White Paper

Flux7

Optimized Code Delivery Pipelines

DevOps, Docker Containers and Automation

Digital business is no longer an advantage, for competitive businesses, it's a requirement. As software becomes essential, web and application development teams are under pressure to do more with less, speed the frequency of software releases and maintain pace with agile competitors. This, in spite of the increasing scarcity and cost of experienced programming talent and the need to modernize development models and processes.

Considerations

- **Gauge your release speed.** Does your team consist of great developers, yet application deployment rates remain unacceptable?
- **Evaluate your onboarding process.** Is onboarding a lengthy process or are your able to quickly slot developers in and get them producing results with the right environment, resources and processes?
- **Weigh-up the cost of opportunity.** Are you able to efficiently meet new market opportunities?
- **Scope potential for automation.** Are there repetitive tasks that could be automated?
- **Estimate alignment.** Does conflict exist between development and operations regarding resources, controls or process?

The New Foundation: Frameworks for Success and Innovation

Innovation can be accelerated by creating code delivery pipelines that promote innovation and enable the easy, efficient delivery of quality software. Reducing the cost of failure and iteration time and improving continuity can significantly improve the process of creating good software. Since you are constantly optimizing developer resources, it will also assist in addressing and reducing the common need to attract, train and retain skilled programmers and make the most of developer resources.

Creating efficient code delivery pipelines means developers can:

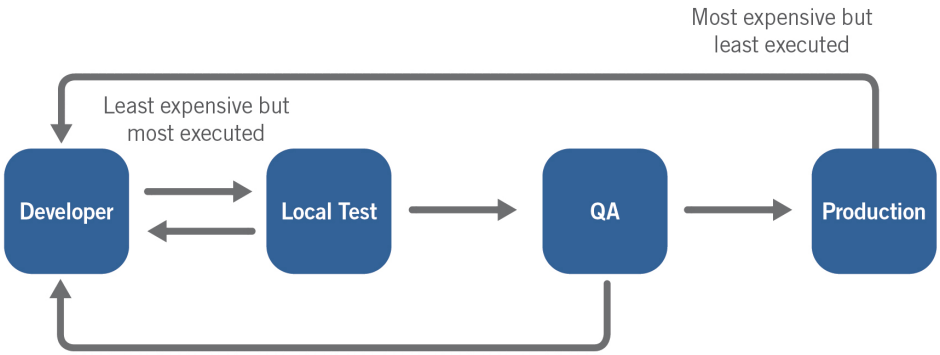
- Autonomously try new ideas and make architecture changes.
- Run concurrent (parallel) environments for A/B comparisons.
- Create, QA and build environments without having to wait for resources
- Fearlessly break and recreate dev and test environments without worrying about consequences.
- Instill changes in the overall production environment while minimizing friction between developers and IT at every step of the process.

Efficient code delivery pipelines that balance security and agility can be achieved by using cloud infrastructure, Docker containers and automation.

This paper is designed to show technical managers, operations professionals and developers the attributes of a well architected framework for continuous delivery and how it contrasts with a poorly designed code pipeline. We'll also show how individual pieces of the pipeline can be implemented and orchestrated.

How Optimized Pipelines Deliver Value

Code delivery starts from a feature or bug request filed with a developer. The developer takes existing code, modifies it, does local testing, and commits to the code repository. The code in the repository is then tested by QA and tagged for production, where IT or operations professionals deploy the code to production servers.



The diagram above describes the full code delivery pipeline.

This creates loops at the local testing, QA, and production stages of the process. Each of these loops has a cost, directly affected by developer productivity and how long it takes to execute each loop.

The local testing loop executes most frequently, so it's critical to make this part of the testing process fast. Production bugs are the least frequent but most expensive for businesses when they exist. Consider the cost of a significant disruption to the workflow, or the negative customer experience, or the inability to deliver a service, or security vulnerabilities. Imagine if you could keepi delivery time and costs under control while minimizing total effort at *every* stage.

Companies wanting to accelerate delivery of production-ready code need to optimize their development environment by automating their processes. Optimized development pipelines deliver significant value by minimizing the time and costs of:

- Local test, QA and production processes.
- Promoting code from one development stage to the next.

When you engage in this process, you reduce the time it takes to identify problems and their causes. When you automate development steps, you dramatically reduce the cumbersome and time-consuming tasks that slow down the release of software code.

Flux7 Approach to Optimized Pipelines

When development environments are optimized, the following attributes are occurring at each stage of the process:

Code to Develop

Committing code often: Micro-agility is achieved when developers test and commit frequently. This approach means that feedback from QA is receive every few minutes rather than once a day and bugs can be tacked and quickly addressed.

An unambiguous process: The latest development is clear, and it's clear where developers commit their code after making changes.

Local Test Loop

Quick to run a test locally: Test should be easy to fully run locally, and is run as a subset of regular QA testing.

The law of PQR: The development environment is:

- **Production-like.** Development and testing environments mimic the production environments very closely.
- **Quick:** Fast and easy to set-up; push button deployment.
- **Repeatable:** Recreate environments that will produce the same results every time.

Promoting Code From Development to QA

QA Ready. Any code pushed by a developer automatically becomes ready for QA.

QA Loop

Automation and Reporting. QA is automated or streamlined and executed in small batches rather than in multi-week long cycles. Reports from QA are communicated to developers immediately. Commits that fail QA are rejected automatically.

Promotion From QA to Production

One click. Promoting to production is a one-click process. There's no ambiguity in which code has gone through QA and is ready for production. Promotion can be achieved through tagging, if needed.

Production Deployment

Full Automation. Deployment is fully automated. It automatically applies any configuration changes made by the developer and approved by QA. Deployment does not lead to system or application downtime and i human errors are greatly reduced.

Production Iterations

Monitoring. Monitoring and alerts should be implemented so that you (and not your customers) are the first to know about errors. In production, code is monitored for off-specification system behavior and components that don't function properly.

Docker in 15 Seconds

Like a virtual machine:

Encapsulates all app dependencies and runs on any platform where Docker is installed.

Like text in a Git repo:

Transports easily.

Like running apps natively:

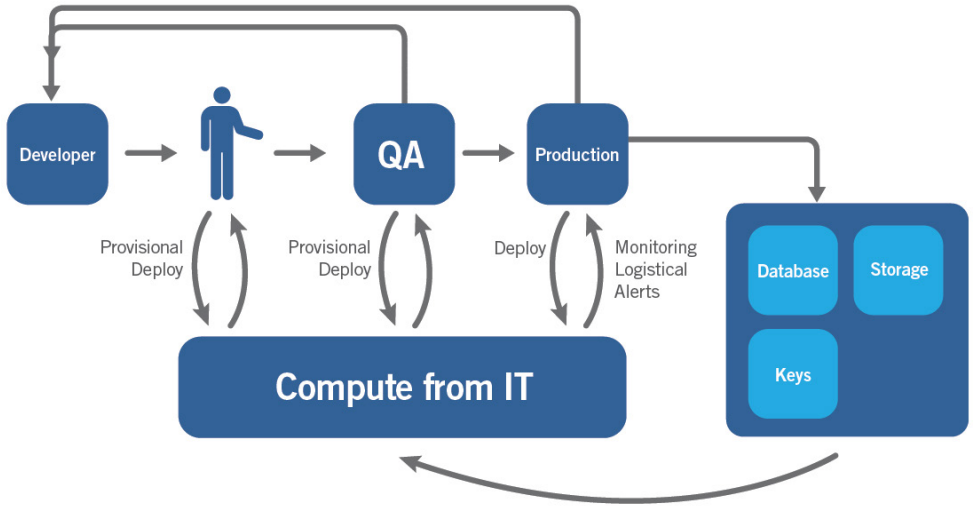
Run with little to no performance overhead.

Alerts. Monitoring is nearly useless without alerts. But, too many alerts are as bad as too few. In production, every alert should lead to an action, which fixes a problem. If there is no problem, a clearly written note of investigations performed should be made. Then, alerts are updated to prevent them from firing again.

Alerts are classified by three preset rules:

1. Alerts that are handled automatically but tallied.
2. Alerts to be fixed by the ops staff.
3. Alerts in application logs that should be sent to developers.

Normally, production servers are not changed manually unless drift monitoring is used.



Case Study: Dev Workflow as a Modernization Test

Modernizing IT and increasing automation across the organization and is common goal for many industries. A large US healthcare software vendor with on-premise IT was interested in modernization, but wanted to start the process by focusing on a discrete area where there was low risk and a high return on investment was expected: optimizing the development workflows.

- Provisioning infrastructure required manual intervention by operations, and caused developers to wait until compute power was provided.
- To work efficiently, many users needed access to development and QA environments at the same time.
- Healthcare information compliance regulations required them to generate audit trails as changes were made to the company's code or infrastructure.

They needed a solution that would help them:

- Build automated controls for provisioning and release to production.
- Automate entire delivery pipeline from development to production.
- Provide a proof of concept with a sample application.

Solution

The optimized developer workflow provided an agile, scalable development and QA infrastructure. Originally, because of compliance concerns, the company was reluctant to move from on-premise infrastructure.

The final architected solution implemented the same level of controls on-premise while gaining the agility of the cloud and leveraged Docker containers to provide self-service IT capabilities to developers. As the development and QA teams grow, cloud-based services could be scaled to support development activities, without the need for extra capital expenses or over-purchasing and under-using compute power.

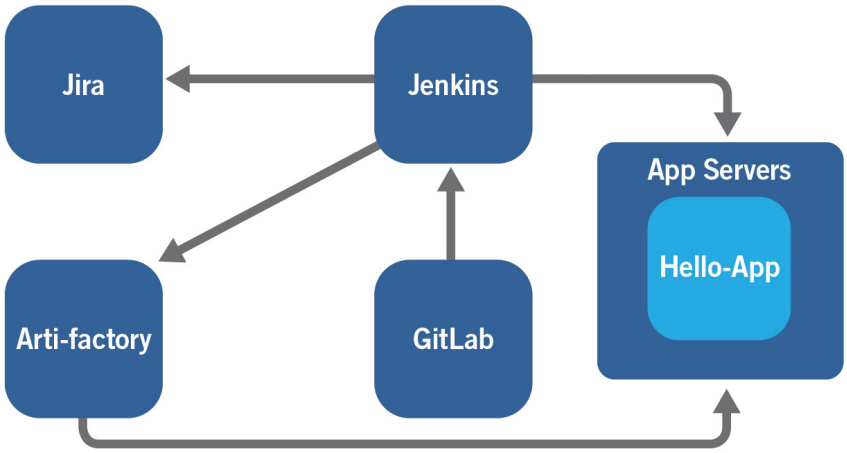


Diagram 2: The new, optimized code delivery pipeline

Now, the new development and QA infrastructure and processes are managed mainly by scripts, not engineers. The new environment generates an audit trail and provides a dashboard that makes monitoring and alerts easy to view and manage.

The optimized pipeline provides support for a rapid deployment strategy and new agility that resulted in higher productivity from developers. The new environment leverages cloud and containers to speed time to market of new applications and features.

Conclusion

- Re-architected code delivery pipelines and leveraging modern technology like containers and cloud, as appropriate, can serve as high ROI investment.
- Creating self-service IT can relieve pressure from operations and enable development to proceed at it's own pace. The code delivery pipeline helps remove obstacles slowing software release.
- Code delivery pipelines are a critical business area for maintaining business advantage and productivity.
- Development processes present a low-risk, high ROI area to test modernization technologies and techniques.
- Dev and test modernization is increasingly a business requirement. When implementation is strategic and based on best practices, it can serve as a POC for other projects.

Find out more about Flux7 continuous delivery code pipelines [URL TBD]

About the Author

Aater Suleman, Ph.D. is co-founder and CEO of Flux7, an Austin-based IT consultancy. He is recognized internationally for his innovative work, speaking engagements and published papers on computer engineering.

Suleman uses his extensive background in computer hardware, performance optimization and software development to create self-healing cloud infrastructure frameworks. These products enable organizations to use infrastructure as a service with minimum effort and cost. A passionate researcher and active mentor, Suleman continues as a member of the faculty of his alma mater, the University of Texas, Austin.

About Flux7

Flux7 is a team of IT experts, who help businesses optimize the benefits of IT. Flux7 DevOps processes combine cloud-based services, Docker containers and process automation.

By using DevOps processes, Flux7 can help your company set up efficient developer and IT workflows. We can help you define your current business requirements and technical state. Then, we work with your team to design a desired state that accomplishes your business goals.

For more information about Flux7, visit **www.flux7.com**. Or, contact us at **info@flux.7.com**.