# SECURE DEVOPS
# SURVIVAL GUIDE

A Field Manual for Developers

VERACODE

# HOW TO SURVIVE IN A DEVOPS WORLD

Welcome to the Secure DevOps Survival Guide. Because you're reading this, you probably know your stuff when it comes to software development, but you might have some funny ideas about DevOps and security. With this manual as your guide, you'll learn how to get the job done. So strap on your boots. Stiffen your resolve. There's absolutely no whining allowed.

## YOU'RE A RECRUIT IN THE DEVOPS ARMY NOW.

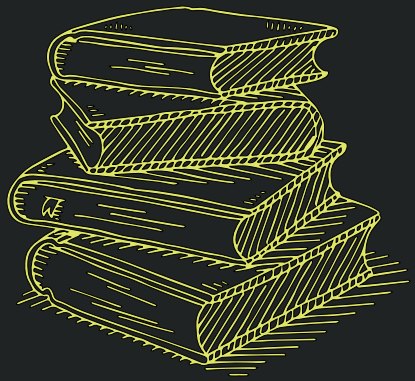

01 **BRIEFING**
Why DevOps?
PAGE 3

02 **PERSONNEL**
People and Culture
PAGE 8

03 **EQUIPMENT**
Tools and Technology
PAGE 12

04 **STRATEGY & TACTICS**
Elements of Secure DevOps
PAGE 16

# 01



# BRIEFING

## Why DevOps?

# WHY DEVOPS?

In the technology world, the only constant is change. Just as you've become accustomed to the demands of Agile development — the daily scrums, the short sprints and constant quality checks — the ground is quaking beneath your feet. There's a new philosophy that's catching on: DevOps. It's bringing development and operations together. And the responsibility for ensuring stability and security of software through production and customer usage is shifting left to include developers.

You might think everyone is using a DevOps model except you, but even if many organizations are thinking about making the switch, most haven't yet. According to Puppet, 22 percent of organizations have made the switch to DevOps. Many of them are large enterprises — according to Gartner, 25 percent of Global 2000 organizations have adopted DevOps.

If you've already started your journey to DevOps, or even if you're only starting to think about DevOps, you know that the old way of doing software development is inadequate for meeting the demands on modern development teams. Traditionally, the goals of development and IT operations are at cross-purposes: development wants rapid change and operations wants stability. Without cooperation between them, projects are rarely completed on time and frequently go over budget.

**22%**

of organizations have made the switch to DevOps, *according to Puppet*.

**25%**

of Global 2000 organizations have adopted DevOps, *according to Gartner*.

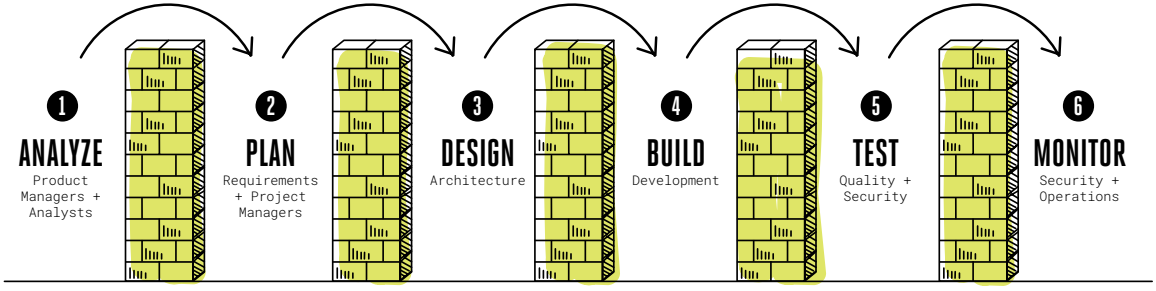
## IN MILITARY TERMS, THERE'S A LOT OF FRIENDLY FIRE — AND SECURITY CAN GO AWOL.

# HERE'S A QUICK BRIEFING ON THE LOGIC BEHIND THE SHIFT FROM WATERFALL TO AGILE AND DEVOPS.

## WATERFALL

**In the Waterfall approach, the development team would code a new application or feature and "throw it over the wall" to operations, the people responsible for running it.**

That creates a lot of stress and constant tension. Quality isn't addressed until the end of the development lifecycle, slowing down time to deployment. Processes are manual because automation doesn't make sense with the slow pace of development. If you're using Waterfall today, you're probably missing the target for budget and deadlines.



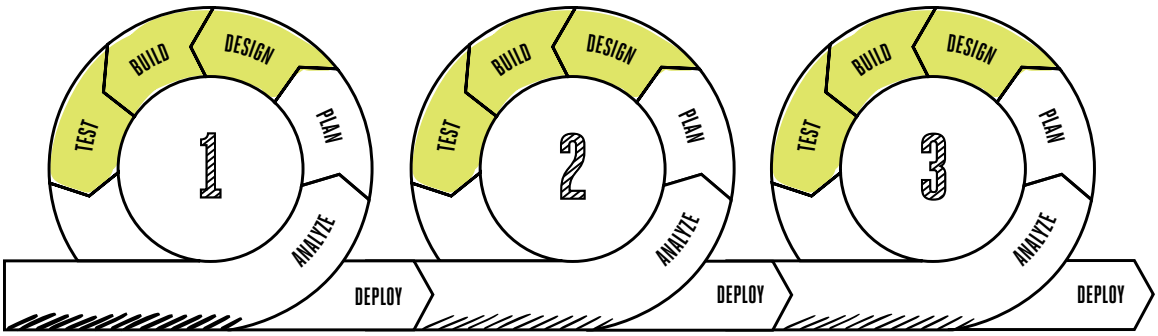

Traditional Waterfall Approach

# AGILE

**Agile development is a revolutionary step from the silos of Waterfall into self-organizing Scrum teams, with high autonomy and core values of transparency and accountability.**

If you've adopted Agile in your development team, you're seeing great improvements in quality and productivity. Development and QA work together as a single unit, under the direction of a Scrum master and product management.

Collaboration, a more manageable workflow and feedback loops improve efficiency on the development side of the wall. But silos remain: security stays separate, left out of the Agile process altogether. And operational requirements are still left until the end of the cycle. Many software projects improve quality, only to fall short in deployment due to unanticipated operational requirements for availability, scalability or manageability. Agile development teams are marching in the right direction, but not everyone is in sync.

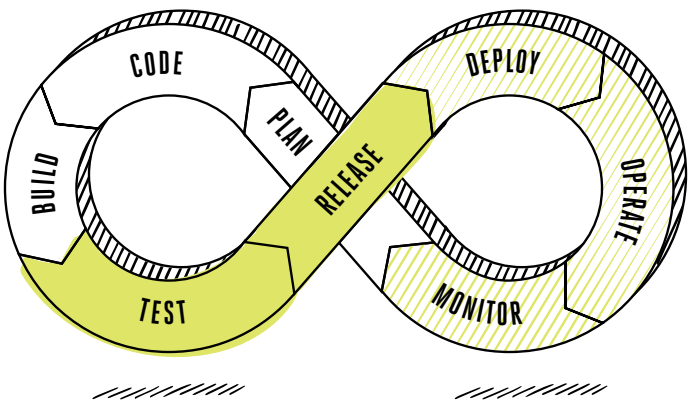



Agile Approach

# DEVOPS

**Fundamentally, DevOps is an extension of Agile's cross-functional teams to include operations. In this world, the developers have to understand how things will run in production, from much earlier in the lifecycle.**

Instead of development telling IT "it's your problem now," DevOps requires developers to assume the attitude of "you build it, you run it." The difference with DevOps is reduced organizational complexity and faster deployment. High-performing DevOps teams are twice as likely to exceed their goals for productivity and profitability, according to Puppet's 2016 State of DevOps Report. Yet there's one big problem: security is still on the outside looking in. Software is being deployed quickly, but with weaknesses that can be infiltrated by the enemy.




DevOps Approach

# WHY PURSUE DEVOPS?

1. Creates a culture of collaboration between development, operations and security.
2. Operational inputs and requirements "shift left."
3. Reduces lag time in delivering solutions.
4. Simplifies organizational complexity.

# 02

# PERSONNEL

People + Culture

# PEOPLE AND CULTURE

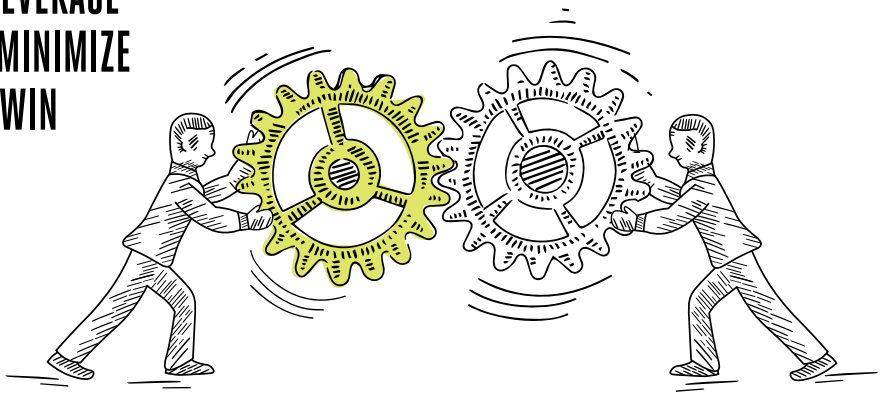
The basis of DevOps is a culture of collaboration and shared responsibility between development and operations for all aspects of the application lifecycle. That means teams must eliminate the impulse to point fingers, and embrace a culture of monitoring and feedback to improve the quality of code. Do your best to bring everyone along, but remember: old habits die hard and deep culture change is a challenge for some. In DevOps boot camp, some soldiers drop out.

Putting people into silos creates a false dichotomy between operational and security requirements. You can think about development, operations and security as separate service branches of the military. Each has a different role to play (army, air force and navy operate in unique environments with separate missions).

## COMPONENTS OF DEVOPS CULTURE

1. Collaboration, not silos.
2. Autonomous teams with decision-making power and no fear of failure.
3. Focus on quality as part of development, including performance and security.

## BUT THEY MUST COORDINATE THEIR EFFORTS, LEVERAGE STRENGTHS AND MINIMIZE WEAKNESSES TO WIN THE BATTLE.



## DEVELOPMENT

**MISSION**

Deliver features and functionality.

**TACTICS**

Develop, test and deploy quickly.

**WEAKNESS**

Wants to produce secure code but only if it's quick and easy.

## OPERATIONS

**MISSION**

Site reliability and uptime.

**TACTICS**

Visibility and automation.

**WEAKNESS**

Technical debt.

## SECURITY

**MISSION**

Protect applications throughout the lifecycle.

**TACTICS**

Thoroughness, not speed; Multiple layers of security.

**WEAKNESS**

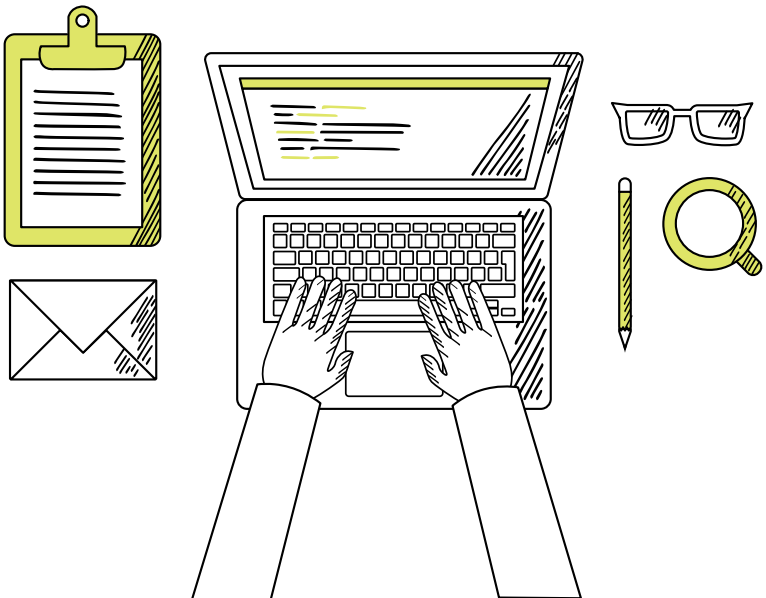
Mistrusts automated and agile processes.

# ROLE OF THE DEVELOPER

**It's easy to make stereotypes about people in different roles, their motives and personalities. One of these false stereotypes says that "developers don't care about security."**

Truthfully, developers tend to want to be efficient, and it drives you crazy to waste your time on manual processes. From your point of view, security is just one of the non-functional requirements, along with stability and performance.

DevOps managers, you need to give developers enough autonomy to unleash their motivation and ingenuity. Give the Scrum teams goals and oversight to make sure they meet set standards — but trust teams to achieve goals in their own way. Help teams learn from past mistakes by encouraging feedback and continuous improvement. Most importantly, give them the tools they need to integrate security into the pipeline.



## YOUR PART IN SECURING DEVOPS

### DEVELOPMENT

Invest in continuous learning; change how you think about code quality.

### OPERATIONS

Maintain visibility; automate for continuous deployment.

### SECURITY

Shift security left; cover the entire lifecycle.

# 03

# EQUIPMENT

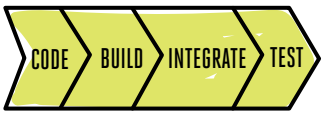
## Tools + Technology

# TOOLS AND TECHNOLOGY

What good is a well-trained army if it doesn't have the right equipment? Continuous integration and continuous delivery, or CI/CD, is the technological enabler of the DevOps Army. CI/CD speeds the delivery of secure code through automation of routine tasks like testing, configuration and deployment, so that people can focus on other valuable activities and avoid human error.

Generally mediated by a build server such as Jenkins (originally Hudson) or Azure DevOps, continuous integration (CI) seeks to automate all processes from software commit (or check-in) to "ready to test." The build server takes the changed software files, builds the software and verifies its correctness using an automated unit test, and creates a testable candidate for release. Unit tests provide immediate feedback on the correctness of a code change, allowing developers to move faster and with confidence that new changes won't cause issues elsewhere in the system.

## THE PROCESS OF CONTINUOUS DEPLOYMENT (CD) ALLOWS SOFTWARE RELEASES TO GO FROM EXPENSIVE, RISKY, MANUALLY INTENSIVE EVENTS, TO ALMOST CONSTANT OCCURRENCES.



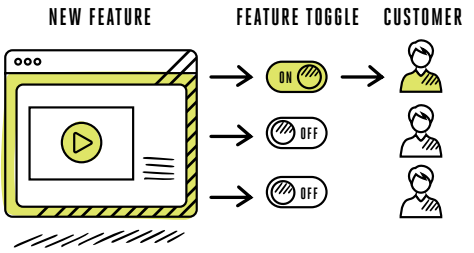

### CONTINUOUS INTEGRATION (CI)

Seeks to automate all processes from software commit (or check-in) to "ready to test."


### CONTINUOUS DEPLOYMENT (CD)

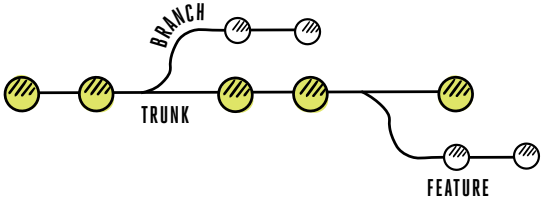
Allows software releases to go from expensive, risky, manually intensive events, to almost constant occurrences.

# CONTINUOUS DEPLOYMENT IS ENABLED BY



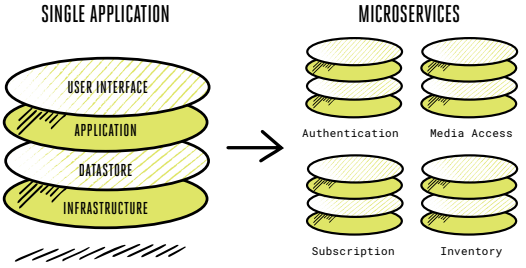

## FEATURE TOGGLES

Feature toggles or feature switches that allow development teams to ship new code as soon as it's ready, validating new features in the same running codebase for a small number of users before making them generally available.




## TRUNK BASED DEVELOPMENT

Rather than batch features on a branch and deploying a large code "monolith," organizations can develop on trunk and ship code from a developer's check-in directly into production.

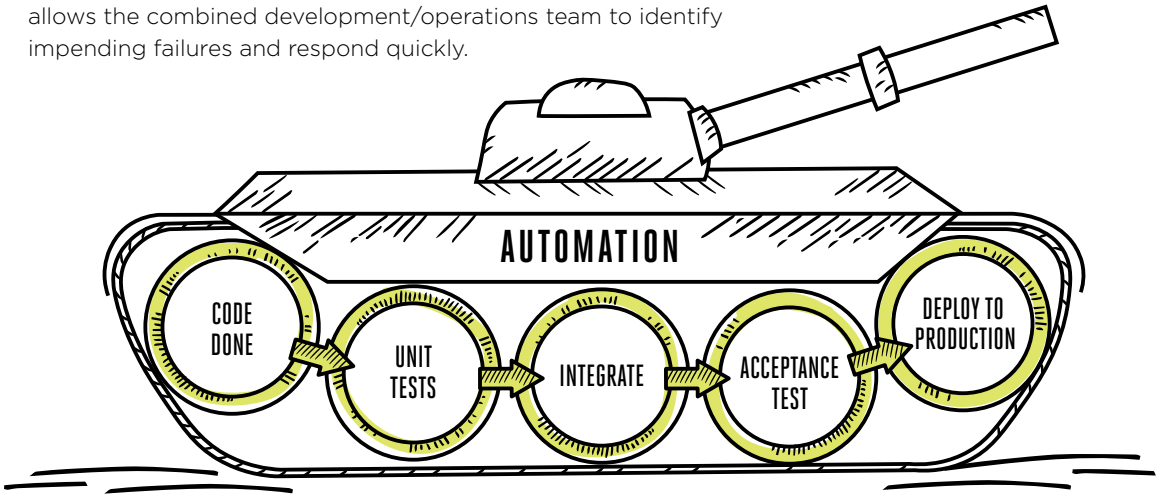



## MICROSERVICES

Applications architected as stand-alone microservices that deliver pieces of functionality — e.g., authentication, subscription maintenance, media access — allow developers to ship updates with increased certainty that no unknown code entanglements will cause the update to fail.

# DEPLOYING SOFTWARE WITH CI/CD IS SOMEWHAT LIKE DEPLOYING AN ARMY.

Orchestrating swarms of microservices being frequently updated and constantly activating new features demands full operational visibility into the state of the deployed application. Monitoring allows the combined development/operations team to identify impending failures and respond quickly.




Continuous Deployment

## DO'S AND DON'TS OF CI/CD

### DON'T BRANCH

Use feature switches.

### DON'T SHIP A MONOLITH

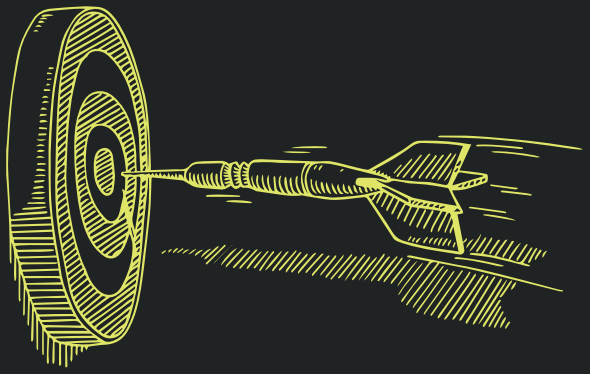
Microservices are cheaper to test and deploy rapidly.

### DON'T PATCH

Redeploy!

### DON'T HAVE ENVIRONMENT DIFFERENCES

Between development, stage and production. Automate creating new environments with code, or put all your environments in a lightweight container.



# 04

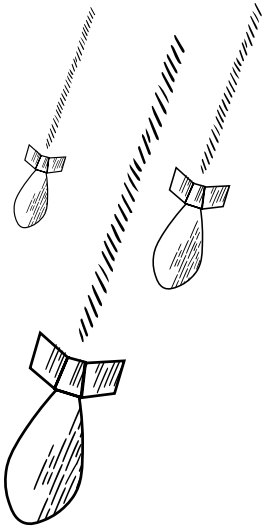
# STRATEGY + TACTICS

## Elements of Secure DevOps

# ELEMENTS OF SECURE DEVOPS

Rome wasn't built in a day. Securing DevOps is not a quick-and-easy process, either. How do you bring together the technological and cultural changes necessary to secure your code at the speed of DevOps? And how do you align the goals of development and operations with security? Your strategy is to bring development, operations and security together under a culture of quality improvement, with the right technology to build security into the pipeline.

## TACTICS FOR SECURING DEVOPS



1

### FAIL QUICKLY, THROUGH AUTOMATION

Build testing directly into the DevOps pipeline through automation. Fail tests as early in the DevOps pipeline as possible.

2

### INTEGRATE APPLICATION SECURITY INTO THE DEVELOPMENT TOOLS YOU ALREADY USE

Integrate security to remove friction. Security assessments should integrate with your IDE, build and ticketing systems to automatically test code and coordinate remediation.

3

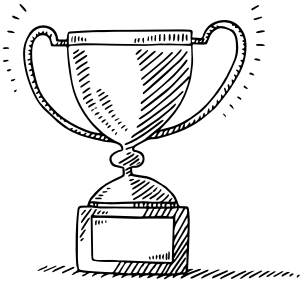
### FIX FLAWS AS YOU WRITE CODE

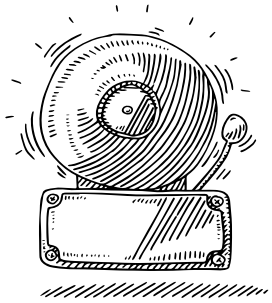
Give developers tools to find and fix coding errors while they write code, with immediate feedback before check-in, including developer sandboxes, "as-you-type" static testing and eLearning.

4

## BUILD SECURITY CHAMPIONS

Designate developers with an interest in security as security champions. These champions help to reduce culture conflict between development and security by amplifying the security message on a peer-to-peer level.





6

## DON'T STOP FOR FALSE ALARMS

Don't tolerate application security solutions that have a high false positive rate. False alarms may stop a critical business function from being deployed.

5

## ADAPT TO NEW DEVELOPMENT PRACTICES AND TECHNOLOGIES

DevOps requires adapting to new development practices and technologies.

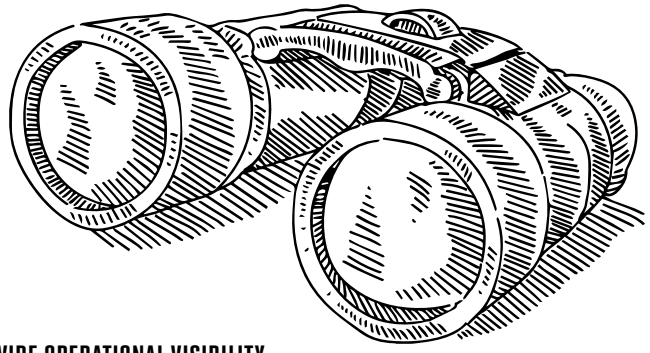
Including:

- Containerization
- Microservices
- Continuous Integration
- Infrastructure-As-Code (e.g., Boto, Lambda)
- New Languages (e.g., Scala, Go)
- New Definitions of Production (e.g., feature switches)
- New KPIs (e.g., downtime budget)

7

## EXTEND APPLICATION SECURITY INTO PRODUCTION

Application security cannot stop after deployment. As with other aspects of DevOps, a well-engineered solution must support closed-loop feedback from production in the event of a security incident.



8

## PROVIDE OPERATIONAL VISIBILITY

DevOps encourages team autonomy, but operations and security need visibility to measure and assess teams for compliance and risk.