Cloud Native Security

The speed and flexibility that are so desirable in today's business world have led companies to adopt cloud technologies that require not just more security but new security approaches. In the cloud, you can have hundreds or even thousands of instances of an application, presenting exponentially greater opportunities for attack and data theft.

Public cloud service providers have done a great job of taking on the build, maintenance and updating of computing hardware, and providing VMs, data storage and databases to their customers, along with the baseline security to protect it all. But it's still up to the customer to provide security for the data, hosts, containers and serverless instances in the cloud.

The following sections explore cloud native security issues including securing Kubernetes clusters, DevOps and DevSecOps, and visibility, governance, and compliance challenges.

The 4C's of cloud native security

The Cloud Native Computing Foundation (CNCF) Kubernetes project defines a container security model for Kubernetes in the context of cloud native security. This model is referred to as "the 4 C's of Cloud Native security". Each layer provides a security foundation for the next layer. The 4 C's of Cloud Native security are:

Cloud – The cloud (as well as datacenters) provide the trusted computing base for a Kubernetes cluster. If the cluster is built on a foundation that is inherently vulnerable or configured with poor security controls, then the other layers cannot be properly secured.

Clusters – Securing Kubernetes clusters requires securing both the configurable cluster components and the applications that run in the cluster.

Containers – Securing the container layer includes container vulnerability scanning and OS dependency scanning, container image signing and enforcement, and implementing least privilege access.

Code – Finally, the application code itself must be secured. Security best practices for securing code include requiring TLS for access, limiting communication port ranges, scanning third-party libraries for known security vulnerabilities, and performing static and dynamic code analysis.

DevOps and DevSecOps

In a traditional software development model, developers write large amounts of code for new features, products, bug fixes, and such, and then pass their work to the Operations team for deployment, usually via an automated ticketing system. The Operations team receives this request in its queue, tests the code, and gets it ready for production – a process that can take days, weeks, or months. Under this traditional model, if Operations runs into any problems during deployment, the team sends a ticket back to the developers to tell them what to fix. Eventually, after this back-and-forth is resolved, the workload gets pushed into production.

This model makes software delivery a lengthy and fragmented process. Developers often see Operations as a roadblock, slowing down their project timelines, while Operations teams feel like the dumping ground for development problems.

DevOps solves these problems by uniting Development and Operations teams throughout the entire software delivery process, enabling them to discover and remediate issues earlier, automate testing and deployment, and reduce time to market.

To better understand what DevOps is, let's first understand what DevOps is not.

DevOps is not:

A combination of the Dev and Ops teams. There are still two teams; they just operate in a communicative, collaborative way.

Its own separate team. There is no such thing as a "DevOps engineer." Although some companies may appoint a "DevOps team" as a pilot when trying to transition to a DevOps culture, DevOps refers to a culture where developers, testers, and operations personnel cooperate throughout the entire software delivery lifecycle.

A tool or set of tools. Although there are tools that work well with a DevOps model or help promote DevOps culture, DevOps is ultimately a strategy, not a tool.

Automation. While very important for a DevOps culture, automation alone does not define DevOps.

Now, let's discuss what DevOps is. Instead of developers coding huge feature sets before blindly handing them over to Operations for deployment, in a DevOps model developers frequently deliver small amounts of code for continuous testing. Instead of communicating issues and requests through a ticketing system, the Development and Operations teams meet regularly, share analytics, and co-own projects from beginning to end.

CI/CD pipeline

DevOps is a cycle of continuous integration and continuous delivery (or continuous deployment), otherwise known as the CI/CD pipeline. The CI/CD pipeline integrates Development and Operations teams to improve productivity by automating infrastructure and workflows as well as continuously measuring application performance.

Continuous integration requires developers to integrate code into a repository several times per day for automated testing. Each check-in is verified by an automated build, allowing teams to detect problems early.

Continuous delivery means that the CI pipeline is automated, but the code must go through manual technical checks before it is implemented in production.

Continuous deployment takes continuous delivery one step further. Instead of manual checks, the code passes automated testing and is automatically deployed, giving customers instant access to new features.

DevOps and security

One problem in DevOps is that security often ends up falling through the cracks. Developers move quickly, and their workflows are automated. Security is a separate team, and developers don't want to slow down for security checks and requests. As a result, many developers deploy without going through the proper security channels and inevitably make harmful security mistakes.

To solve this, organizations are adopting DevSecOps. DevSecOps takes the concept behind DevOps – the idea that developers and IT teams should work together closely, instead of separately, throughout software delivery – and extends it to include security and integrate automated checks into the full CI/CD pipeline. This takes care of the problem of security seeming like an outside force and allows developers to maintain their speed without compromising data security.

Visibility, governance, and compliance

Ensuring your cloud resources and SaaS applications are correctly configured and adhere to your organization's security standards from day one is essential to prevent successful attacks. Additionally, making sure these applications, as well as the data they collect and store, are properly protected and compliant is critical to avoid costly fines, brand reputation damage, and loss of customer trust. Meeting security standards and maintaining compliant environments at scale, and across SaaS applications, is a requirement for security teams.

Despite the availability of numerous tools, most organizations struggle to effectively control their data exposure and enforce security policies across ever-changing cloud environments and SaaS applications. Furthermore, ensuring compliance where data is stored across distributed environments puts a significant burden on constrained security teams.

Ensuring governance and compliance across multi-cloud environments and SaaS applications requires:

Realtime discovery and classification of resources and data across dynamic SaaS as well as PaaS and IaaS environments.

Configuration governance, ensuring application and resource configurations match your security best practices as soon as they are deployed, and preventing configuration drift.

Access governance using granular policy definitions to govern access to SaaS applications and resources in the public cloud as well as to apply network segmentation.

Compliance auditing, leveraging automation and built-in compliance frameworks, to ensure compliance at any time and generate audit-ready reports on demand.

Seamless user experience that doesn't force additional steps or introduce significant latency in the use of applications as you add new security