DATADOG

WHITE PAPER

DevSecOps Maturity Model

A blueprint for assessing and advancing your organization's DevSecOps practices.

datadog.com

Table of contents

Executive Summary

Organizations must advance their DevSecOps practices to deliver high quality, secure digital services to market quickly and efficiently. In order to do that, leaders must ask themselves three key questions:

- What is our current level of DevSecOps maturity?
- Where is our desired level of DevSecOps maturity?
- How do we get there?

This white paper introduces a DevSecOps maturity model that technical leaders can use to answer these three questions, and enable their organizations to stay competitive in the digital economy.

We close with a discussion of the metrics leaders can use to demonstrate the business value of their DevSecOps initiative.

1 Three key DevSecOps questions for leaders to answer

DevSecOps is a necessary requirement for organizations to deliver at the speed and quality necessary to compete and innovate in the digital economy. However, while leaders acknowledge that DevSecOps is a strategic imperative, organizations struggle to get started on the journey and advance their practices. In order to move forward, technical leaders must ask themselves three key questions:

1. Where is my organization now?
2. Where do I want my organization to be?
3. How do we get there?

The first question requires an honest assessment of the organization's current DevSecOps competencies. The second question asks leaders to define what "good" looks like for their business given their competitive landscape. Finally, leaders need to identify initiatives that will bridge the gap between where they are now and where they want to be.

To answer the three key questions, technical leaders need a maturity model

According to software development expert Martin Fowler:

> "A maturity model is a tool that helps people assess the current effectiveness of a person or group and supports figuring out what capabilities they need to acquire next in order to improve their performance."[1]

A maturity model presents a prescriptive point of view on a particular domain and the most efficient and effective method to advance within that domain, as shown below:

[1] https://martinfowler.com/bliki/MaturityModel.html

Methodology

Our technical enablement teams work hand in hand with customers to help drive their DevSecOps transformations. In addition, we have more than 10 years of experience helping over 14,000 companies drive DevOps (and now DevSecOps) practices. As a result, we've observed companies at all levels of DevSecOps maturity and seen their paths of progression. We've built a DevSecOps maturity model that distills these customer experiences into efficient paths that any organization can replicate.

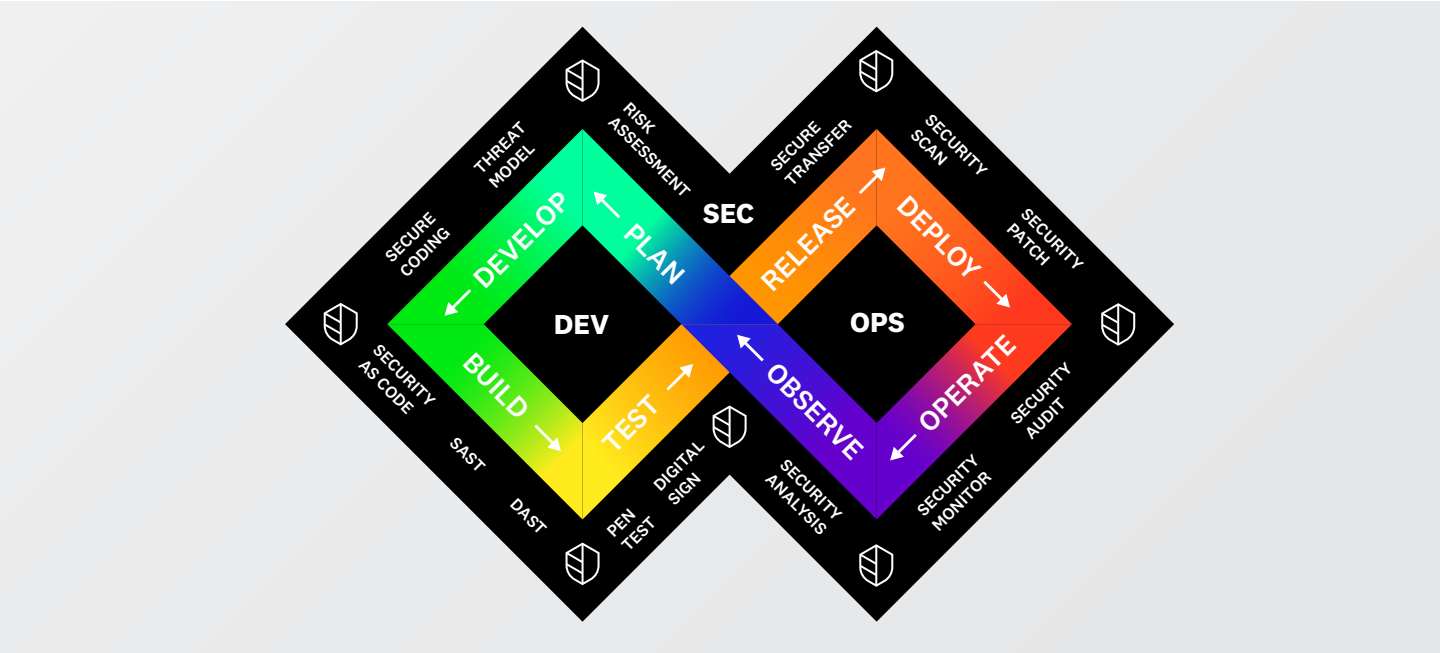
DevOps vs. DevSecOps

The DevOps movement emerged more than 10 years ago to improve the speed and quality of writing and running software by encouraging greater collaboration and shared responsibility between Dev and Ops teams. Organizations are still progressing in their DevOps journeys with varying degrees of speed and success.

The increasing velocity of DevOps teams has opened the door for two complications: (1) security issues are overlooked because DevOps teams are mainly concerned with functional and performance characteristics of software, not security, and (2) security is a bottleneck (or ignored) because security teams still exist in a separate silo with separate tools, culture, and processes from their DevOps counterparts (who are also moving with increasing speed).

Importantly, these complications, which slow down the DevOps lifecycle, are also occurring at a time when security itself is of increasing importance. Organizations are under continuous attack from a wide variety of threat actors. As more business is conducted through digital channels and as organizations' attack surface increases, technical and business risks correspondingly grow.

DEVSECOPS LAYERS IN SECURITY CONTROLS, TOOLS, AND PRACTICES THROUGHOUT THE DEVOPS LIFECYCLE.



All these developments suggest that Security must be more deeply integrated into the DevOps lifecycle. DevSecOps is therefore the logical next stage of evolution in the DevOps movement. By integrating security teams and practices into DevOps workflows, firms can further accelerate their speed of delivery, increase the quality of their software, and boost the reliability of their services in production. Breaking silos between security teams and DevOps teams is essential for realizing the full potential of the DevOps movement. DevSecOps is not a departure from DevOps, but is simply the next evolution of DevOps.

2 The DevSecOps Maturity Model

The DevSecOps Maturity Model identifies four stages of maturity across six major competency areas. Below, we give an overview of each stage and competency area before presenting the full maturity model.

The Stages

We identify four key stages of DevSecOps maturity. These stages are based on patterns witnessed in thousands of diverse organizations. Importantly, there are no shortcuts to advancing, and no ways to “leapfrog” a level. DevSecOps as represented by the maturity model is a journey.

- **Beginner:** This phase marks the beginning of the DevSecOps journey. Most important is a shift in culture and mindset that emphasizes sharing and collaboration across technical disciplines, and a desire to improve performance as a team. This is the foundation of DevSecOps.

- **Intermediate:** In this stage, organizations are consistently releasing software but may experience bottlenecks, performance issues, and some team friction. While security controls are shifting earlier in the development process, much of the security-related work is still done towards the end of the process, which can slow down release cycles and result in lower quality code.

- **Advanced:** In this stage, organizations are highly efficient and productive, releasing high quality, secure software on a regular basis to a reliable platform. Security checkpoints are embedded throughout the software development lifecycle.

- **Expert:** These are DevSecOps practices employed by the most cutting edge organizations. These organizations release high quality code multiple times per day. Security controls are deeply embedded throughout the SDLC, and security has ceased to be a siloed domain. A key aspect of this stage of maturity is a very high level of automation of processes across Development, Operations, and Security.

The Competencies

The DevSecOps Maturity Model covers six key competencies:

- **People & Culture:** This competency is the foundation of DevSecOps. This area encompasses organizational structure, communication styles, values, incentives, behaviors, leadership, and individual and team health.

The remaining five competencies can be mapped to the major phases of the end-to-end DevSecOps lifecycle. These competency areas blend process and technology.

- **Plan & Develop:** This competency area encompasses how work is prioritized, how much work is planned versus unplanned, how much work is new feature development versus paying down technical debt, and how much risk assessment and code validation factors into the earliest stage of the development process.

- **Build & Test:** This area covers testing processes and automation, quality assurance, code scanning techniques, and build and signature validation.
- **Release & Deploy:** This competency focuses on deployment strategies and release frequency, automation of the deployment process, and validation and remediation of deployment issues.
- **Operate:** This area covers infrastructure as code, capacity planning, scaling and reliability, chaos testing and red teaming, patching, and disaster recovery.
- **Observe & Respond:** This competency focuses on Service Level Objectives (SLOs), vulnerability and misconfiguration scanning, security monitoring, user experience monitoring, incident management, and post-mortems.

The Model

In the matrix below, each of the six competency areas encompasses a series of separate competencies, at least two of which are a security-related competency. For each competency, we identify four levels of maturity: Beginner, Intermediate, Advanced, and Expert.

(Note: the Appendix to this document contains additional detail on each cell in the matrix below.)

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
People & Culture	– Functional teams siloed – High inter-team friction – Nascent onboarding processes – Burnout common	– Silos breaking down – Embracing experimentation & transparency – Onboarding process exists – Burnout openly discussed	– Continuous collaboration across teams – Blameless culture – Comprehensive onboarding process – Burnout quickly addressed	– Cross-functional teams aligned to products and services – High trust, experimentation, learning culture – Burnout rare
Plan & Develop	– Risk and security not considered – High technical debt – Excessive bug fix work – Code not validated	– Limited risk assessment – Moderate technical debt – Moderate bug fix work – Some code validation	– Threat modeling and risk assessments – Low technical debt – Low bug fix work – All code validated	– Extensive threat modeling/risk assessment – Minimal technical debt – New feature focus – All code validated automatically
Build & Test	– Manual testing – No code scanning – No build/signature validation – Limited core functionality testing	– Partial test automation – Partial code scanning – Partial build/ signature validation – Partial core functionality testing	– High test automation – Dynamic code scanning – Significant build/signature validation – Significant core functionality testing	– Complete test automation – Comprehensive dynamic code scanning – Comprehensive build/signature validation – Comprehensive core functionality testing
Release & Deploy	– Manual deployments – Large, infrequent releases – No deployment security posture criteria – Difficult to remediate failed deployment	– Partial deployment automation – Medium-sized, monthly releases – Basic deployment security posture criteria – Acceptable failed deployment remediation times	– High deployment automation – Small, weekly releases – Detailed deployment security posture criteria – Fast failed deployment remediation times	– Full deployment automation – Numerous daily releases – Automated deployment failing – Bias to fast forward fixes

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Operate	– Manual provisioning/ configuration – Long capacity planning cycles – Manual scaling – Single availability zone – No chaos testing or red teaming – Poor patching hygiene – No disaster recovery strategy	– Partial configuration/ provisioning automation – OpEx-based capacity planning – Partial auto-scaling – Multi-availability zone/region – Basic chaos testing or red teaming – Basic patching hygiene – Basic DR strategy	– Extensive configuration/ provisioning automation – Capacity planning based on seasonality/ growth – Significant auto-scaling – Multiple cloud providers / high availability – Significant chaos testing & red teaming – Fast patching – Comprehensive DR strategy	– All infrastructure configurations and instructions instantiated as code – Capacity planning based on granular usage trends/ predictions – Comprehensive auto-scaling – Multiple cloud providers / very high availability – Continuous chaos testing & red teaming – Patching SLA – DR plans tested often
Observe & Respond	– No SLOs formed – No vulnerability/ misconfiguration scanning – No security metrics defined – Siloed telemetry – User journeys unknown – Excessive MTTD and MTTR – No post-mortems	– Basic SLOs formed – Partial vulnerability/ misconfiguration scanning – Some security metrics defined & visible – Some common observability data sets – Basic understanding of user experience – Moderately high MTTD and MTTR – Basic post-mortems	– SLOs & error budgets favored – Significant vulnerability/ misconfiguration scanning – Security metrics defined & visible for most services – Common observability data platform – Detailed user journey visibility – Moderate-to-low MTTD and MTTR – Detailed post-mortems	– SLOs & error budgets drive decisions – Extensive vulnerability/ misconfiguration scanning – Security metrics defined & visible for 100% of services – Standardized metadata model – Complete user journey visibility – Very low MTTD and MTTR – Clear, blameless post-mortems

3 Implications for your DevSecOps Journey

Let's return to the three key questions for technical leaders (Where is my organization? Where do we want to be? How do we get there?), and discuss how the Maturity Model can help answer them.

Where is my organization now?

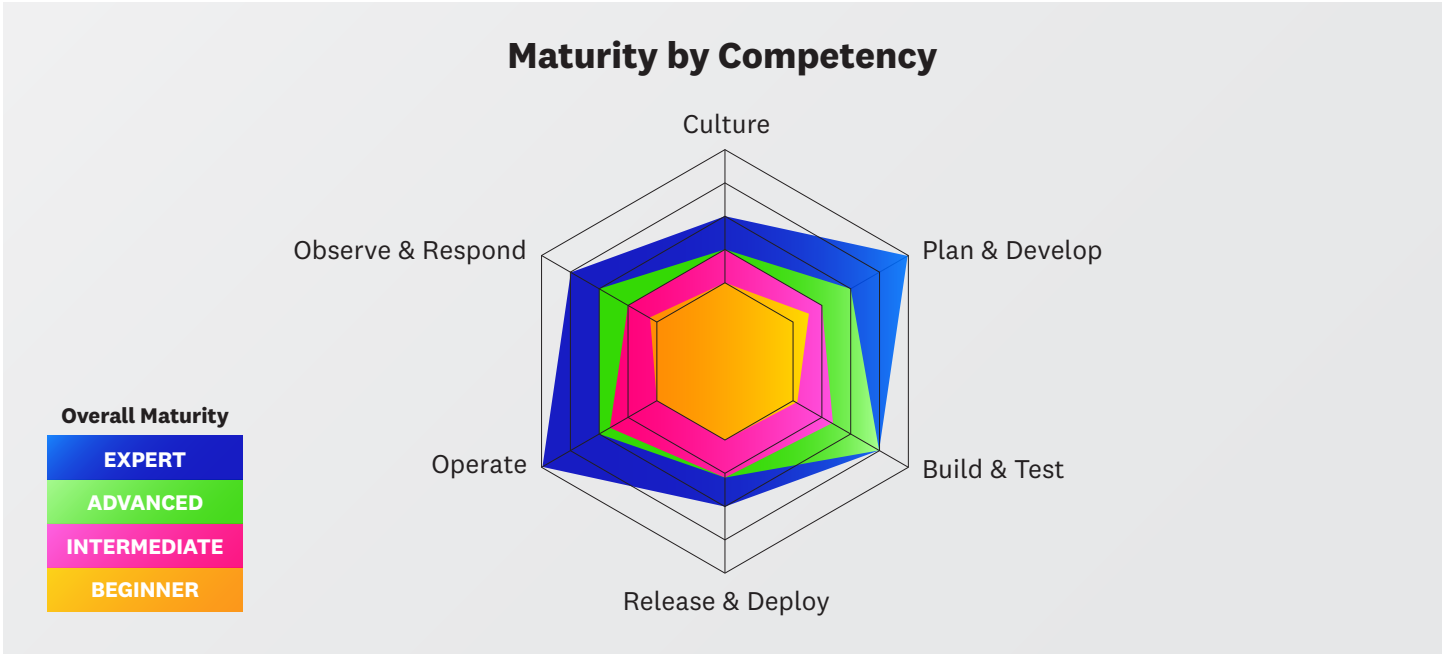
The DevSecOps Maturity Assessment

Technical leaders need to calibrate where their organizations are on the DevSecOps maturity curve. Towards that end, we've developed an online self-assessment tool based on the DevSecOps Maturity Model. The assessment is 36 questions and takes 10 minutes to complete.

The assessment is a diagnostic tool that is not meant to be precise, but to give a rough indication of an organization's DevSecOps maturity, and areas to consider for improvement. The assessment generates an overall maturity score.

Because organizations often have varying levels of maturity across competency areas, it's valuable to plot maturity levels on radar / spider charts, as shown below.

STEP 1: ASSESS WHERE YOUR ORGANIZATION IS



Based on the output of the assessment, leaders can see at a glance where there is room for improvement and investment.

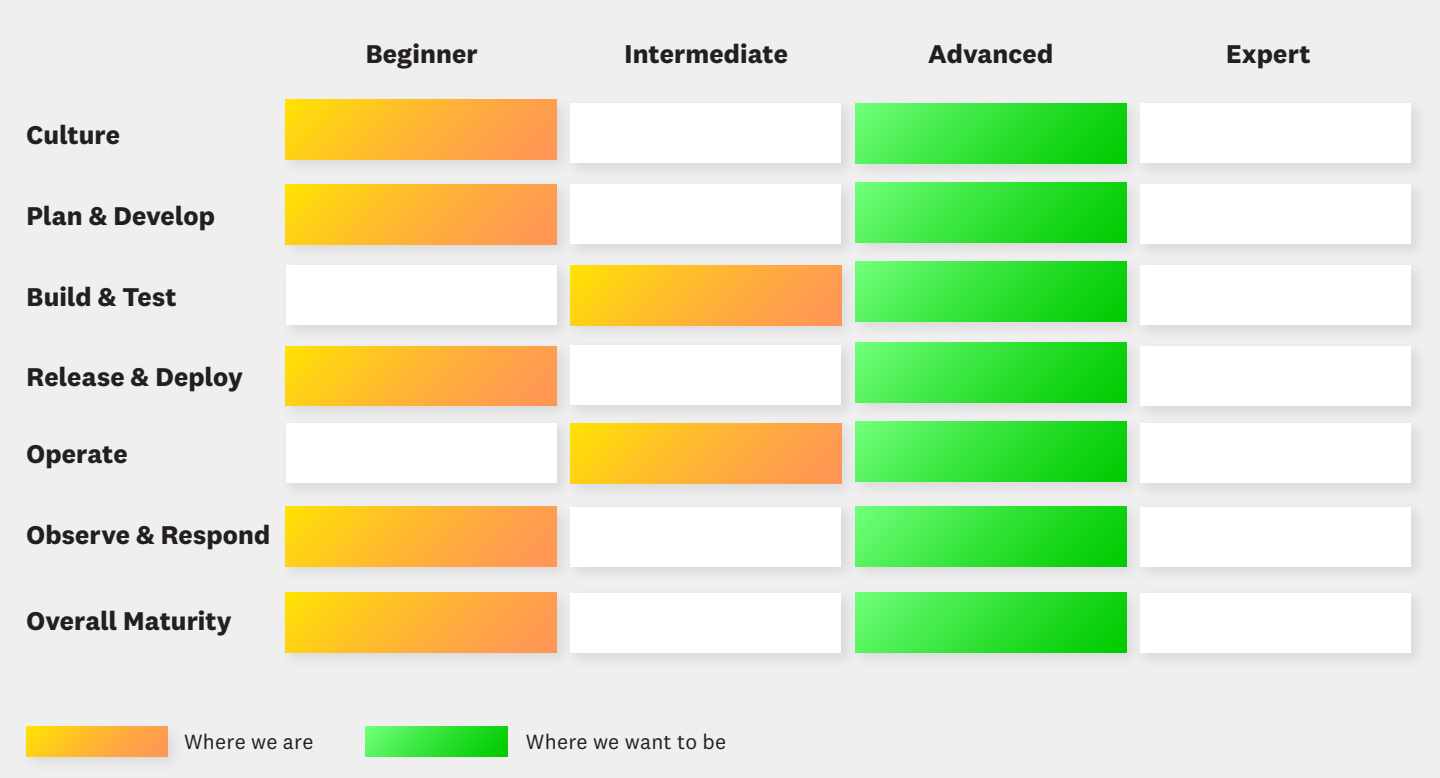
Where do I want my organization to be? Moving right in the matrix

Once leaders have a sense of where their organizations stand in their DevSecOps practices, the next step is to determine what "good" looks like given their industry and business goals. The stages at the far right of the maturity model show what best-in-class DevSecOps practices look like in today's enterprises.

It's important to note that DevSecOps maturity varies across industries. An "Intermediate" rating might be highly competitive in one industry but lagging in another industry.

A useful exercise is shown below. Here, we highlight a hypothetical organization's maturity across all competencies, and we highlight reasonable targets to hit within the next 12–18 months.

STEP 2: DEFINE WHERE YOUR ORGANIZATION NEEDS TO GO



Keep in mind that progress is incremental. Advancing even one level of overall maturity within 12 months is an accomplishment. Depending on the amount of work to be done, leaders may need to set multi-year plans with intermediate targets.

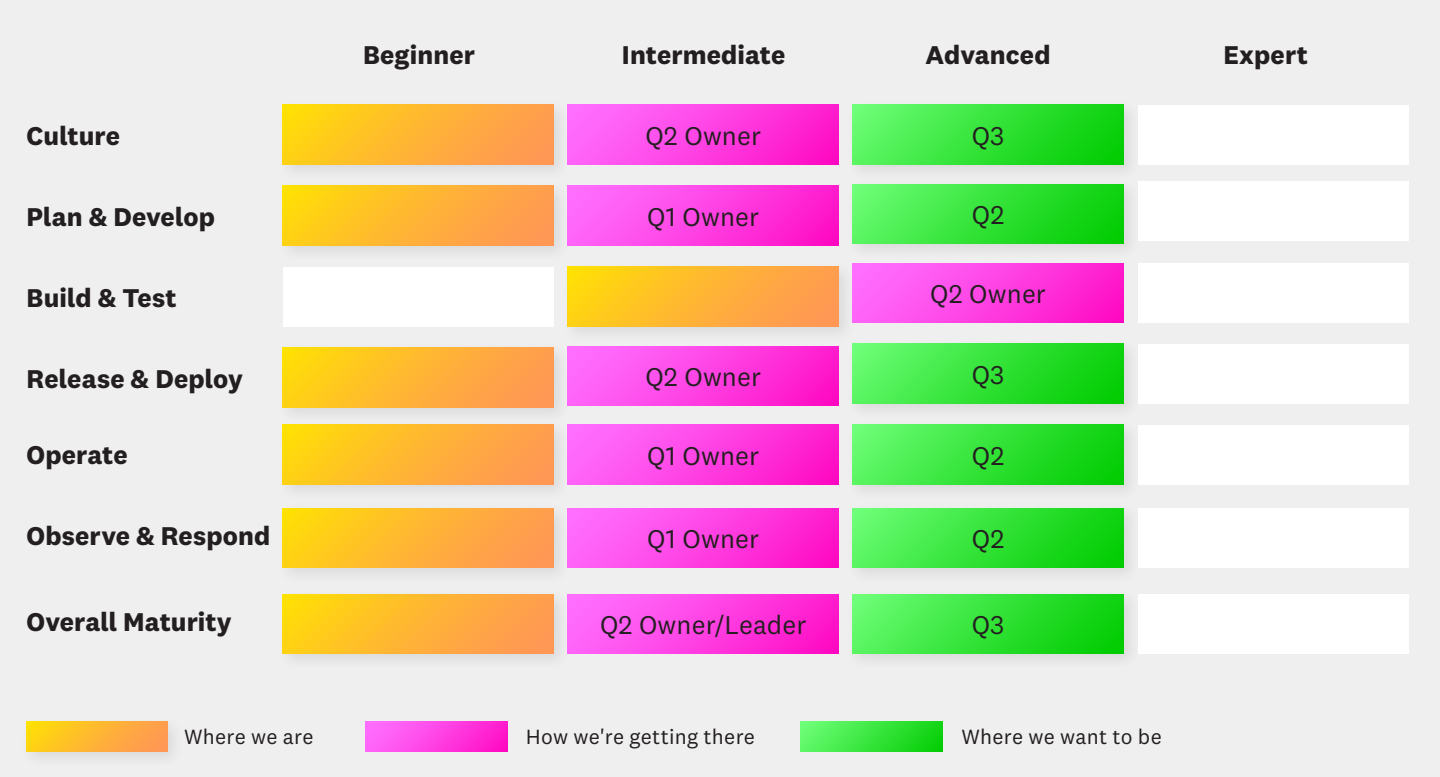
It's also important to remember that state of the art DevSecOps practice is constantly evolving and advancing. An "Advanced" rating one year might become an "Intermediate" rating the next. For this reason, it's important for both maturity models to stay up-to-date and for leaders to continually reassess their organizations using the latest models.

How do I get there? One cell at a time.

Because DevSecOps is a holistic set of practices spanning people, process, and technology, each competency reinforces the other competencies. For this reason, an organization with low maturity in one area will likely not be able to advance overall very quickly until the lowest maturity areas are addressed. We recommend prioritizing low maturity areas first in order to build a strong foundation for more advanced stages of maturity.

The cells in the maturity model show the incremental steps leaders can take to move from one level to the next.

STEP 3: DETERMINE HOW YOU'LL GET THERE BY PRIORITIZING INITIATIVES AND INVESTMENT AND NOMINATING OWNERS FOR EACH COMPETENCY AREA AND INDIVIDUAL COMPETENCY



Each of the competency categories and each of the specific competency areas is a large topic unto itself. We recommend leaders enlist direct reports to own specific competency areas, who can then enlist their team members to own specific competencies (e.g. Test Automation).

4 The Business Value of DevSecOps

DevSecOps drives more productive, collaborative, and responsive teams that deliver high quality, secure software faster to highly reliable production environments. This translates to tangible business value. Let's break down how.

Four primary value drivers

Organizations that adopt the DevSecOps practices in the maturity model realize business value through four key drivers:

1. **Faster, more agile delivery and reduced time to market:** DevSecOps enables organizations to deliver applications to market faster, and confidently iterate revenue-impacting applications with more frequency to protect and grow revenue. The integration of security into DevOps workflows eliminates potential bottlenecks and accelerates organizations' efficiency and agility.

2. **Improved security posture and reduced risk:** DevSecOps integrates security stakeholders and security practices into all phases of the software development lifecycle and the operation of services in production. Greater collaboration, trust, and transparency among Dev, Sec, Ops teams results in lower risk software.

3. **Reduced operational and development costs:** The fast feedback loops of DevSecOps practices streamline the software development life cycle and eliminate the vast mtajority of issues before they reach production environments. Incidents that do occur are resolved very quickly.

4. **Improved customer experiences and satisfaction:** By producing higher quality and more secure software, DevSecOps increases the value organizations provide to their customers. Customers also value more frequent enhancements and upgrades to their services. Finally, customer satisfaction is also boosted when organizations are able to observe systems from the end-users' perspective and have visibility into end-to-end customer journeys.

	METRIC	COSTS	CUSTOMER	REVENUE
Faster, more agile delivery and reduced time to market	↑ Release frequency ↓ Time to market ↑ Issues identified in Dev & QA environments	↓ QA time required to identify, recreate, and document defects ↓ Developer wait time	↑ Customers/ market share ↑ Customer satisfaction ↑ Customer share of wallet	↑ Revenue from new customers ↑ Revenue from increase in share of wallet ↑ Revenue from accelerated time to market ↑ Revenue from new products ↑ Revenue from pricing innovation
Improved security posture and reduced risk	↓ MTTD/MTTR ↓ FTEs involved per incident ↓ Customer complaint calls ↓ Incidents/ outages	↓ Engineer time to resolve incidents ↓ Tech support center costs ↓ Financial losses due to performance degradation or security incidents	↑ Customer satisfaction ↑ Customer share of wallet ↓ Customer churn	↓ Lost revenue due to outages ↑ Revenue from reduced churn ↑ Revenue from higher share of wallet
Reduced operational and development costs	↑ Release frequency ↑ Issues identified in Dev & QA environments ↓ Incidents/ outages	↓ Engineer time to resolve incidents ↓ QA time required to identify, recreate, and document defects ↓ Developer wait time ↓ Over-provisioning infrastructure		
Improved customer experiences and satisfaction	↑ Customer satisfaction	↓ Tech support center costs	↑ Customer satisfaction ↑ Customer share of wallet ↓ Customer churn	↑ Revenue from reduced churn ↑ Revenue from higher share of wallet

DevSecOps business value metrics

As organizations increase their DevSecOps maturity, the business value derived from each of these drivers increases. The table above covers in more detail the business value of each driver in terms of productivity metrics, customer metrics, costs, and revenue.

Technical leaders can and should measure their DevSecOps journeys using the above metrics. These metrics are essential for demonstrating progress throughout the organization, and for justifying the investments necessary to progress along the maturity curve.

5 Get Started

The DevSecOps Maturity Model is based on patterns we've seen working with thousands of customers to advance their DevSecOps practices. It has been validated with customers and used as a tool to help leaders answer the three key questions: Where are we? Where do we want to go? How do we get there?

We recommend technical leaders complete the 10-minute DevSecOps Self-Assessment to take the first step in your DevSecOps journey.

Authors

Jeremy Garcia, Director, Technical Community & Open Source

Andrew Krug, Technical Evangelist

Fahim Ghaffar, Vice President, Technical Services

Boyan Syarov, Principal Solutions Engineer

Christy Pasion, Director, Technical Enablement

Ziquan Miao, Senior Technical Account Manager

About Datadog

Datadog is the monitoring and security platform for cloud applications. Our SaaS platform integrates and automates infrastructure monitoring, application performance monitoring and log management to provide unified, real-time observability of our customers' entire technology stack. Datadog is used by organizations of all sizes and across a wide range of industries to enable digital transformation and cloud migration, drive collaboration among development, operations, security and business teams, accelerate time to market for applications, reduce time to problem resolution, secure applications and infrastructure, understand user behavior and track key business metrics.

For more information, visit datadoghq.com

Appendix: Detailed Maturity Model

1. Culture

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Communication	Siloed by functional team.	Limited to Dev and Ops teams. Security remains siloed, and team members don't know who to report security concerns to.	Security stakeholders regularly share with Dev and Ops teams but not as frequently as Dev and Ops teams share.	Regular communication and sharing across operations, development, and security. Team members know who to report security concerns to.
Onboarding	No standardized onboarding process.	Onboarding process exists, but engineers are not fully productive after completing and ramp up time is long.	Engineers are considered productive after onboarding.	Comprehensive onboarding process enables engineers to be fully productive and ramp up quickly.
Accountability	Fear, lack of trust, blame, and fingerpointing.	Fear of experimentation, some transparency, behind the scenes fingerpointing.	Blameless culture and frequent experimentation.	Transparent, blameless, high trust, learning culture, and experimentation.
Team health	Team members not able to discuss burnout and not empowered to take mitigation measures.	Team members openly discuss burnout, but are not empowered to take mitigation measures.	Team members are able to discuss burnout and are empowered to take mitigation measures.	Burnout is rare, but is openly discussed and quickly addressed.

2. Plan & Develop

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Risk assessment	Security and risk are not considered at the beginning of the development cycle.	Security and risk considerations are introduced in middle-to-late stages of the development cycle.	Risk assessment or threat modeling is conducted at the beginning of some but not all services at the design stage.	Risk assessment or threat modeling is used for every new service as part of the design phase.
Technical debt management	Technical debt increases uncontrolled.	Technical debt is semi-regularly reduced but reduction is not prioritized.	Technical debt management is emphasized.	Technical debt reduction across applications and infrastructure is consistently tackled and remains low.
Prioritization	Engineers spend the majority of their time performing unplanned/ bugfix work and remediating incidents.	Engineers are frequently interrupted by unplanned / bug fix work, which delays planned releases.	Engineers spend most of their time on new features, but unplanned work is still significant.	Engineers spend the majority of their time creating new customer-facing features and functionality.
Code validation	Code is not validated after development.	Code is validated partially and manually after development.	Static code analysis (e.g. Static application security testing, or SAST) is performed on some code to prevent commits of vulnerable code.	Static code analysis (e.g. Static application security testing, or SAST) is performed during the development phase to prevent commits of vulnerable code.

3. Build & Test

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Test automation	Manual testing is performed by dedicated teams.	Testing is partially automated with significant manual testing.	Testing is mostly automated.	Testing is fully automated and various testing regimes are applied at all stages of the development lifecycle.
Code scanning	Committed code is not scanned to stop the packaging of vulnerable code.	Some code is scanned to stop the packaging of vulnerable code.	Dynamic code scanning (e.g. Dynamic application security testing, or DAST) is performed on some committed code to stop the packaging of vulnerable code.	Dynamic code scanning (e.g. Dynamic application security testing, or DAST) is performed on all committed code to stop the packaging of vulnerable code.
Build validation	Builds and signatures are not validated to block unsigned or vulnerable packages.	Builds and signatures are partially validated to block unsigned or vulnerable packages.	Most builds and signatures are automatically validated to block unsigned or vulnerable packages.	Builds and signatures are automatically validated to block unsigned or vulnerable packages.
Quality assurance	Core business functionality is not tested.	Infrequent or manual testing of core business functionality.	The core business functionality of many applications is frequently and automatically tested.	The core business functionality of all applications is continuously and automatically tested.

4. Release & Deploy

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Deployment automation	Teams manually move code from one environment to another.	Partial automation of deployment process.	Automation of most of the deployment process.	Tooling allows fully automated deployments into production.
Deployment strategy	Waterfall methodology results in large, infrequent releases.	New code is released semi-regularly (e.g. monthly).	Agile methodology and modern deployment strategies (e.g. canary, blue-green, shadow) support regular releases (e.g. weekly).	Agile methodology and modern deployment strategies (e.g. canary, blue-green, shadow) facilitate releases multiple times per day.
Deployment validation	There is no criteria for failing a new deployment based on security posture.	There is a limited set of criteria for failing a new deployment based on security posture, and deployment validation is inconsistent.	A set of criteria exists for failing a new deployment based on security posture, but implemention is not fully automated.	A set of criteria exists for failing a new deployment based on security posture, and it is automatically implemented.
Deployment remediation	Remediating a failed deployment is a time consuming and manual process.	Teams have the ability to quickly roll back a failed deployment.	Teams can quickly roll back a failed deployment but often make a forward fix instead.	Teams are biased to forward fixing deployment issues, and are capable of doing so quickly.

5. Operate

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Platform management	Configuration management sprawl and lack of deployment templating.	Infrastructure configurations partially committed to code repository and some manual processes exist.	Infrastructure configurations fully committed to code repository with mostly automatic deployments.	Infrastructure managed by configuration management/ orchestration tools and committed to code repository.
Capacity planning	Long capacity planning cycles (annual or quarterly) leveraging CapEx budget.	Capacity planning leverages OpEx budget, but limited insight into seasonality and growth.	Capacity planning leverages OpEx and informed by seasonality and growth.	Capacity planning leverages OpEx and based on seasonality and growth data.
Scaling	Manual scaling.	Pre-warmed environments with mix of automatic and manual scaling processes.	Partial auto-scaling of the environment.	Auto-scaling occurs when certain conditions are met (e.g. influx of legitimate requests).
Reliability	Production environments run on single cloud provider region or availability zone.	Production environments span multiple availability zones and/or regions.	Production environments span multiple AZs, regions, cloud providers.	Highly available production environments span multiple AZs, regions, cloud providers.
Resiliency testing	Environments not tested to the breaking point and no red team tests/ adversary simulation conducted.	Performance testing only in pre-production environments. Infrequent red team testing.	Frequent chaos testing on some production environments. Frequent red team testing.	Continuous chaos tests on production environment. Continuous red team tests.
Patching	Patching is infrequent and not systematic.	Regular patching but systems remain vulnerable for long periods of time.	Consistent patching after vulnerabilities detected but no established SLA.	Established SLA for patching systems found to be vulnerable.
Disaster recovery (DR)	No DR strategy in place.	DR strategy in place but not tested regularly and involves significant downtime.	DR strategy in place that is tested semi-regularly.	DR strategy in place that is tested at regular intervals.

6. Observe & Respond

COMPETENCY	BEGINNER	INTERMEDIATE	ADVANCED	EXPERT
Service level objectives (SLOs)	No SLOs formed.	Rudimentary SLOs formed which may not reflect user experience.	SLOs and error budgets are primary indicators of service reliability.	SLOs and error budgets are the primary driver of engineering decisions.
Vulnerability & misconfiguration scanning	No scanning.	Some infrastructure and applications scanned.	Most infra and apps scanned.	Continuous scanning of all infra and apps.
Security monitoring	Security metrics (e.g. failed logins) not defined or visible.	Security metrics are partially defined and visible.	Security metrics defined and partially visible for 100% of services.	Security metrics defined and fully visible for 100% of services.
User experience	No visibility into end-to-end customer journeys.	Partial visibility into some customer journeys.	High visibility into most customer journeys.	Full visibility into all customer journeys.
Data model & access	Data is uncorrelated, and ingested into separate systems owned by separate teams and not shared.	Some common datasets, but not easy to correlate, search, and filter. Frequent context switching.	Common data platform with a metadata model, usable by most teams.	Mature metadata model, via the use of tags or labels, that is usable by all teams.
Incident management	Incident detection and remediation times excessively long and not precisely known.	Incident detection and remediation times improving but not precisely measured.	Incident detection and remediation times low and roughly measured.	Incident detection and remediation times very low and rigorously measured.
Post-mortems	No formal template or process for post-mortems.	Inconsistent post-mortems that are not entirely blameless or clear.	Blameless post-mortems created in a timely manner.	Blameless post-mortems created in a timely manner with clear action items.