

Security at the Speed of Software Development

Implementing a cultural sea change with DevSecOps



Author

Larry Maccherone is an industry thought leader in the DevSecOps movement.

Maccherone published the largest ever study correlating practices with performance, which has been a cornerstone project of his career.

Still an active developer, Maccherone has 12+ projects he manages, and encourages developers to stay active in development, even as they ascend the hierarchy.

Maccherone is a Distinguished Engineer at Comcast and is leading the DevSecOps transformation initiative within the organization. bdelivery process.



What is DevSecOps

DevOps has become a well-known concept in the software world over the last several years, but the focus on simply marrying the development and deployment aspects of the pipeline fall short on a critical piece that can no longer be an afterthought: security.

DevSecOps is the integration of security checks and practices built directly within the development pipeline. The DevSecOps movement is an opportunity to re-introduce the concepts of security in a way that fits seamlessly within modern development practices and pipelines.

This represents a sea change from older DevOps methodologies that treat security as an afterthought and handle it as such. Market research suggests DevSecOps will grow at 30+ percent per year for the next seven to eight years.

Development teams must reposition their existing offerings and development practices to “bake in” security to protect against technical compromises. Further, leaders in development-oriented organizations must make traditional security concepts sit with the teams that will directly benefit from DevOps.

The old bolt-on way of thinking about security does not scale. The rate of change of software is accelerating faster than the rate of bolt-on security solutions, and security specialists simply can't keep up with a gating mentality.

Problem Statement: We must get development teams to become involved firsthand in the process of securing the products that they're building and pushing into production.

The element of ownership is a key driving factor. DevSecOps means empowering engineering teams to take full ownership of how their product performs in production, not simply handing it off to another team to worry about.

You can't just tack on security at the end. DevOps is about building security elements into the foundations of the solution; not just in writing the software but also how it performs in production.



Where Did We Get DevOps Wrong?

Many project managers and engineers are quick to push DevOps as a value add for their clients, but few actually know how to implement it successfully.

The phrases “DevOps” and “DevSecOps” are marketing waves and have led to the practice being significantly over-marketed, without the execution to back it up.

Many of the market offerings don't fit well with the core philosophy of DevOps, which we'll get into in this eBook.

DevSecOps has marketing built into it because it is simply the future of how development should be approached. **Marketing should never precede functionality, especially in regards to security.**

Development has traditionally been viewed as somewhat modular: one team does one role and passes the project along to another team, with security elements simply being “bolted” on at the end. **So many of the security risks are centered in operations, that we're severely neglecting our ethos as developers by taking subpar products to market.**



The DevOps Continuum

The DevOps continuum is an infinity loop of the most pertinent development sequences to ensure the creation of the highest-quality and most-secure product possible.

There are a few elements to become intimately familiar with, but let's focus on the Analyze and Learn phase of the continuum as an example.

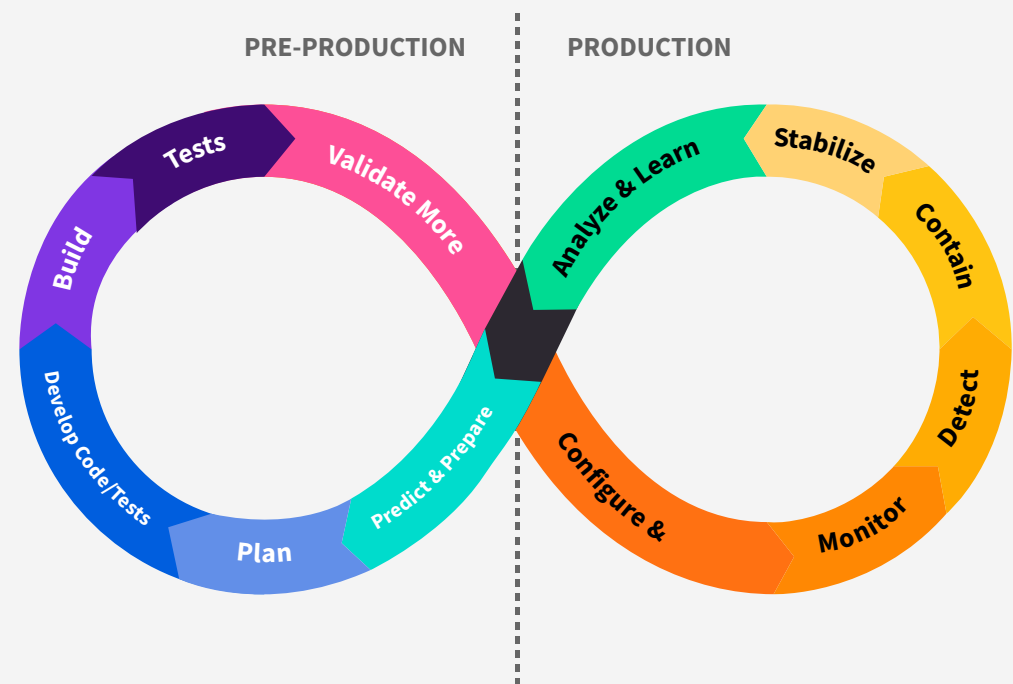
The traditional model of gaining products through a security review into production is flawed. The problem is that when independent security audits are done after the product is shipped into production, and not embedded in the initial and ongoing development process.

When you get rid of the gating function of security assessments, and you switch it to an out-of-band process, then its value becomes one of learning.

Security practices on DevOps continuum



DevSecOps



The Defect-Incident (or Vulnerability Three-Step)

For example, teams should implement some manual code review of changes periodically, and especially before launch. Suppose you don't do an independent security code review from a security professional's point of view before pushing to production. In that case, you're exposing yourself to the threats in the future.

The idea is there are **three things that you should do when you find a defect**, have an incident, or your security assessment team comes back with a list of vulnerabilities.

Step #1: Fix those particular vulnerabilities (but you shouldn't stop there).

Step #2: Try to discern a pattern that matches this vulnerability.

1. How did this happen?
2. How does this manifest itself?
3. Are there any other places in your current product, the one that went through the assessment, and other products you're responsible for that have this vulnerability?

Step #3: look for a way to change something that will prevent this vulnerability from happening again. Change your technology, change your testing, suite, training, whatever will prevent that particular problem from ever occurring again.

Defect-Incident In Practice

For example, let's imagine you've identified a buffer overflow as the source of a serious defect or vulnerability.

In the past, when almost everything was C++ in production, developers found that by switching over to Java, they were able to eliminate the possibility of a lot of buffer overrun type vulnerabilities.

That was a dramatic technology change to prevent a whole class of very dangerous vulnerabilities from ever occurring.

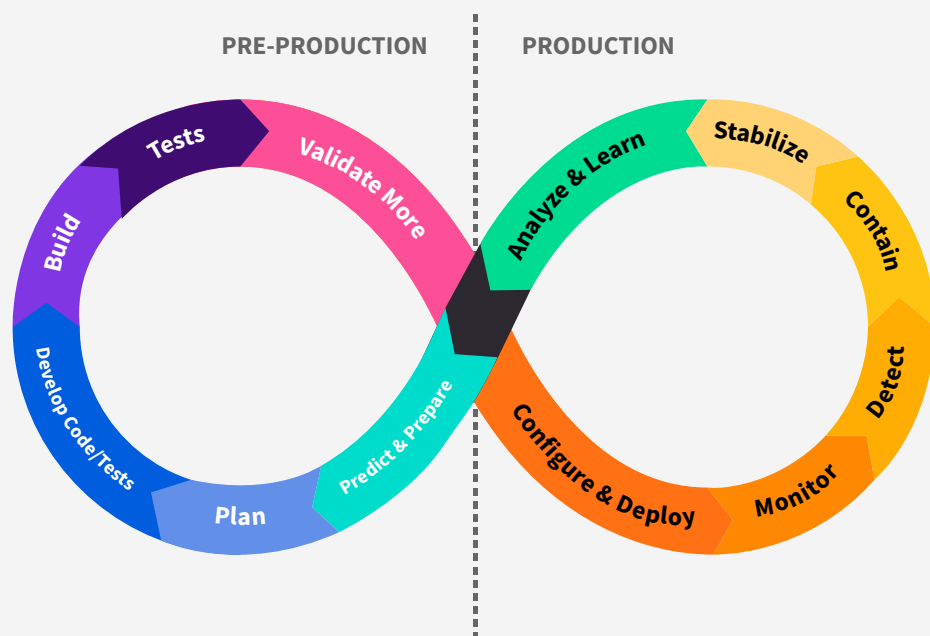
Most of the time, you don't need to make that dramatic of a change, but you should think about and try to come up with a preventive measure that will be effective going forward.

By using Defect-Incident (or Vulnerability Three-Step), you'll be able to effectively diagnose and solve the issue at hand.

The DevOps Continuum (part 2)

The fully fleshed out DevOps Continuum is a lot of stuff to consider, and many DevSecOps leaders have this printed out over their desk, but studying the steps here isn't the hard part...

Security practices on DevOps continuum  DevSecOps



PRE-PRODUCTION

Predict & Prepare:

- If we do X, will it mitigate Y?
- Capacity forecasting
- Learning -> Update playbooks and training.

Plan:

- Threat modeling -> backlog items
- Analyze/predict -> backlog items
- Does the design comply with policy?

Develop Code/tests:

- Static/IAST analysis
- Abuse case tests
- Code Review

Build:

- Interrupt-the pipeline code analysis

Test:

- Test security features
- Common abuse cases

Validation more:

- Pen testing (Vuls found -> test scripts)
- Compliance validation (PCI, etc.)
- Fuzzing.

PRODUCTION

Configure & deploy:

- Configuration validation
- Feature toggles/traffic shaping configuration
- Secrets management

Monitor:

- Log information for after-incident analysis

Detect:

- Intrusion detection
- App attack detection

Contain:

- RASP auto-respond
- Roll-back or toggle off
- Block attacker
- Shut down services

Stabilize:

- Restore/maintain service for non-attack usage

Analyze & Learn:

- Analysis -> Learning
- Defect/Incident 3-step
- New attack surface? Plan to update the threat model

The Hardest Part of Instituting DevSecOps

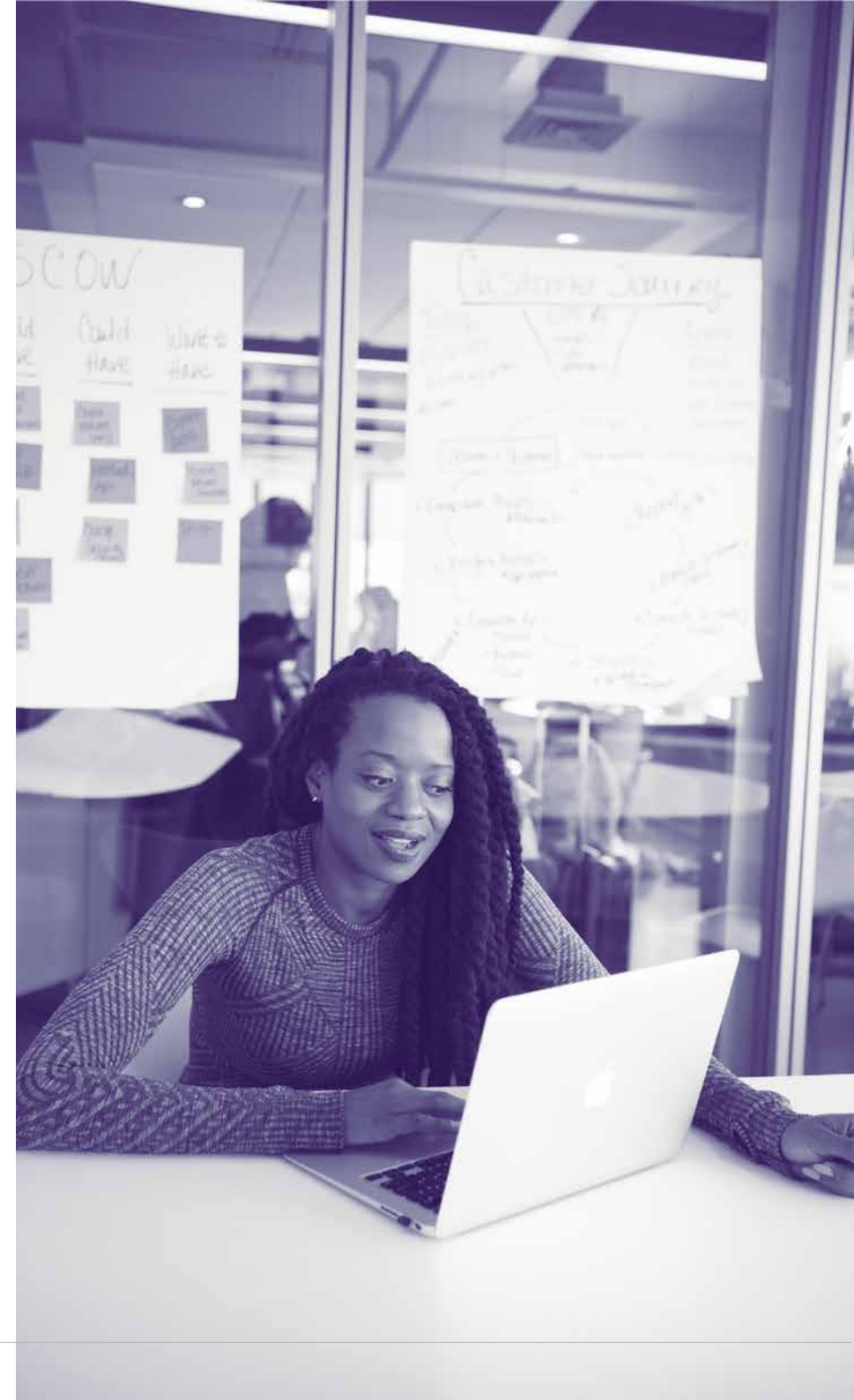
The most difficult part of pushing DevSecOps within any organization isn't memorizing a framework; it's **winning your development teams' hearts and minds to believe in, and commit to, the DevSecOps mission.**

How do we get engineering teams to adopt DevSecOps at a foundational level?

How can we institute behavioral and cultural change at lasting scale?

Coming up with a list of things is easy while getting people to adapt and change is the hard part. That's really where the magic of the DevSecOps framework comes in.

We've built a three-part framework for adopting new DevSecOps practices into development organizations.



What is the Three-Part Framework?

This framework is the result of the largest ever study correlating practices to performance conducted by Larry Maccherone during his time at Rally Software. The framework was initially designed to help develop agile teams, and Maccherone eventually adjusted this framework to modern DevSecOps from his experiences working with large companies such as Ford Motor Company, USAA, and Comcast.

Step #1: Win the Hearts and Minds of Developers

You have to have credibility. You have to have trust. You have to win the hearts and minds of developers.

This can be hard for security specialist groups because the typical relationship between the security group at large organizations usually has a little bit of tension.

You're never going to win the hearts and minds of developers with a lack of trust, so instilling a DevSecOps transformation becomes a function of trust.

There are a number of tools that we give to security specialist groups that will help them win the hearts and minds of developers. One of those tools is the principles labeled the DevSecOps manifesto.

The DevSecOps manifesto

There are alternatives to the DevSecOps manifesto, but this is the one that we think fits our particular approach the best. It's formulated in a format such that many development teams will recognize something more than something else, in exactly the way that the agile manifesto is formulated.

1. Build security in more than bolt it on.

2. Rely on empowered engineering teams, more than security specialists.

3. Implement features securely more than security features.

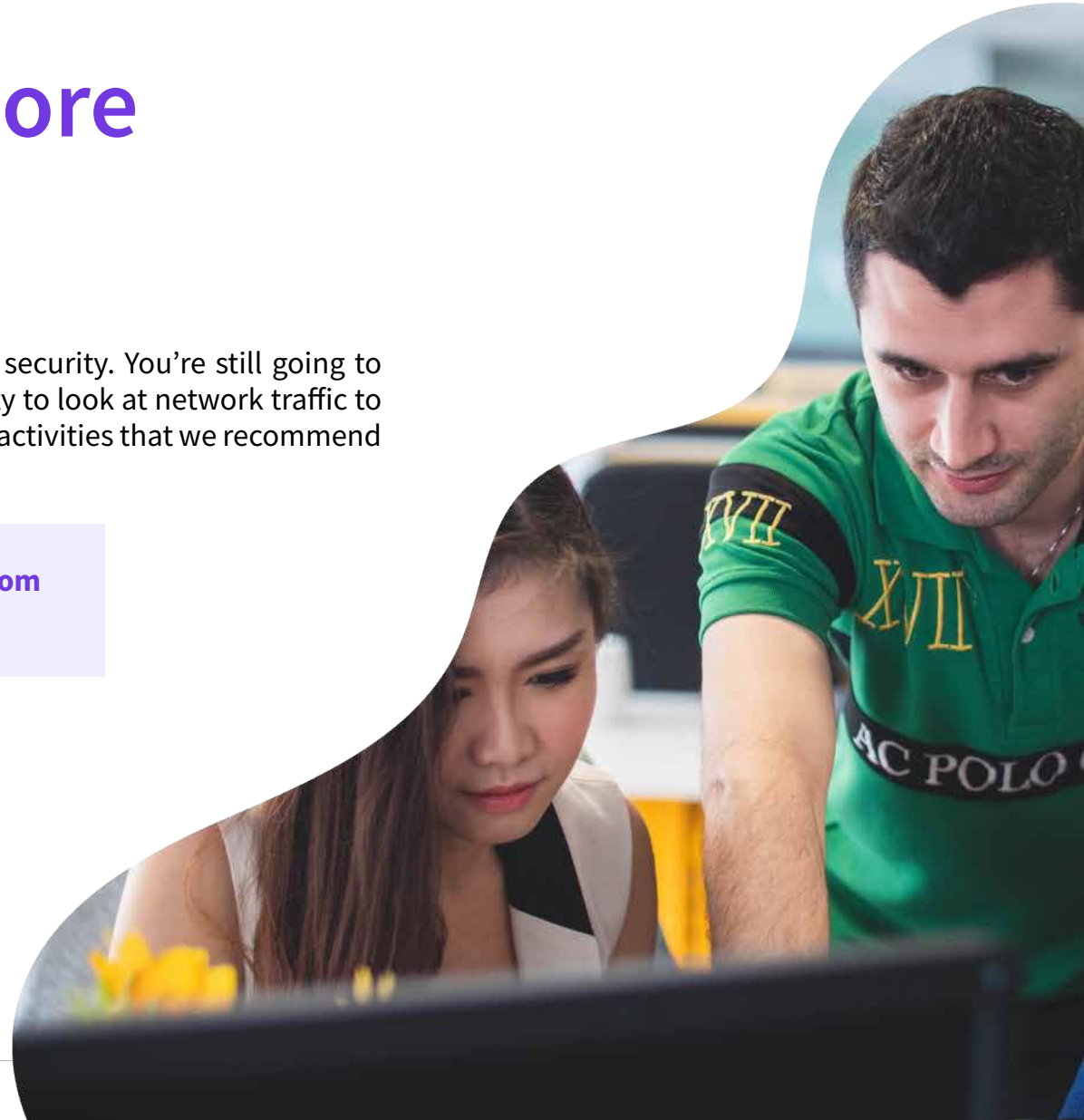
4. Rely on continuous learning more than end-of-phase gates.

5. Build on culture change more than policy enforcement.

Principle #1: Build security in more than bolt it on.

This isn't a marching order to stop doing any bolt-on on security. You're still going to have networks and firewalls, and you'll still have the ability to look at network traffic to identify bad actors, and a wide range of other bolt-on type activities that we recommend continuing to use.

This principle puts the emphasis on building security from the get-go.

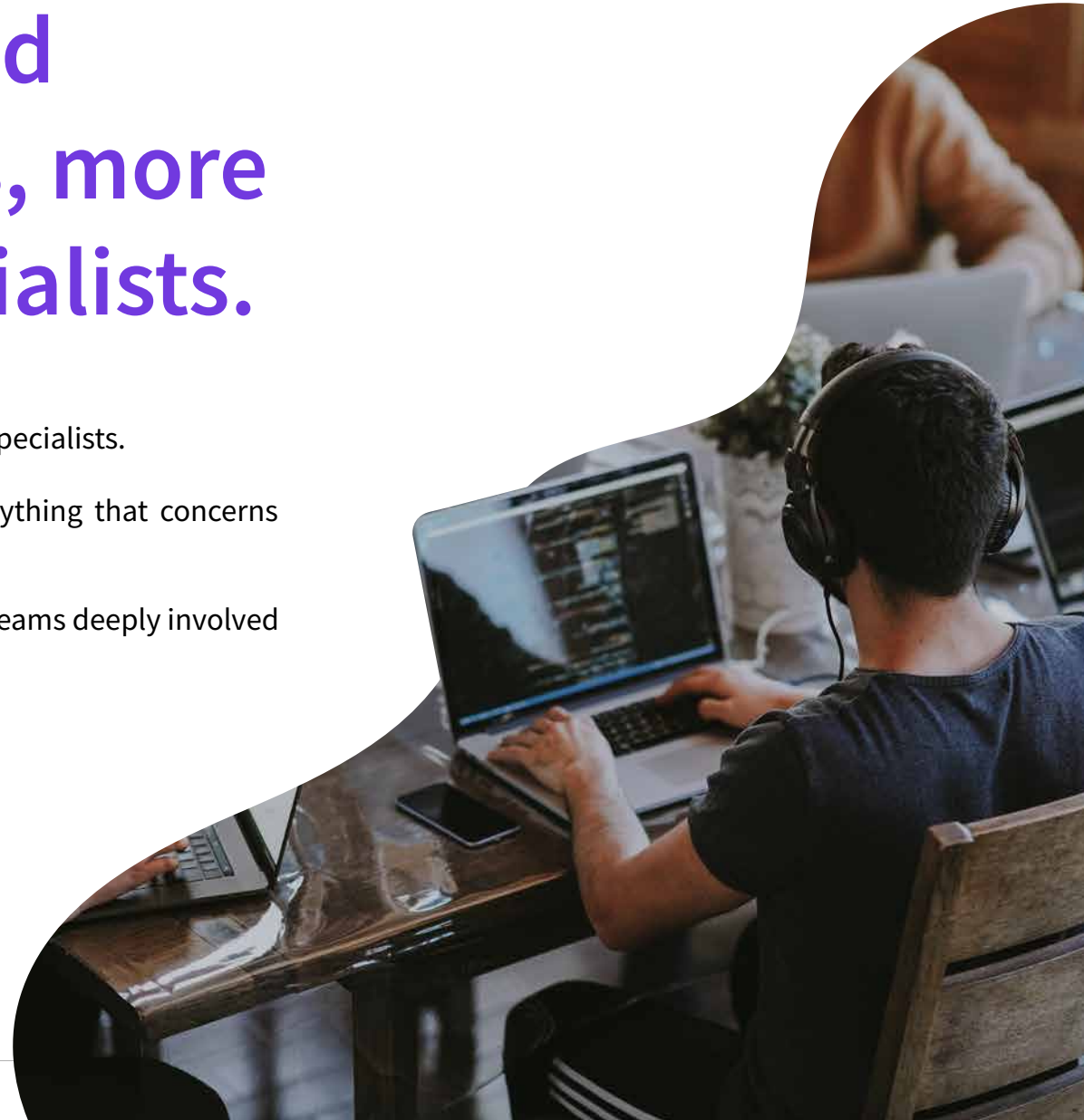


Principle #2: Rely on empowered engineering teams, more than security specialists.

You will never achieve optimal security with just security specialists.

We have to have the engineering team involved in everything that concerns implementing features securely.

The only way you can do that is to have your engineering teams deeply involved in the process.



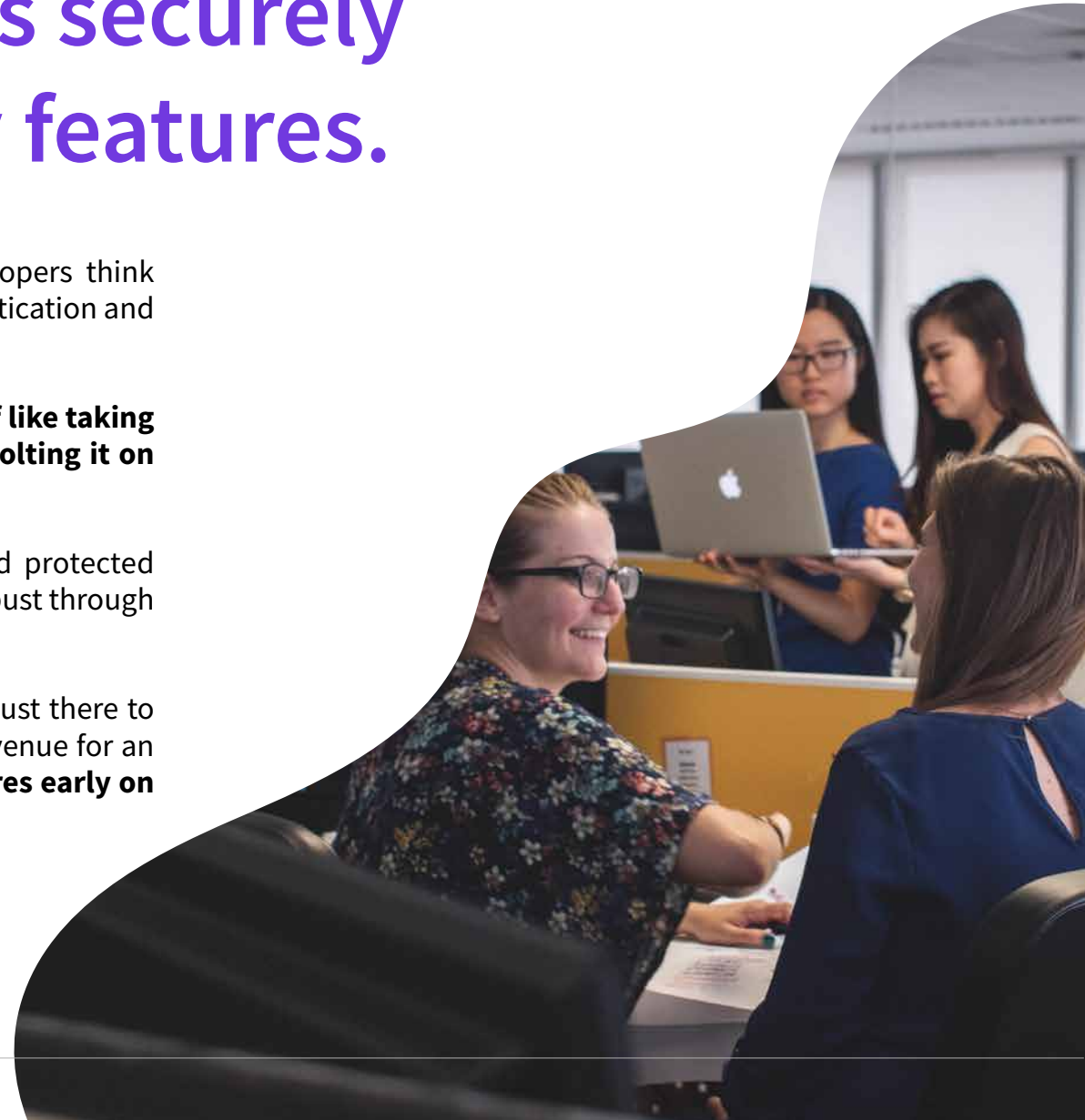
Principle #3: Implement features securely more than security features.

It's more than just adding security features. When developers think about security, they usually think about encryption, authentication and authorization.

If you just focus on those security features, that's sort of like taking a bank vault door with a really hard-to-pick lock, and bolting it on any old room in any old building.

The bad guys are going to ignore the highly evolved and protected security feature of the bank vault, and they're going to just bust through the walls of the building.

The software equivalent is that our products' features are just there to provide the functionality, and each feature is a potential avenue for an attacker to exploit. **We need to implement security features early on to cover any holes that a bolt-on feature misses.**



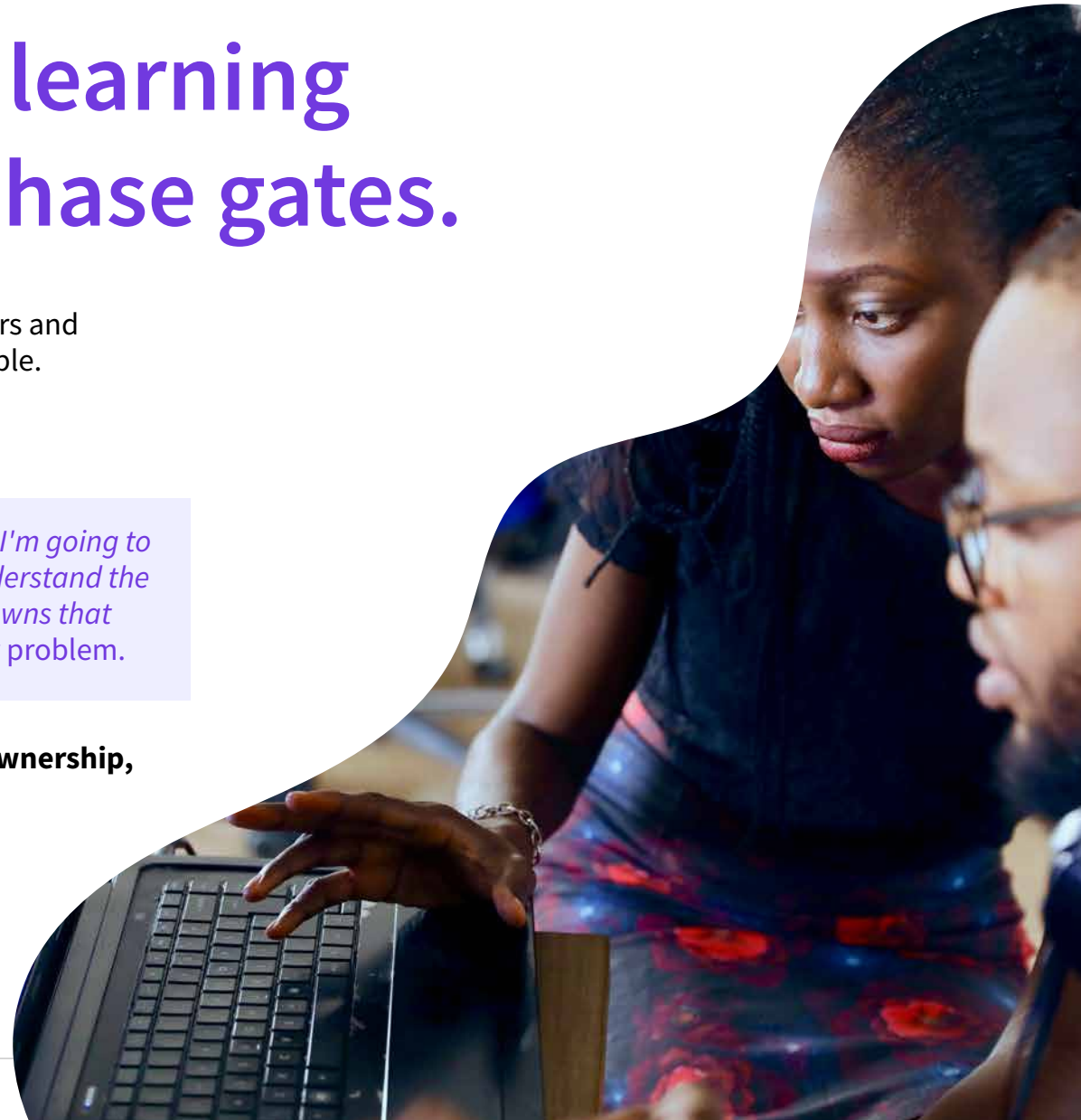
Principle #4: Rely on continuous learning more than end-of-phase gates.

Simply relying on end-of-phase gates disempowers developers and disincentivizes them to create the most secure product possible.

“That’s not my job, why should I care?”

“Security is going to check it, they’ll enforce their own policies, I’m going to just do the best I can, but I’m not going to try really hard to understand the security implications because we’ve got this other group that owns that aspect.” - Developers stop thinking of security as part of their problem.

If you shift the burden of security, you can never get full ownership, and ownership is the key to making this all work.



Principle #5: Build on culture change more than policy enforcement.

When you only focus on policy enforcement, you get a very negative pattern.

Policies outline a goal, but not always a common outcome. Organizations exposed to many security threats often find out that their **policies are outdated** and they **don't match the current schools of thought regarding security**.

A policy-oriented culture encourages developers to ignore potential security threats outside the scope of the policy and says developers don't have to worry about any security threat as long as they adhere to company policies.

Your company culture must be very adaptable to change because security best practices are constantly changing.

We need to let developers genuinely believe we trust them. They're integral to the business context and application of what they're building- so involve them in the process.

The Pledge

This pledge is the result of getting feedback and buy-in from dozens of small teams at Comcast. It's designed to **align the entire hierarchy of development with the overarching organization**. This has become our philosophy and marching order for our development teams.

The Pledge communicates one simple but powerful message to developers: **We trust you and we trust you want to do the right thing.**

We, the Security Team

Recognize that Engineering Teams:

1. Want to do the right thing
2. Are closer to the business context and will make smart trade-off decisions between security and other risks
3. Want information and assistance so they can improve our security posture.

Pledge to:

1. Lower cost and effort side of any investment in developer security tools or practices
2. Assist twice as much with preventative initiatives as we beg for your assistance in reacting to security incidents.

We **understand** that we are **no longer gatekeepers**, but rather tool-smiths and advisors.



Why the Pledge Works

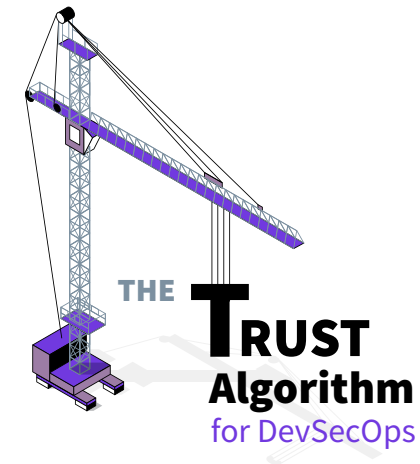
The Pledge is an incredibly powerful way to communicate with development teams.

The only time most development teams have contact with security people is when some vulnerability has been exposed, and **this changes the script**.


Development teams recognize that this is completely different from any other interaction they've had with a security team. We've seen firsthand how many developers are eager to work together and get behind this message.

The Trust Algorithm

At the heart of the trust algorithm is the trust formula, which is essentially credibility + reliability + empathy all over apparent self-interest.



$$\text{Trust} = \frac{\text{Credibility + Reliability + Empathy}}{\text{Apparent self-interest}}$$



Back to the Three-Part Framework:

Summary of Step #1

Winning the hearts and minds of developers needs organizational buy-in at both ends of the hierarchy. By adhering to the DevOps Manifesto, we can empower our developers.

In summary:

1. Build security in more than bolt it on.
2. Rely on empowered engineering teams, more than security specialists.
3. Implement features securely more than security features.
4. Rely on continuous learning more than end-of-phase gates.
5. Build on culture change more than policy enforcement.

This is all done to build trust with your development team, and increasing the numerator of credibility, reliability, and empathy, to establish more trust in the face of apparent self-interest.

**Make it easier
for development
teams to know
what the right
thing is (and
make it easy for
them to do it)**

You need to be able to sit down with a dev team and ask if they're getting the training they need, and make sure that training program is available to them.

You need to make it as easy as possible for your development team to accomplish their goals, by essentially helping them every step of the way.

The hard work is getting them to change their working agreements: What's the working agreement inside of my team for when and when we'll consider a story, feature, or sprint done?

We accomplish this through a DevSecOps self-assessment.

The DevSecOps Self-Assessment

The DevSecOps Self-Assessment is a facilitated self-assessment where you sit down with your team with a table of 30 different practices organized into a list of disciplines.

This table must be tailored to each individual organization, as you must decide what the working agreements are within your team.

Disciplines	< Less mature practices		More mature practices >	
Craftmanship	100% of group members who have been employed at least 90 days have been through Yellow Belt training	Group has access to at least one green Belt	Group has: -Minimum 15% green belt -At least one brown belt	Group has: -Minimum 25% Green Belt -Minimum 5% Brown belt -At least 1 Black Belt
Architecture and Design	As appropriate, a threat model or an security architecture review (SAR) has been done and kept up to date with every significant change to the attack surface and resolved all issues within SLA (typically 120 days for critical and high)			
Asset management	In iTRC all components of your applications have an accurate security POC. (To find: Application > Component > Responsibilities) (To add: Applications > component > New responsibility > security POC (select responsibility from drop down) and add New			
DevSecOps Tools Primary code analysis (PCA = SAST/IAST) Software composition Analysis (SCA)	PCA/SCA tool(s) integrated (nonvisible text) pipeline and (nonvisible text) merge, or cadence “Stop the bleeding” – Policy/configuration is set to interrupt the pipeline or stop progress unless we get a clean scan confirming no new (not legacy) vulnerabilities with all appropriate checkers turned on	Team has working agreements on how it becomes aware of, triages, and resolves findings in the current dev cycle to (nonvisible text) within current team chosen policy (see below). Ex: Notices in slack/email; and/or findings put in Jira backlog via (nonvisible text) considered at planning; and/or some highly visible console is checked on a regular cadence; and/or (nonvisible text) (“failed build” or disabled merge button), etc. We (nonvisible text) Application’s legacy code or a small # of high risk checkers (OWASP to 10 or appropriate subset)	We have clean scans for our application’s legacy code for a large # of (nonvisible text) high risk checkers (nonvisible text) PCI 3.2, etc. as appropriate)	We have clean scans of all code with all appropriate checkers on including 3rd party code and open source (that isn’t covered by an SCA)
Network-originated Scans	Findings for network-originated scans (ex: Qualys, Nessus) run against your components by the security team are resolved within the SLA (typically 120 days for critical and high)		Authenticated automated self-scans (Uscan, etc.) are run and findings are resolved in the current dev cycle	(PCI only) Authenticated scans are being run for your components and findings resolved within SLA
Independent Security Assessment	Independent security assessment (aka Pen testing) done & kept up to date at appropriate cadence (typically annually) and issues resolved within SLA		Red/Purple team exercises done on the system your resolved in the SLA.	

The DevSecOps Self-Assessment is so powerful in its ability to break an extremely complex and comprehensive organization down into a simple table.

The table is designed for the most gradual on-ramp for development teams. It should have all of the workings necessary to actually adopt a full DevOps shift into your team's core practices and keep you and your team accountable.

The table is color-coded to correspond to whether things are already done (green), being worked on (light green), not yet being worked on (red), or not launched yet (black).

Color code every cell, and at the end, ask the development team to pick a cell they want to turn green next quarter– enabling this and making accomplishing these goals as easy as possible is your next mission.

Get management involved

Summary of Step #3

In this step, management gets transparency of rollout status and the mechanism to set goals moving forward.

This allows management to prioritize building security features into every step of the programming, rather than steamrolling through feature after feature.

In parallel to the DevSecOps self-assessment, management should have their tracking criteria to view their organization's developmental practices' maturity as they're adopted.

Visualizing an Org's DevSecOps maturity:

Disciplines	< Less mature practices		More mature practices >	
Craftmanship	100% Yellow Belt after first 90 days of employment	At least 1 Green Belt	Min 30% Green Belt At least one Brown Belt	Min 50% Green Belt Min 20% Brown At least 1 Black Belt
Threat Modeling	A threat model has been done and kept up to date with every significant change to attack surface			
Static Analysis	New code break the build clean scan, all checkers "stop the bleeding"	Legacy code clean scan, small # of checkers (OWASP Top 10)	Legacy code clean, more checkers (SANS Top 25)	All code including 3 rd party clean scan, all checkers
Dynamic Analysis	Port Scans	Fuzzing (black-box dynamic)	White box dynamic	Automated remediation and/or responde (RASP)
Pen Testing		In house pen testing		Red team exercises
Reference Components and Designs	Mostly builtwith reference components and designs	Non-reference components have had an security code review		Contributing to reference components and designs

In Summary:

Encouraging engineering teams to adopt DevSecOps at a foundational level is about instituting meaningful behavioral and cultural change. **It's going to require buy-in from every level of your organization.**

By following our three-part **framework, you'll be able to encourage a cultural shift in your organization that fortifies your development practices against the growingly complex security threats of the modern era.** By understanding the base principles, honoring the pledge, and by performing meaningful self-assessment, individuals, team, and the organization as a whole can create buy-in and affect the needed change.

Reviewing the Steps:

Step #1:

Win the Hearts and Minds of Developers. Follow the five principles of the **DevSecOps Manifesto**:

- Build security in more than bolt it on.
- Rely on empowered engineering teams, more than security specialists.
- Implement features securely more than security features.
- Rely on continuous learning more than end-of-phase gates.
- Build on culture change more than policy enforcement.

Step #2:

Make it easier for development teams to know what the right thing is (and make it easy for them to do it). DevOps cultural change takes time– use a DevSecOps Self-Assessment tailored to your organization to track progress.

Step #3:

Get management involved. Get your management teams deeply involved and intimately aware of how progress and maturity happens in your development organization.

Introduction to Opsera

Opsera is a platform designed to help software teams ship code faster with optimal security, flexibility, and efficiency. Our Continuous Orchestration platform automates any CI/CD toolchain, enables declarative pipelines, and provides unified insights across your entire software delivery process.

Continuous Orchestration

Toolchain Automation

Automate any CI/CD toolchain, with zero coding involved. You choose your tools, we take care of the rest. Put together the perfect CI/CD stack that fits your organization's goals with zero vendor lock-in. Eliminate manual scripts and stop building toolchain automation. Free your engineers to focus on your core business.

Declarative Pipelines

Build declarative Pipelines, quality and security piped in. Pipeline workflows follow a declarative model so you focus on what is required — not how it's accomplished — including: software builds, security scans, unit testing, and deployments.

Unified Insights

Unify analytics and logs across your CI/CD ecosystem. Comprehensive and personalized software delivery analytics across your CI/CD process in a unified view — including Lead Time, Change Failure Rate, Deployment Frequency, and Time to Restore. Contextualized logs for faster resolution and improved auditing and compliance.

SaaS & Hybrid

Use Opsera as SaaS, with a separate VPC for each account.
Or, deploy Opsera as a Hybrid model in your own cloud.

Identity

We support SSO, including Okta with SAML, and LDAP. Manage access to your enterprise users with RBAC.

Security

A dedicated VPC per account, log aggregator, and database. Data is fully encrypted and secrets are managed via vault.

How it works.

1

Pick your tools

Choose any tool for any step in your toolchain. You're the architect of the perfect CI/CD pipeline that best fits your goals. Opsera is nonopinionated on your toolchain, so you get to choose your own best-of-breed vendors or opensource tools.



Build



Security



Testing



Deploy



Monitoring

We provision and manage, automagically

Opsera automatically provisions your tooling infrastructure with no coding involved. We even automate the updates of your toolchain.

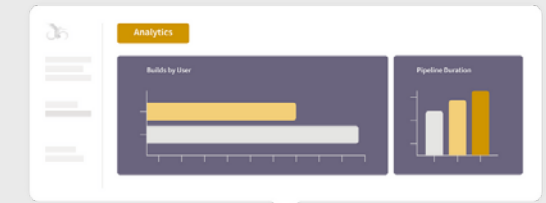
2

Build your pipelines, code free

Drag 'n drop the perfect workflow for each of your pipelines. You can build dependencies around parent/child pipelines to handle even the most complex deployments. Build pipelines for any language, Salesforce and AI/ML code deployments.



3

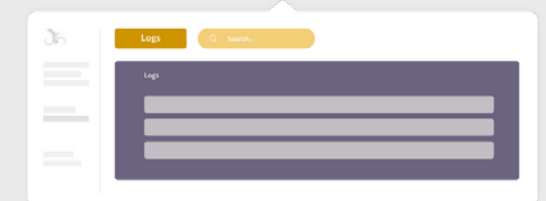


Metrics for all the things, and all the people

85+ KPIs across your DevOps ecosystem. Combining the analytics across all your tools, you get access to dashboards that give you the big picture across your pipelines, planning, security, quality, code and operations.

View aggregated logs, with context

Opsera aggregates the logs across all your pipelines and tools. Easily search by build number and swiftly diagnose failures. Generate files for auditory and compliance requirements.



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 for any app, and deploy anywhere.

Opsera accelerates best-practice CI/CD adoption so software teams can deliver software faster, safer, and smarter.



opsera.io