved without customer-visible disruption**, and Pocket avoided the renewal.

Good migrations aren't defined by never failing. They're defined by making failure recoverable.

Remove Operational Toil

At **Newfold Digital**, our existing Jenkins environment was on-premises, difficult to maintain and upgrade, and carrying an enormous workload that couldn't simply be switched off.

We designed its successor around **Kubernetes and Helm**, with Jenkins architecture and configuration managed as code. Build agents became Docker images and YAML definitions launched as **temporary Kubernetes pods**, moving build workload away from the primary Jenkins system and allowing compute to exist only while jobs were running.

We built reusable pipeline templates for containerized and non-containerized applications, integrated **Artifactory and AWX/Ansible**, standardized semantic versioning, and developed delivery patterns supporting application migration to **OCI**.

The objective wasn't merely newer tooling. It was a **low-touch delivery environment** in which application engineers didn't have to fight Jenkins, containerization, versioning, or deployment mechanics every time they shipped software.

We also incorporated **AI-assisted Jenkins diagnostics** to help engineers recognize when failures in pipeline logs belonged to their applications and when platform support was appropriate.

Application teams still owned their templates and migrations. We provided documentation, training, reporting, deadlines, and hands-on support. When an application didn't fit the paved road, we improved the road and documented the pattern for the next team.

Automate the work that teaches nothing. Don't automate away ownership.

Make Complex Infrastructure Problems Legible

When Constant Contact separated from what became **Newfold Digital**, Newfold would receive the existing Payments-as-a-Service platform. Constant Contact still needed to operate independently with effectively the same capabilities, but only its own data.

The TSA program was struggling to turn that requirement into executable work.

I broke the separation down application by application:

What must be duplicated? What remains temporarily shared? Whose data belongs where? Which dependencies must be broken? In what order? Who owns each piece? What proves separation is complete?

The largest risk was the database. Payment transactions continued throughout the work, database teams on both sides were stretched thin, and data had to be replicated, separated, verified, and eventually purged while maintaining **PCI controls**.

I converted the technical work into a timeline the PMO and executive teams could operate against and identified when database, network, application, security, contractor, and leadership involvement would be required.

When contractor capability threatened the deadline despite documentation and side-by-side training, I moved back into hands-on implementation while database and network teams retained ownership of their specialties.

That included **CloudFormation, Jenkins pipelines, scripting, tokenization operations, application changes, and supporting automation**.

Technical leadership doesn't always mean writing the code. It means staying close enough to know when you need to.

Build Platforms Engineers Can Own

At **Fugro**, OARS software operated aboard offshore survey vessels across a mixture of application technologies, on-premises systems, and IoT devices. Delivery depended heavily on Make and lacked mature version-control and release practices.

Working with an AWS partner and the internal engineering team, I introduced **Docker, Git, Jenkins pipelines, versioning, and repeatable delivery practices** while helping shape the cloud architecture.

The important outcome wasn't that I knew how the delivery system worked.

The OARS engineers learned to own it themselves.

At Constant Contact, that philosophy expanded across engineering. Together with engineers from two other groups, I developed a five-part Docker education series with presentations, examples, and repositories. More than **100 engineers** attended the first session. I later received a **Rovie Award for Engineering Excellence** for helping engineers adopt DevOps practices across the organization.

Platform engineering works best when the platform becomes something engineers can trust without needing a specialist standing beside them every time they use it.

Relevant Technical Landscape

Cloud & Infrastructure: AWS • OCI • Kubernetes • Docker • Helm • CloudFormation • Infrastructure as Code • ECS • S3 • Route 53 • IAM • Networking • Load Balancing • Disaster Recovery

Platform Operations & Automation: Jenkins • CI/CD • AWX/Ansible • Artifactory • Git • Bitbucket • GitHub • Maven • Bash/Scripting • Semantic Versioning • Pipeline Templates • Ephemeral Build Agents • Developer Self-Service

Systems & Software: Linux/Unix • REST APIs • Distributed Systems • Java • PHP • C/C++ • SQL • MySQL • SQL Server • Tomcat • Application Modernization

Security & Governance: PCI-DSS • Secure Payments • Tokenization • IAM • Architecture Reviews • Vulnerability Remediation • Penetration Testing • Audit Support • Operational Readiness

Technical Leadership: Platform Strategy • Cloud Migration • Technical Program Leadership • M&A/TSA Separation • Developer Enablement • Mentoring • Cross-functional Engineering • Vendor/Partner Management

Career Index

Newfold Digital — Principal Engineer | Platform Engineering / Developer Experience
Jenkins/Kubernetes modernization • Helm • Docker • Ephemeral build agents • Artifactory • AWX/Ansible • IaC • OCI • Pipeline standardization • AI-assisted diagnostics

Constant Contact — Payments-as-a-Service | Cloud / DevOps Engineering
AWS migration • CloudFormation • Docker • Jenkins • Git • PCI • Secure payments • TSA separation • DevOps education

Fugro — OARS | Cloud & Delivery Engineering
AWS • Docker • Jenkins • Git • Release automation • Offshore/IoT systems • Platform enablement

Apex Innovations — Technology Leadership | Healthcare Education
AWS adoption • Application modernization • Source control • Versioning • Repeatable delivery • Cloud migration

Pocket Communications — Director of Information Technology
Software/systems engineering • Billing • Provisioning • Payments • Operational automation • $1M billing replacement • ~450K-customer migration

Additional

Rovie Award for Engineering Excellence — Constant Contact
Patented Payments/ACH Work — Pocket Communications
United States Air Force — Early technical and mission-oriented leadership experience