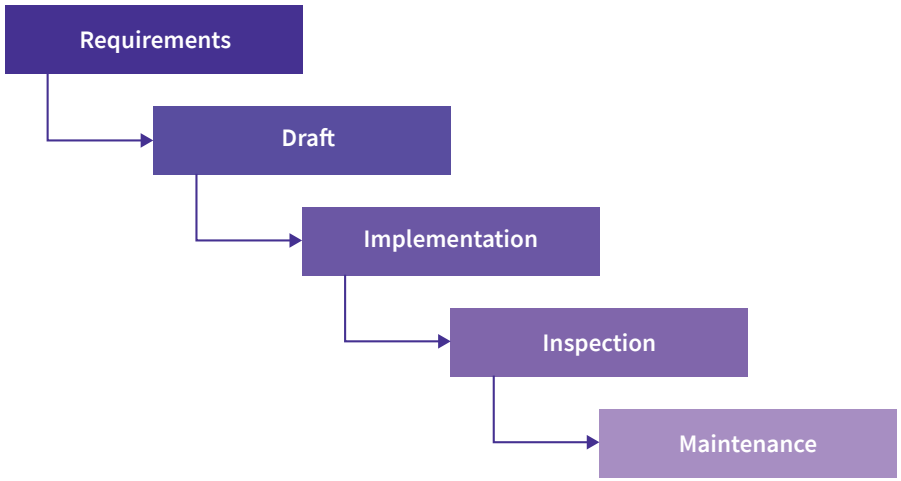
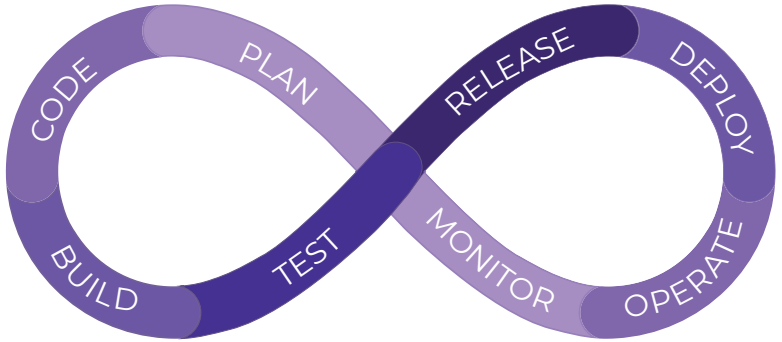
OPSERA

CI/CD Best Practices for Stress Free Software Development

Modern development cycles bear little resemblance to their predecessors in today's fast-paced release-or-fail environment. Old models included design, develop, and test phases and tended to be major revisions with extended development cycles between releases. Each release could include many bug fixes, major UI changes, and/or new or improved features. As a result, customers would have to wait a long time before receiving an update with a much desired feature or change. Fast forward to today's app- and cloud-centric environments where continuous improvements and short release cycles are the norm. This has dramatically reshaped the software development paradigm. Through this we have seen the rise of DevOps, and most recently DevSecOps, to help enable these rapid deployment cycles.

This shift has moved us from a linear cycle to the continuous loop model of modern DevOps





What is Continuous Integration?

To understand what continuous integration is, let's take a look at what it is not. In standard practice, a developer will checkout code from a repository to make their changes. Legacy waterfall development cycles could mean code checked out for extended periods of time, maybe with single daily check-in. Integration would only be run at the completion of changes, whether adding a new feature or making changes to existing code.

The challenge with this methodology is that the code does not regularly sit in a deployable state and is prone to errors or failures when integration is finally (manually) performed. Tracking what code blocks are causing which failures can be time consuming and challenging. With continuous integration, all of this changes as we shift towards an Agile methodology.

Code changes are checked-in multiple times per day, and every time, automated integration is run to check for errors. This means tracing the source of an error is straightforward and can be quickly corrected. Frequent integration means faster feedback, as well, allowing developers to stay on task and focused on their work. And at the end of the day, your application is always deployable (even if incomplete).

What is Continuous Deployment?

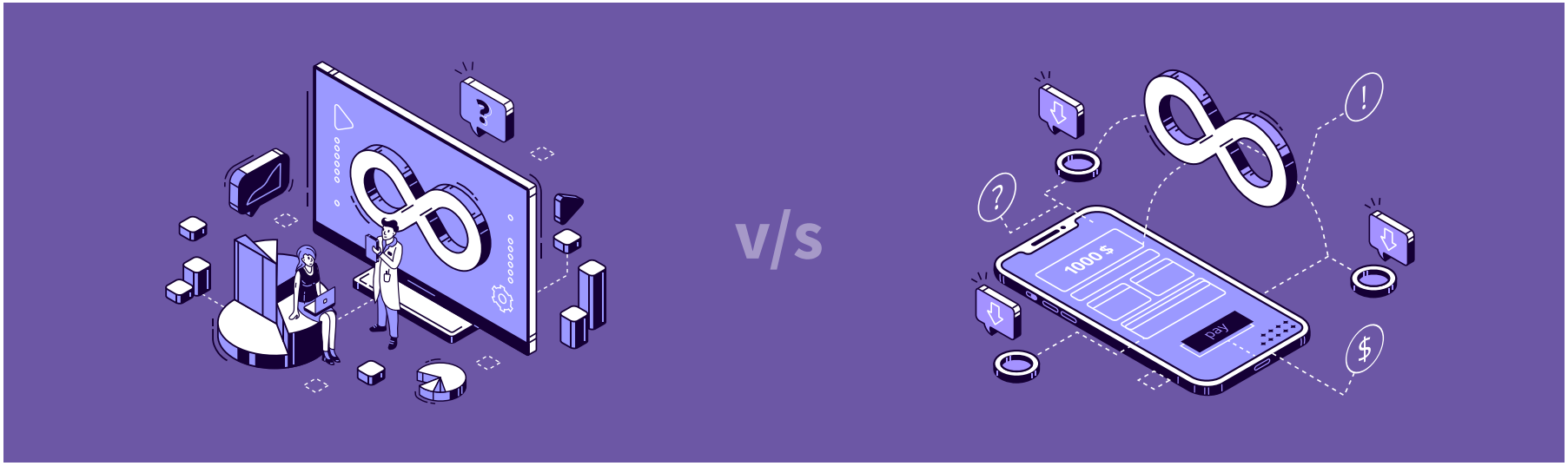
Now that we've streamlined and automated aspects of integration, let's look at how deployment functions in a continuous model. **Continuous delivery** is the natural next step in automation. Once your code has been merged with the master branch and freed from errors, it's time to push master branch code to production. They key here again is automation of your pipeline.

The challenge is having the right tools to test and verify the code, not just for errors but for function and security as well. These checks must pass in order for the code to flow from pre-production to production in a ready-to-be-deployed state. This stopping point is a key differentiator between continuous delivery and it's nearby neighbor, **continuous deployment.**

Delivery Versus Deployment
Which Continuous Model to Use

Continuous deployment is the successor to continuous delivery, in which we add an additional phase of automation to the pipeline. Delivery leaves the code in a ready state for deployment, but stops automation here - the deployment step is manual, in that someone has to push the proverbial button to make it so. With continuous deployment, the creation of, and deployment to, the appropriate infrastructure is all automated as part of the pipeline. The automation stream no longer stops and waits for a push, the pipeline can now take the code from the repository, pull the appropriate configurations, build VMs, containers, etc. on the fly, and deploy the code, all in one fell swoop.

Though this model truly allows for efficiency and speed, the challenge is having the proper tools and checks in place to manage your configurations and rollback in the event of errors or failures.



Bringing it all Together: The CI/CD Pipeline

Now that we've looked at both halves of this equation, we can bring them both together to build our DevOps Continuous Integration/ Continuous Deployment Pipeline. **The critical steps are:**

A continuous orchestration platform can help you design such a pipeline - like Opsera's Continuous Orchestration

The major difference between legacy waterfall development and agile is that instead of doing all the development, then merging all the code into a final build, then finally testing, developers are consistently looping through the develop-build-test cycles. Every code integration, multiple times per day, aka continuously. At the end of the day, the code could theoretically be deployed as-is. So how does this achieve more frequent releases?

The hallmark of a truly efficient pipeline is seamless integration of tools that allow automation from end-to-end through each of these phases.

Checking in code more frequently alone is not what speeds release cycles, but certainly plays a critical role. The idea is to break down features and new development into byte-sized pieces that are fast to write. These ultra-fast cycles paired with continuous integration achieve the following:

1. If integration errors occur, it's easier to determine their source and quickly correct the issue.
2. Code is consistently being deployed to a production-like environment so we have built-in assurance of how the code will work for end-users.

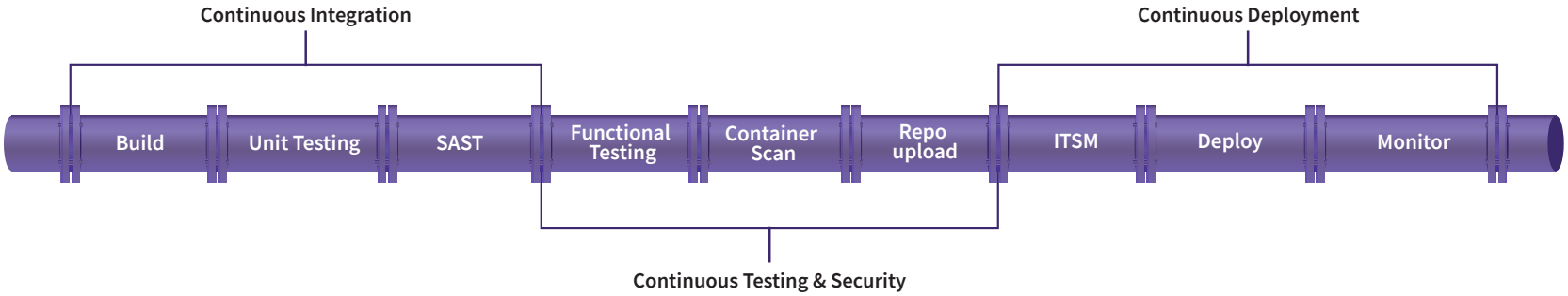
Automation is the Secret Sauce in CI/CD pipelines

A CI/CD pipeline achieves speed and efficiency with automation. Every step in the pipeline should have the appropriate tools and automation **baked-in.**

Now let's take a step back and look a bit deeper at the phases of each CI and CD and their workflows and automations.

As we saw previously, the 4 overarching phases are:

1. **Commit**
2. **Build**
3. **Test/Quality**
   - Unit testing
   - Integration testing
   - Regression testing
   - Performance testing
4. **Security Scan**
   - Static code analysis
   - Container scan/TVM (before staging)
   - Dynamic testing
5. **Deploy**
6. **Operate**
7. **Monitor**



Now let's look at the tools and mechanisms at play in each phase.

Develop

Here we are writing the code, meaning that we need a repository for storing and checking out code. Since most development projects involve teams, team members have to edit the same files, which increases chances of mistakes in the codebase. To avoid this, developers use version control.

Commit

With version control, developers don't keep just the latest version of the code. They can save the entire history, including all changes made by every team member. This makes it easy for them to restore or refer to earlier versions, in the event of any mistakes or anomalies.

Once a developer has worked on a patch of code, they add it to the existing codebase, which changes its history and structure.The act of addition is what it means to commit code.

To commit code, we must use version control software, the most popular of which is Git. It is exceptionally useful, and lets devs do a multitude of things, such as running experiments without messing with the codebase. All devs have to do is create a new branch, which serves as a separate copy of the code. If it works, they can integrate changes into the main branch.

Build

When code is checked-in to the repository, that code is integrated into the master branch. Here is another point where version control is necessary. In older waterfall style workflows, integration/build would only occur after the completion of a major feature set or large volume code changes.
When working in a more agile, continuous model, code is integrated into the feature times per day. Code is broken down into smaller working functions, allowing for iterative improvements and fast integration results.

Test/Quality

Once the code is built, the resulting application must be tested for errors, functional failures, and quality. These tests can and should be automated using any number of purpose-built tools. By testing at every build, feedback is received quickly and corrections can be implemented swiftly.

Different tests are run at different stages in the CI/CD pipeline. Generally, teams and organizations have processes in place for code review. To start with, **unit tests** are run before pushing the code to the central repository.

Once integrated with the master branch, the entire codebase (now changed) is run through **integration tests**, in order to verify that multiple software modules work in tandem without disruptions.

Additionally, **regression testing** must be executed to check that changes to the codebase has not been broken or scrambled by recently added code. As mentioned earlier, version control software lets you create a separate copy of the existing code. Do that, run necessary tests and if results are favourable, merge new code into the main branch.

Performance tests, of course are necessary to verify operational metrics for the software at hand - speed, stability, reliability, response time, scalability, resource usage. Software cannot be considered ready for deployment if these functional parameters do not match minimal technical requirements.

Security Scan

As part of our transition away from bolt-on and after-the-fact auditing, security scanning is a critical part of the continuous loop. Security scans can automatically detect vulnerabilities and insecure implementations before being released into the wild and exploited by bad actors.

Include **static code analysis** in this stage.This refers to the examining of source code before it is run, by analyzing it against a set of predetermined standards for functionality. It is best to run static analysis right after writing the first batch of code, before unit tests are run. While this can be executed manually, the process is too cumbersome and prone to error. Automating it is the way to go.

Run tests to ensure container security, which verify that everything in the container in usage (relevant executables, binary code, libraries, configuration files) are running in expected order. This encompasses the infrastructure, runtime, supply chain, runtime host, application layers, platform etc. **Container scans** are an optimal way to ensure this outcome, which need to be automated and built into the CI/CD pipeline. The container scan confirms that containers are technically reliable, secured against threats, free of common security issues, and fit for deployment to production. It mitigates risk and reduces vulnerability to possibilities of external attack or internal malfunction.

Bring security scans into the purview of **threat and vulnerability management**. Given the increased instances of cyber attacks (DoS, DDoS, MitM attacks, phishing, SQL injections, and so much more), development teams and organizations backing them must assess endpoint weaknesses and identify means to reduce risk. Threat and vulnerability management provides the blueprint for steps taken to reduce security risks and establish higher software resilience.

It is essential to include this step in the testing portion of your pipeline - as they say,

“an ounce of prevention is worth a pound of cure.”

Deploy

The code is built, testing has provided the green light, and it's time to push our changes to an environment, whether pre-production or production. With continuous deployment, regular automated push to non-production environments gives clear feedback and metrics for how the code will perform when released to a customer-facing environment. After passing the appropriate tests in pre-production, code can be automatically released to production using whatever method or model meets your requirements.

Operate

Now that the software is live, the Ops team is knee-deep in activities to ensure that it is functioning smoothly and adequately meeting user expectations. The configuration of the hosting service, in particular, determines how software functionality scales in response to peaks and lows in terms of active users at any time.

Additionally, the Ops team has to man the feedback mechanism built into the application, collect customer opinions and push them back to developers so that they can start working on and rolling out software updates. Regardless of how much research went into software development in the beginning, some features & functions might be missing.

Remember that your customers are the finest testing team in the world. They will test your software with greater scrutiny than most QA teams, and their opinion is the only one that matters for your application's success.

Monitor

The final phase of the CI/CD pipeline actually functions in parallel with the Operate stage. Feedback is collected and analytics performed on user behaviour, software performance, errors, etc.

This stage also requires examination of the CI/CD pipeline itself. Check for bottlenecks that may slow down the process, adversely impact its efficacy, or frustrate the developers.

This data is forwarded to product managers, QA managers and development heads so that they can instill improvements in the process. This progression remains continuous, because even if a project ends, learnings from it are used to strengthen development and operational mechanisms in the future.

Who am I and Why Should I Care About CI/CD?

CI/CD matters at every level of the organization, across functional and operational teams. The efficiencies realized by the continuous model allow business to move forward at speeds never before possible, while maintaining quality and integrity. Different members of the organization will find unique benefits depending on their role, and the value to the overall organization will be greater than the sum of its parts.

Developer

For developers, the value in CI/CD is making it easy to deliver clean, high quality code by integrating frequently and resolving errors as they happen instead of debugging a multitude of disparate bugs across the codebase days, weeks, or months after the code was originally written. It also replaces manual steps with automation, and speeds the pace of feedback, so it's easy to stay focused and accomplish deliverables. With deployments happening as part of the continuous loop, developers are no longer divorced from the infrastructure hosting their code and deeper ties with IT and/or Operations are forged.

DevOps Engineer

Older development and deployment strategies treat the code and its underlying infrastructure (as well as security) as separate. With DevOps, and more importantly, DevSecOps, and the CI/CD pipeline, these all become unified and designed

to work together. Security is no longer an afterthought, infrastructure is right-sized and purpose-built to deliver the application to best advantage. Again, all of this is automated at deployment, often using Infrastructure-as-Code configurations, microservices, and containers. There is no longer the tug-of-war between developers and infrastructure engineers when trying to determine who owns the problem when errors or failures occur. When these functions are more deeply integrated, collaboration becomes the name of the game instead of “hot potato”.

Head of Engineering or Applications

When we think about CI/CD, we often look at this from the standpoint of code, tools, integrations - all the technical aspects. But what about management - how does CI/CD directly impact business leaders such as the CIO? CI/CD offers a multitude of benefits from a planning and resource management perspective. Business leaders are always looking for ways to increase operational efficiencies while reducing costs. Costs and complexities can both be reduced with a well designed CI/CD pipeline. With fewer manual steps, teams can work more efficiently and effectively with less resource overhead. Not to mention doing away with huge time sinks and wasted efforts. With stronger team-to-team collaboration, less time is wasted on tracking or assigning ownership to issues and overall technical and business processes are streamlined. It becomes easier to plan and predict future needs and prepare the business to scale.

Customer

In the end, all of these efficiencies and automations don’t mean much unless there is a customer on the other side reaping the true benefits. In the case of CI/CD, frequent updates means short waits between improvements and always having access to the latest and great iteration of the application or platform. Especially with subscription-based software licensing being the norm, customers expect frequent improvements to justify the ongoing expenditure. CI/CD enables not only the fast-paced release of features, but helps ensure the code that is delivered is functional and that bugs are few and far between. New features can be painfully overshadowed by a poor user experience and well-executed pipeline can prevent customer dissatisfaction by keeping quality high.

Solving the DevOps Conundrum with no-code DevOps Orchestration

In order to remain nimble and, dare we say it, agile, in today's lightning-paced environment, CI/CD is a much needed DevOps strategy. The benefits to the organization from top to bottom are unparalleled. Not to mention the many benefits to more highly satisfied customers. The common saying "if you have to do it more than once, automate it" has never been so true in the software development life cycle as it is today. Achieving success with CI/CD means having the right tools in play to support your shift left goals and efforts.

There is no easier way to build and maintain a flexible and high-quality pipeline than Opsera's Continuous Orchestration. Build drag-and-drop declarative pipelines, with security and quality piped in. Your toolchain, your preferred vendors,, with no coding or manual integrations. Unified insights help drive business decisions with aggregated and contextualized logs, personalized dashboards, and over 100+ KPIs to help measure your success.

"The investment into Continuous Orchestration will not only pay back in dividends but also in the empowerment of developers to ship core features, faster, and with fewer defects."

- Declan Morris
Former Splunk CIO

"Opsera DevOps continuous orchestration delivers easy automation of release management across the enterprise applications. Comprehensive visibility from the platform helped us decommission legacy release management tools helping reduce the Opex."

- Dayakar Duwuru
Sr. Director of Enterpris applications, NortonLifeLock

OPSERA

About Opsera

Opsera's platform allows you the freedom to choose your own DevOps stack - with zero scripting involved. With Opsera, you can automate any stack, build pipelines forany app, and deploy anywhere.
Opsera accelerates best-practice Cl/CD adoption so software teams can deliver software faster, safer, and smarter.

opsera.io

Accelerate your software delivery with no-code CI/CD pipelines

Schedule A Demo