Checkmarx

The world runs on code. We secure it.

The Many Facets of Modern Application Development

A Comprehensive Guide for Leaders and Practitioners – Part 1

Table of Contents

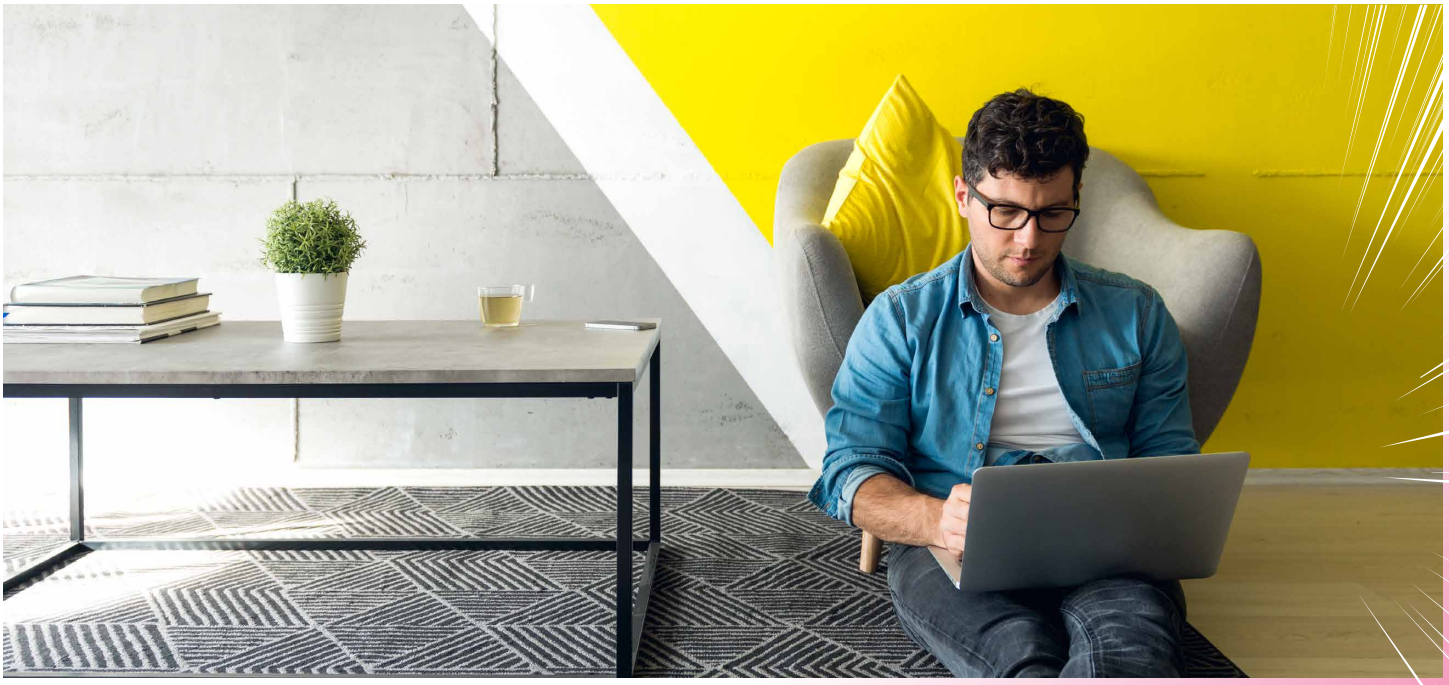
Introduction

In a world that has evolved on the physical and logical underpinnings of the internet, we're completely dependent on software applications (apps) that run nearly every aspect of our lives. Few would debate that software is the foundation of the future.

For organizations that have evolved into software-driven entities, the marching orders are clear: make faster, smarter, and easier-to-use apps to influence revenue and market share. As a result, the demands driving innovative app development, delivery, and deployment are moving at a previously unimaginable pace. Free from the chains of the internet's underlying infrastructure, the potential of unprecedented usability, scalability, and scope are limitless.

Amid the obligations to revolutionize, modern application development (MAD) was spawned and has since grown beyond its adolescence to stand at the forefront of nearly every public- and private-sector organization that thrives on software it creates. A new approach to creating value through software is here to stay, and MAD holds the key to modernization. Those who embrace this new era will reap the benefits, and those who don't will likely be compensated with a high margin of failure.

In Part 1 of this e-book, we'll uncover the many facets of MAD to help you build a foundational understanding of the concept, including how your organization can begin to move to this new development and delivery paradigm. The topics we'll cover will be of particular interest to those in leadership positions, but you'll also find a great deal to consider as a practitioner. Organizations that embark upon a MAD journey must acknowledge that moving slowly and cautiously may keep them from ever crossing the finish line. In the world of MAD, nothing moves gradually.



What Is Modern Application Development?

In the new age of software development, imagine developing applications capable of running anywhere—and on any commodity infrastructure—at scale. MAD brings this into reality through cloud-based, native facilities that disassociate the services software provides from previous design, development, and deployment practices, ultimately allowing teams to operate at greater scale than previously possible. Functioning under the notion that everything is code, MAD assumes nothing is defined with the old operational aspects of previous monolithic software builds.

In the past, software was expected to be developed in a traditional, well-defined fashion, often deployed on-premises in a data center. That approach required a host of physical servers to run the various functions code performs: one server might be used for the web-based user interface, another for a data access layer, another to provide a data store, and so on.

MAD isolates software innovation from operational boundaries, enabling teams to spend more time building innovative features and functions instead of wasting time considering the effects on underlying systems. In MAD, small malfunctions don't cause or influence system outages because services are decoupled from what they are running on. Requiring developers to fundamentally shift the way they think, MAD is all about the creation of new transactional value.

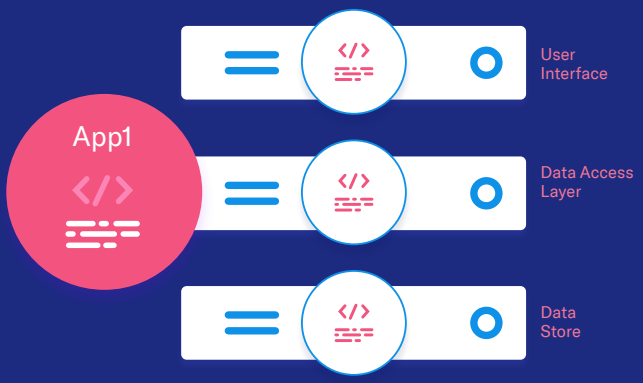
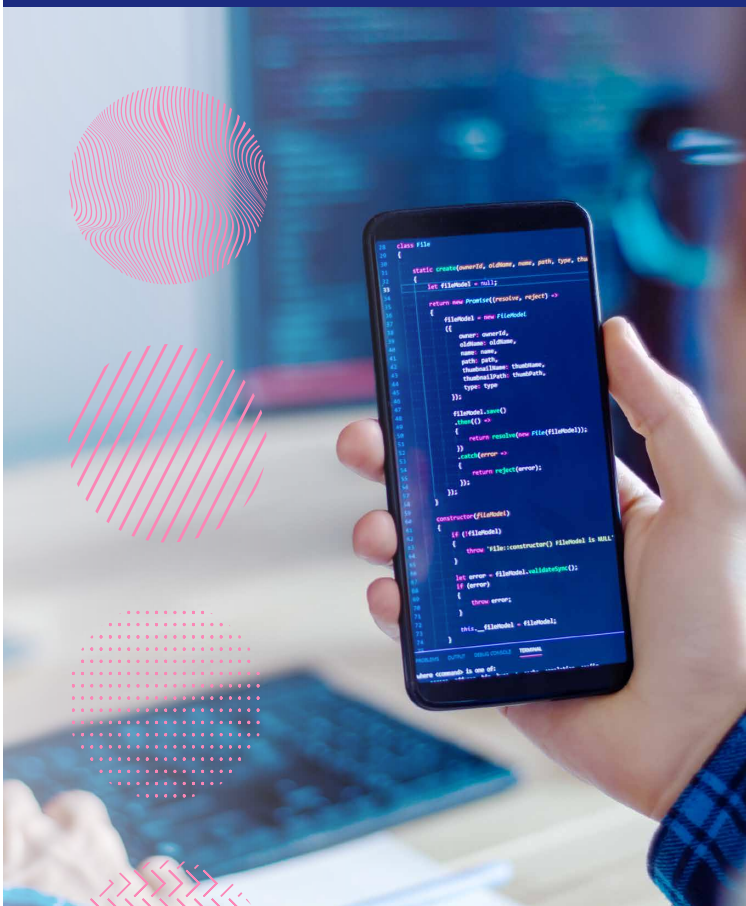


Figure 1 provides a conceptual view of what a traditional architecture might look like. Today, MAD changes all this.



MAD Speed

MAD operates at ridiculous speeds. Initially, this can stress developers out because of the linear mindset ingrained from their previous experiences. Everything is integrated and automated in MAD, whereby all delays caused by humans are removed or overcome. MAD influences development, delivery, and development practices so radically that many may not initially accept its promises due to sheer lack of understanding or fear of the unknown. MAD is like delivering five years' worth of software development in one year.

Now that we've taken a quick look at MAD from a theoretical perspective, let's dig in a little deeper and compare MAD to what is commonly called cloud native.

MAD vs. Cloud Native

In the past, people in IT and operations would deploy servers with operating systems and applications running on top. Cables, switches, routers, load balancers, and more were physically deployed and implemented. Today, all that effort can be abstracted into lines of code that account for everything needed to run an application and provide a service.

In comparison to MAD, cloud native is a methodology of building and running applications that exploits the advantages of the cloud computing delivery model. Software development in a cloud native environment still includes the conceptual initiatives of DevOps and certainly capitalizes on continuous integration, delivery, and deployment (CI/CD) fundamentals. In addition, cloud native can often be considered a subset of MAD because not all projects are fully designed to be cloud native. Software designed to be deployed on-premises, solely in the cloud, or in hybrid environments can still be developed using MAD fundamentals and approaches.

The Cloud Native Computing Foundation says: "Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil."[1]

Although cloud native technology is undoubtedly a component of MAD, let's uncover some of the drivers bringing MAD to the forefront of innovative organizations.

[1] "Who we are," Cloud Native Computing Foundation, accessed August 30, 2021, https://www.cncf.io/about/who-we-are.

Why MAD now?

Traditional software development can often be compared to assembling a train in a railyard. Train cars (autorack, boxcar, centerbeam, flatcar, etc.) are physically connected to an engine one at a time in a line, eventually assembling a long train. Connecting the cars, ensuring they are operational, and releasing the train to safely travel is a long and cost-prohibitive process, but simply put, the train cannot proceed to its destination until everything is assembled.

Traditional methods of developing software were similar. Organizations hoping to deploy a new software build quickly saw delays as teams poured effort into low- value deliverables and waited on other internal tasks to be done, but many of these delays had little to do with actually developing software. MAD was born to eliminate the linear inner workings by taking a parallel approach to delivery. Time-to-market demands in today's competitive market landscape mean a need for speed.

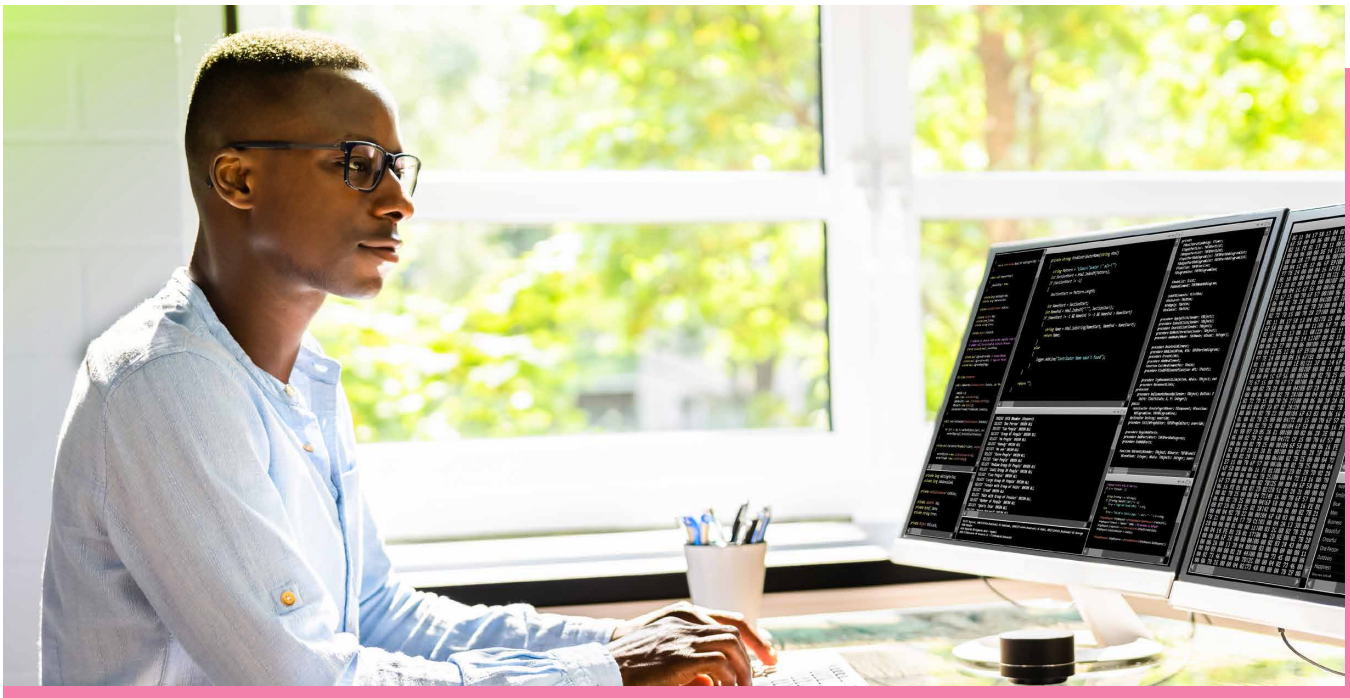
MAD empowers developers to take advantage of cloud native technologies to accelerate the design process, and then applies the same logic to the modeling, building, and delivering phases. It lets developers innovate through a componentized approach to optimizing software development, accelerating delivery, and improving the quality of their deliverables. It tears down the walls between development, operations, and infrastructure—and with digital transformation in overdrive, organizations are beginning to understand the importance of MAD initiatives. Moreover, MAD influences and inspires changes to the traditional culture. Let's delve into the cultural changes needed to make MAD initiatives successful.

MAD Culture

Companies often used to assume that if they built something, people would buy it. In today's world of software and services, that is no longer the case. MAD reverses that concept with a customer-first focus. Working backward from the customer's point of view, companies make business decisions based on a constant stream of customer feedback. This way, they can continually iterate their products and services to deliver what customers want and expect. With this in mind, what type of culture does MAD need?

The main cultural shift required for a successful MAD environment is to enable innovation through ownership. In the past, developers worked primarily on "projects"— they received a task, they worked on it, and they delivered the result. However, in MAD, developers take ownership of the products they are to deliver, and they are held accountable for the development of the whole product and its overall success. They have autonomy to determine how and where their product runs, in addition to continuously maintaining and improving it to meet ever-changing customer demands. This approach to software modernization means developers can work in an atmosphere where experimentation thrives instead of worrying about non–business value aspects.

This cultural shift fosters ownership of innovative outcomes. Empowering developers to deliver new and exciting customer results with more autonomy opens the door to more creativity, which flourishes in a culture increasingly inclined to accept risk. Trust is at MAD's very core, so developers will no longer feel like they're just a piece of the puzzle. Instead, they're the designers of the puzzle, and it's ultimately up to them how all the pieces fit together. MAD can generate a culture developers want to work in and often thrive in, which requires a new operational architectural model. Let's look at that concept next.



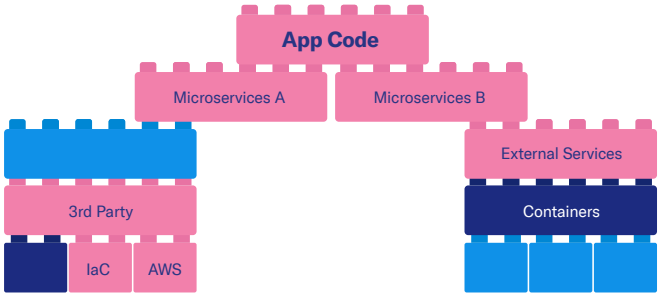
MAD Architecture

An iterative mindset is at the very core of MAD, and it requires a sort of rinse, improve, and repeat mentality. Ensuring internal and external customer feedback loops are in place, MAD folds in the feedback during the mixing process, which happens by design as MAD is driven by increasing levels of automation. MAD is all about the quality of the end product since continuous functional and application security testing are foundational elements.

MAD does not rely on a monolithic, linear approach to software development and deployment. Instead, you could compare it to something built from Legos. Each block has a slightly different purpose, but they all fit together perfectly because they are made with that intention in mind. The components of MAD that supportthis new architecture are as follows:

- **Applicative code**
  - Open source (third-party) code
  - Microservices/Serverless code
  - API code
- **Container code**
- **Infrastructure as code**

As shown in figure 2, in the MAD “legolized” architecture, each piece has its intended purpose, and they all fit together to deliver the desired outcome. Developers can update, improve, and deploy the various code pieces on the fly as needed, and the speed and flexibility MAD offers is a complete game changer.



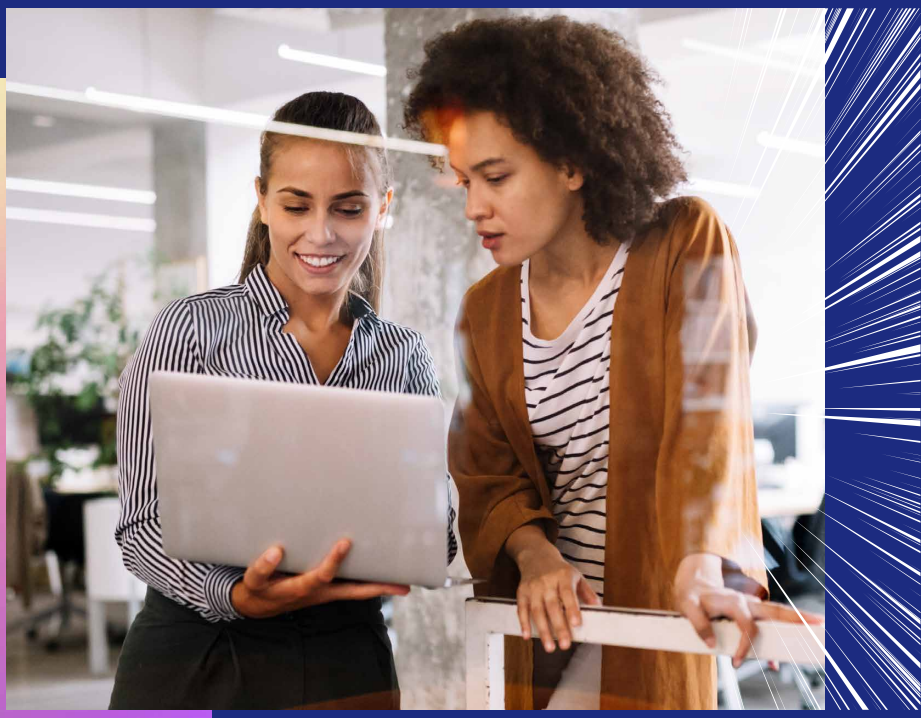
Now that we’ve discussed many of the positive attributes of MAD, we should also look at the challenges organizations may face with MAD initiatives.

MAD Challenges

Although MAD is considered revolutionary, its result is still software. Inputs and outputs, networking, storage, databases, and so on still exist in MAD outcomes. Developers enjoy the freedom MAD offers, but as they become more accustomed to using development frameworks, the abstractions keep increasing. In other words, when a deployed application is not working as desired, an organization's developers may not know how to quickly fix it since they may not fully comprehend the underpinnings of how that application worked in the past.

For example, in the days of writing code in COBOL and using a COBOL assembler, developers who were intimately familiar with the code could rapidly fix a particular problem. MAD keeps abstracting this further and further away. Take JavaScript frameworks for another example: there are so many abstractions that when things start to break, it can take much longer to troubleshoot. Developers must fully understand the dependencies, interdependencies, dependencies of dependencies, and so on, which can present a significant challenge.

In the past, development of an application to operate in a data center running on a three-tier architecture was often done by one development team. Likewise, security fell to one AppSec team, and support for the deployed application to one operations team. Now, everything is decentralized and all the frameworks in use can have different development, security, and deployment requirements. In MAD, developers can choose the frameworks and languages they want to use, further adding to the overall complexity of the final product. Before moving on, we should take a closer look at MAD abstractions.

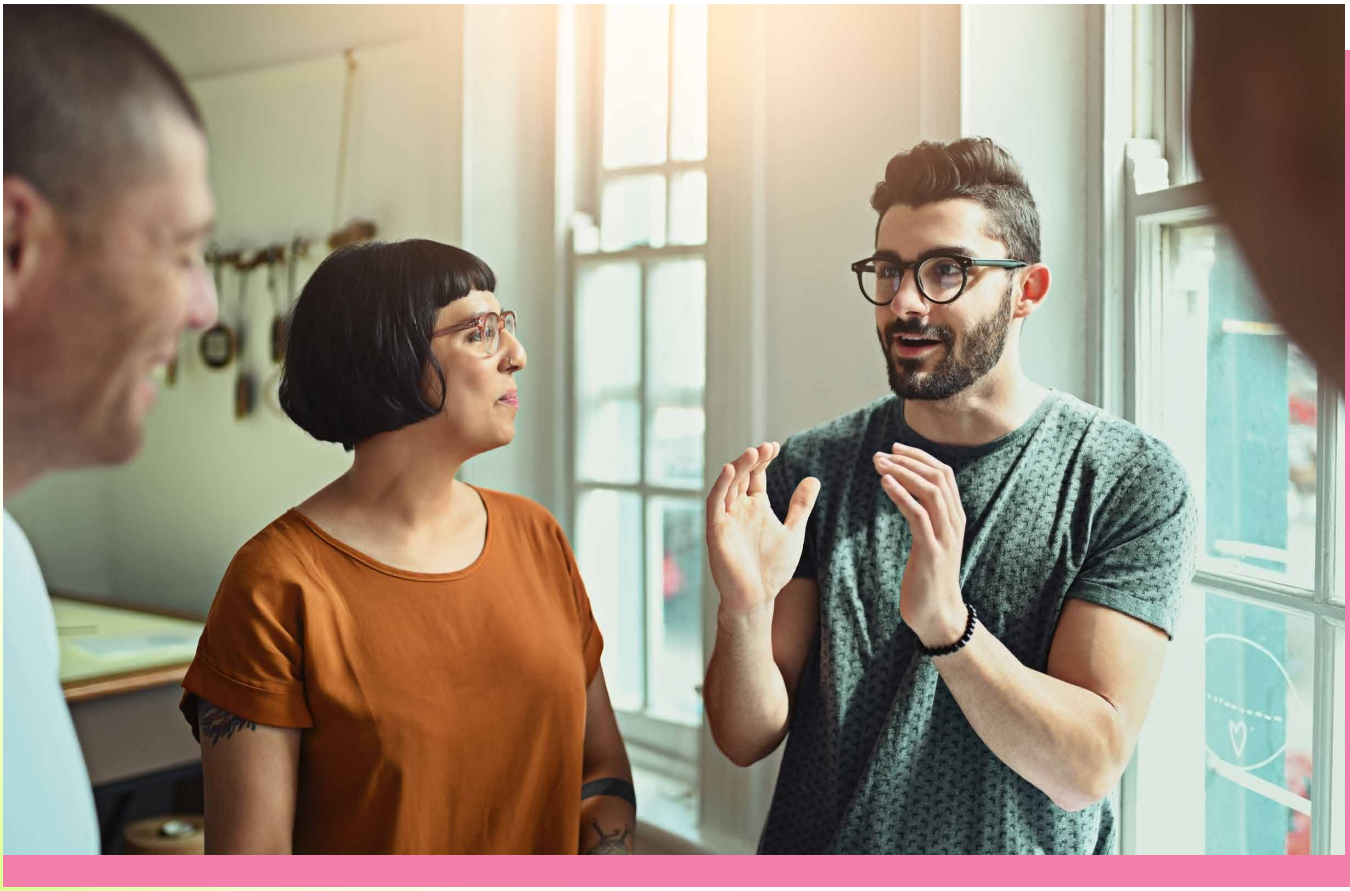


MAD Abstractions

Remember the days of building a data center network using a Visio diagram, following the design verbatim, and plugging hundreds of Cat 5 cables into the switches, routers, load balancers, and servers in a 19" rack? Now, that entire process has been summarized into simply calling an infrastructure as code function. That's the type of abstraction we are talking about here.

Suppose an organization suddenly realizes that its Amazon Elastic Block Store (EBS) no longer works with its Amazon Elastic Compute Cloud (EC2) due to a recent change. Because developers are abstracted so far from the result and lack necessary knowledge, no one in the organization knows how to rapidly fix the issue. Then, when development moves from using an assembler in the past, abstracted all the way up to Golang for example, there are abstractions of abstractions. When a production incident occurs, you'll often hear, "I only know this layer and not the rest."

Because of this abstraction, developers often don't fully understand security, beginning with the basics at the network layer. Instead of developing hardened approaches through a comprehensive understanding of ports, protocols, and the network layer, they'll pervasively allow everything to get through using an any, any, allow all policy—effectively leaving the door wide open. These are just a few of the caveats of MAD. Let's move on to a few more.



MAD Scalability and Observability

Although speed is the name of the game in MAD, scalability and observability are an entirely different discussion. Unfortunately, in MAD, organizations must address how they scale, since many older practices don't scale as well as one might hope. It may take longer to provision, support, fine-tune, troubleshoot, and so on in MAD environments. When an organization moves to cloud native for instance, the support needs and complexity can be extremely high.

Beyond scalability, observability should be on everyone's radar. When there are stability issues—say, processes coming online and quickly dying—without an observability layer in place, many teams may say, "What just happened?" They will seriously lack the necessary details to keep things running smoothly. Simply put, the most complicated pieces of some newer architectures are the logging and observability capabilities.

When observing a monolithic application running in a data center, seeing the entire stack is relatively simple. Through monitoring software, developers, security, and operations teams can get a bird's-eye view easily, quickly, and dependably. In MAD, observability is completely different. Take microservices, for example. When potentially hundreds of microservices (if not more) are running to support a single application and many depend on other services up- or downstream, it's imperative to be able to observe all of them, no matter where or how they are deployed.

In MAD, organizations need a comprehensive view that can fully correlate all the running statistics and metrics into a single pane of glass so they can calculate and measure a healthy app over time. App-level visibility that can be associated with customer experience is optimal. Response rates, error rates, logs, traces, metrics, and more are key parts of complete visibility across an entire system. Beyond scalability and observability, let's look at another stipulation of MAD: tool sprawl.

MAD Tool Sprawl

Before embarking on a MAD initiative, you might think MAD will reduce or eliminate many of the tools in place to develop and secure code. However, you should expect the exact opposite. In the past, you needed only a handful of IDE, CI/CD, SCM, and AppSec tools. In MAD, suddenly, you might hear, "We need 15 new development tools we didn't need yesterday." And that's just the tip of the iceberg.

Take Kubernetes for example. Many devs know just enough about containers to begin using them more and more. The next big problem is when developers start extending Kubernetes, and start incorporating open source, and then suddenly management hears, "We need another tool (or two) in our arsenal to make this work."

Soon, everyone believes their tools are the greatest, but organizations still need to standardize—particularly to manage and reduce costs—and standardizing isn't as easy as it sounds. Most want to give their teams more flexibility, and the teams often fight for it because they want to be as agile as possible. One group will want their tools, another group will want their tools, and soon there are so many in play that they become completely unmanageable. Now that we understand some of the organizational challenges to be expected in MAD, let's not forget about an entirely new set of security risks that come into play. Let's look at an overview of MAD security risks next.

MAD Security Risks

In a corporate context, security risk—in the form of data loss, breaches, ransomware, etc.—is certainly top of mind. In software, security risk is primarily derived from exploitable vulnerabilities when the software is running in production. Regarding software risks like those highlighted in the OWASP Top 10, CWE/SANS Top 25, and other similar lists, organizations normally have application security testing (AST) processes in place to help detect and mitigate these risks.

These lists are excellent starting points to help developers and application security (AppSec) pros understand secure coding practice and how to mitigate software risk. However, and specifically pertaining to MAD, the risk landscape becomes increasingly larger due to a new set of risks that emerge. Beyond traditional, web-based software risks as highlighted in the OWASP Top 10, the outcome of MAD extends the landscape considerably beyond traditional risk.

Today, organizations must not only acknowledge the risks listed by OWASP (and others), but also an entirely new set of risks that emerge in MAD initiatives, such as:

> Open source code

- Inconsistent security standards
- Unknown source code origins
- Licensing noncompliance

> Microservices

- Expanding complexity
- Limited environment control
- Inappropriately securing data
- Inappropriately securing the network

> Containers

- Running containers from insecure sources
- Exposing sensitive data through container images
- Too much faith in image scanning
- Broader attack surface
- Bloated base images
- Lack of rigid isolation
- Less visibility

> Infrastructure as code

- Steep learning curve
- Human error
- Configuration drifts
- Exposing sensitive data and ports

> APIs[2]

- Broken object level authorization
- Broken user authentication
- Excessive data exposure
- Lack of resources and rate limiting
- Broken function level authorization
- Mass assignment
- Security misconfiguration
- Injection
- Improper assets management
- Insufficient logging and monitoring

[2] "OWASP API Security Top 10 2019," OWASP, accessed August 30, 2021, https://owasp.org/www-project-api-security.

OWASP Top 10 Web Application
Security Risks

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

We can manage some of these emerging risks by implementing better security-focused policies, processes, and procedures. Others, however, we can only solve by implementing an entirely new set of AST solutions, using a platform-based approach accompanied by a correlation layer specifically designed to address many other emerging MAD risks. Organizations making the transition to MAD will quickly realize that their current approaches to security testing don't always fit well in this model.

MAD Next Steps

In this e-book, we began with a discussion on the fundamentals of modern application development and explained why many organizations worldwide are adopting this new methodology. We highlighted the culture of MAD and briefly discussed some of its architectural components. Hopefully, we showed that while MAD offers many benefits, like anything that brings change, you should also expect challenges.

We then looked at these challenges and highlighted some of the issues that would likely surface. We also listed the expanding security risks organizations will face due to the way modern applications operate. We're absolutely not trying to dissuade organizations from moving to MAD. On the contrary, we fully support MAD initiatives—we use these same approaches at Checkmarx. Our AST solutions are built by developers, for developers, and used in both traditional and modern development environments.

MAD can spawn lengthy discussions since the concept is quite complex, but now you should have a better idea of what MAD is all about. In Part 2 of this e-book, we'll delve much deeper into each of the security-related risks mentioned in Part 1 and offer recommendations on how to manage and mitigate them. For now, it's a good idea just to acknowledge that a whole new set of risks exists.

About Checkmarx

Checkmarx is constantly pushing the boundaries of Application Security Testing to make security seamless and simple for the world's developers while giving CISOs the confidence and control they need. As the AppSec testing leader, we provide the industry's most comprehensive solutions, giving development and security teams unparalleled accuracy, coverage, visibility, and guidance to reduce risk across all components of modern software – including proprietary code, open source, APIs, and Infrastructure as Code. Over 1,600 customers, including half of the Fortune 50, trust our security technology, expert research, and global services to securely optimize development, at both speed and scale. For more information, visit our website, check out our blog, or follow us on LinkedIn, Twitter, YouTube, and Facebook.

Checkmarx

Checkmarx at a Glance

1,675+
Customers in 70 countries

750
Employees in 25 countries

45%
of the Fortune 50 are customers

30+
Languages & frameworks

500k+
KICS downloads in 2021

The world runs on code. We secure it.