Buyer's Guide

Cloud Native Security Platform

Table of Contents

Introduction

This guide is intended to help DevOps, engineering, security, and compliance managers understand and define the capabilities required to secure their cloud native applications and the infrastructure on which those applications run.

New security challenges introduced with cloud native applications

Security teams are accustomed to planning and operating in an environment that, unlike a cloud native environment, is composed of discrete, infrequent releases with little open source, and in which workloads are persistent, the host has a permanent address, and the hypervisor or hardware is isolated.

Key security concepts for cloud native environments

Traditional environments	Cloud Native Reality	Cloud Native Security
Discrete, infrequent releases	CI/CD	Shifting left with automated testing
Persistent workloads	Ephemeral workloads	Contextual runtime controls that follow the workload
Permanent address	Orchestrated pods	Identity-based segmentation
Hypervisor or hardware isolation	Shared kernel, obscured OS	Enforce least privilege on each workload
Very little open source	Open source everywhere	Software composition analysis
Proprietary software	Reuse of public images & libraries	Software supply chain risk
Vertical control of the stack	Multi-cloud	Detecting cloud service account misconfigurations

But Security and DevOps teams must incorporate new concepts and methods, as well as take a holistic approach to integrate security into the fabric of the cloud native environment, from CI/CD to Kubernetes and cloud infrastructure. A summary of these new concepts includes:

Shifting left with automated testing

Frequent, automated releases where fixes are not made in production mean that security must shift left, testing earlier in the software development process. And security teams must work with developers and DevOps teams to become part of the CI/CD automation flow.

Software composition analysis and vulnerability management

Open source is everywhere, and many applications are assembled from open source components. The incorporation of open source code and the dependencies involved creates an entirely new attack surface.

Software supply chain risk

Developers will freely use others' images, which themselves may be based on third-party images, and so on, obscuring visibility into the contents and true activities of any image. Managing vulnerabilities, protecting against malware, and other classic security concepts must take this supply chain risk into account.

Contextual runtime controls that follow the workload

In a cloud native environment, workloads are ephemeral and are spun up and down as needed, pods are orchestrated, and the kernel itself is shared. As a result, the traditional, primary focus on the hosts and the OS must be replaced with focus on:

- Networking activity at the pod, container, orchestrator, and host levels
- Differences between the intent of the image or container in the build versus the actual behavior in runtime
- Runtime policies for containers, VMs, and functions

Identity-based segmentation and enforcing least privilege on each workload

Identity verification must function on every asset that is trying to access resources on your network, regardless of whether it is inside or outside a network perimeter. This means assigning appropriate behavior across subnets, security groups, resource groups, and labels so that traffic can be controlled while pods are continually orchestrated.

Multi-cloud environments

Organizations are working strategically across multiple cloud providers to take advantage of cost efficiencies, optimize performance, and avoid being locked in with one vendor.

Now that we have a summary of the requirements for a cloud native security solution, we need a framework to gauge an organization's capabilities and needs in more depth.

Holistic, full lifecycle cloud native security

Cloud native security should involve a comprehensive set of capabilities across the build, infrastructure, and workloads.

Secure the Build

Prevention First

Build: All artifacts must be scanned for vulnerabilities, secrets, bad configurations, malware, and permissions. Issues must be prioritized. Developers must be enabled to shift left and move to a prevention-first model. Security teams must set policies to flag and block potentially harmful artifacts from making their way through the build pipeline. Supply chain attacks also must be detected and remediated.

Secure the Infrastructure

Harden the Environment

Infrastructure: In a cloud native environment, securing the infrastructure of public cloud, infrastructure as a service (IaaS), and orchestrators relies heavily on proper configuration. For example, accidentally enabling anonymous access to the kubelet could allow an attacker to gain control of an entire cluster. There are dozens of configurations for each orchestrator and cloud service, so checking automatically and continually for issues is critical. Remediation can be automatic or customized via policies, and compliance benchmarks for hardening must be met.

Secure the Workload

Protect Runtime Workloads

Workload: Workloads run in different places, so security instrumentation and interception points must be specific to each of these locations. Effective workload protection partially depends on appropriate protection in the build, knowing what a clean artifact (image, function, or VM) looks like to enforce its intended use and permissions.

An integrated platform is required to provide critical contextual information. For example, workload protection requires a full understanding of the contents of build artifacts to be able to enforce container immutability. And knowing what workloads are running gives critical context into which vulnerable artifacts should be prioritized for remediation in the build.

A great, initial heuristic for organizations in determining the effectiveness of their current cloud native security posture is whether they have controls across the build, infrastructure and workloads, and whether those controls share contextual information to increase the effectiveness of security across the board.

Critical capabilities

Scanning: Containers, VMs, and functions

Shifting left and incorporating security into the build via image and function scanning is essential to detect low-hanging fruit such as known vulnerabilities, malware, hardcoded secrets, and open source dependencies. Scanning also performs the critical function of populating profiles for artifacts so that policies aimed at controlling the behavior of those artifacts are accurate and effective.

Essential capabilities: Image and cloud VM scanning

- □ Detect vulnerabilities in open components
- □ Scan multiple languages and binaries, including C++, PHP, NodeJS, Golang, .NET, Java, and Python
- □ Scan OS packages (e.g., RPM, Deb, Alpine)
- □ Automatically scan Linux and Windows hosts for OS vulnerabilities, malware, and login attempts
- □ Acknowledge security risks in images and allow them to be remediated upstream
- □ Detect and enumerate sensitive data and secrets
- □ Provide an image bill of materials (e.g., packages, files, OSS license information, layer history)
- □ Detect if an image user is defined as root
- □ Scan VM and container images against configuration security best practices
- □ Use custom compliance checks (e.g., via SCAP, custom scripts) to scan for sensitive data
- □ Detect vulnerabilities in base images for fast remediation across all subsequent image builds
- □ View vulnerability origins throughout the image hierarchy and layers and validate the chain of custody
- □ Provide actionable remediation information on detected vulnerabilities
- □ Enforce effective security hygiene in the CI/CD build process via scanning plugins to any CI/CD tool (e.g., Jenkins, VSTS, Bamboo)

Essential capabilities: Serverless function scanning

- □ Scan functions to detect vulnerabilities, embedded secrets, configuration errors, and sensitive data
- □ View relationships between a function and its dependencies for end-to-end vulnerability visibility and traceability
- □ Detect risks and amend identity and access management privilege issues associated with functions
- □ Detect secrets embedded in functions
- □ Generate an audit trail of all scan events, vulnerability status, scan timelines, and remediation trends

Key considerations: Scanning

- Does the artifact (image, function) include sensitive data or embedded secrets (e.g., API key, SSH key)? Do you have visibility into these issues for serverless code?
- Are there vulnerabilities in the base image itself?
- Are there vulnerabilities in open source components that were added?
- Can your developers see the key remediation information required to continue shipping code quickly from within their preferred CI/CD tools?
- Do you have insight into OS packages in the same solution that provides data on vulnerabilities and application dependencies?

Risk-based vulnerability management

Gartner's vulnerability management model suggests that to effectively manage vulnerabilities, you must prioritize them based on their relevance in your environment; choose whether to accept the risk or to mitigate or fix the issue; and then assess the results to improve your accuracy and effectiveness in identifying and prioritizing vulnerabilities.

Essential capabilities: Risk-based vulnerability management across images, Kubernetes, and hosts

- □ Track attempts to exploit vulnerable packages
- □ Check for vulnerable, deprecated versions of Kubernetes
- □ See the severity of vulnerabilities on the host
- □ Use multiple resource feeds for scans (e.g., public CVEs, vendor-issued, proprietary vulnerability data streams, malware) to achieve refined results
- □ Get detailed information on a vulnerability's exploits and the available fixes
- □ Use vendor's specific severity ranking as a factor in prioritizing vulnerabilities
- □ Prioritize vulnerabilities based on actual risk to the environment (e.g., exploitable workloads)
- □ Visualize running workloads with a live map to better identify the security posture of each component
- □ Provide the capability to remediate, mitigate, or acknowledge the vulnerability upon discovery
- □ Automatically mitigate vulnerabilities with surgical policies that can prevent exploits in a non-intrusive way
- □ Use mitigation capabilities as a feedback loop to further prioritize highest impact vulnerabilities for remediation

Key considerations: Prioritizing vulnerabilities

- Can you get insights on actively used packages?
- Does your runtime environment have workloads that are impacted by the CVE?
- From a single dashboard, can you review all running workloads and assess the security posture of each component?
- How easily can you search for a specific vulnerability or a software component that is running in many containers?
- Can you filter business-critical applications and view a simple list to prioritize contextual risks?
- Can you stop an exploit from occurring without fixing or patching the vulnerable artifact?
- How will you prioritize and review only the vulnerabilities that are relevant to running workloads?
- Can you check whether you are running a vulnerable, outdated version of Kubernetes?

Implementing assurance policies for cloud native applications

Assurance policies are critical for both security and DevOps teams to create a shared sense of trust around the guardrails for allowing artifacts into production. In an environment where implementing trusted code is critical across multiple pipelines and rapid code commits, using assurance policies to control the parameters of what should and shouldn't be allowed will lower the overall amount of noise, ultimately reducing the overall attack surface and making runtime policies more effective. Assurance policies should apply across images, VMs, serverless functions, and orchestration tools such as Kubernetes.

Essential capabilities: Real-time image assurance policies

- □ Scan images before deploying them in production
- □ Auto-scan images that did not originate from registries in your environment
- □ Rescan modified images before reintroducing them into production
- □ View timeline and scan results of both registry images and user CI image scans
- □ Control the criteria for which images are scanned (e.g., image creation date, image name or tag, schedule)
- □ Automatically scan private registries upon image push across any platform
- □ Discover and maintain an up-to-date inventory of image repositories
- □ View the relationship between an image and its base image
- □ Prevent unapproved images from running in your environment
- □ Create flexible rules based on the security needs of different applications
- □ Enable multiple image assurance policy settings (per image name, label, registry) for effective mitigation
- □ Provide a broad set of predefined image assurance policies (e.g., for popular container images)
- □ Use image labels to restrict usage in certain environments (e.g., production, non-production)

Key considerations: Image assurance policies

Is every artifact (image, function) that is used in the CI/CD pipeline scanned?

- Did you scan all image registries to ensure that images that either skipped the CI/CD process or have gone stale are secure?
- How can you prevent users from running images from outside the pipeline?
- Can you customize the scan time of your images?
- Can you define which images will be pulled from your registry for scanning?
- Does the image include only the executables required for its ongoing operations?
- Can you stop images from being used based on CVE severity and risk scores?
- Can you customize assurance policies based on the security needs of different applications?

Essential capabilities: Serverless assurance policies

□ Automatically discover and maintain secure function inventory

□ Protect functions without altering the functions' code

□ Identify abnormal usage trends in function runtime duration and invocation frequency

□ Detect and block attempts to run malicious executables via functions

□ Block malicious code injection

□ Prevent non-compliant functions from running

Key considerations: Serverless assurance policies

- Can you automatically fail a function if it contains sensitive data?
- Can you prevent a function from significantly changing its form and purpose in runtime?
- Can you prevent functions from containing excessive permissions?

Essential capabilities: Virtual machine assurance policies

- □ Automatically discover and maintain secure function inventory
- □ Protect functions without altering the functions' code
- □ Identify abnormal usage trends in function runtime duration and invocation frequency
- □ Detect and block attempts to run malicious executables via functions
- □ Block malicious code injection
- □ Prevent non-compliant functions from running

Key considerations: Virtual machine assurance policies

- Can you ensure least privilege for OS and host resources?
- Can you prevent the creation of new, compromised hosts?

Essential capabilities: Kubernetes assurance policies

- □ Protect Kubernetes workloads that don't meet assurance policies based on pod or node configuration
- □ Apply out-of-the-box best practice rules for secure Kubernetes configurations
- □ Address least privilege security gaps for Kubernetes privileges access
- □ Reuse and standardize on Rego rules used with the Open Policy Agent (OPA)
- □ Block DDoS attempts with CPU limits and specified resource requests
- □ Enforce segmentation by label hosts and containers as production or non-production, by sensitivity level, purpose, etc.

Key considerations: Kubernetes assurance policies

- Can you determine, with granularity, least privilege access across all the options for Kubernetes workloads?
- Are you able to make least privilege remediation recommendations based on best practices?
- Are your Kubernetes policies written in Rego, to ensure OPA compatibility?
- Can you create or import your own custom Kubernetes policies?
- Can you integrate your Kubernetes policies into the CI/CD process?

Protecting against supply chain attacks

It's common practice for developers to recycle the base images of popular projects from public libraries or third parties in order to work faster. But this can inadvertently insert malicious images into the build, and attackers have found ways to evade traditional scanning.

To check images while importing them from third-party sources, or to set up a last stop for the production registry before going into production, a container sandbox should analyze how an image would behave in runtime.

Essential capabilities: Dynamic image analysis

- □ Run images in a secure sandbox before production to see detailed information on malware tactics
- □ Detect behavioral anomalies and encrypted or obfuscated files that execute inside the container while it's running
- □ Find container escapes, reverse shell backdoors, malware, cryptocurrency miners, and code injection backdoors, packers (including encrypters), and downloaders
- □ See the results of a forensic analysis on the behavior of an image across the MITRE ATT&CK framework, detailing tactics such as command and control, sleep tactics, etc.
- □ Detect malware and zero-day attacks without known signatures
- □ Prevent the use of malicious base images in supply chain attacks

Key considerations: Dynamic image analysis

- Can you see the runtime behavior of images before running them in production?
- Can you use dynamic analysis results to mark images as compliant or non-compliant in your CI/CD pipeline?
- Can you identify malicious behavior in the build that is guaranteed to be present in production?
- Can you find the exact cause of an image's malicious behavior and perform forensic analysis?

Secrets management

Managing secrets, such as API keys and security tokens, is particularly challenging in cloud native environments due to the dynamic and ephemeral nature of containers and functions. And while secrets are generally included in image and serverless function scanning, secrets management in runtime is many times an afterthought when designing a cloud native environment. Passwords are set up manually and shared through e-mail, and password rotation is often ignored because it is time-consuming and error prone.

Common security challenges for secrets include:

- Storing secrets inside a container image or function risks exposing secrets to anyone with access to that object, the registry, or CI/CD pipeline, and to potential intruders.
- Embedding secrets in an image means the image is tied to the life cycle of the secret. To rotate a secret, a new image must be built.
- Providing secrets as an environment variable when running a container exposes the secret to end users, since it's a common practice for software to log its entire environment.

Essential capabilities: Secrets management

□ Inject and rotate secrets in runtime (no downtime or restart)

□ Encrypt secrets in transit

□ Integrate with the preferred secrets store (e.g., CyberArk, HashiCorp, AWS KMS, Azure Vault)

□ Ensure that secrets are visible only inside the workload that uses them

□ Ensure that secrets are not accessible via the network, host, or orchestrator

Key considerations: **Secrets management**

- Where are secrets stored today? Are they hardcoded into images?
- Can you map secrets to relevant containers?
- Are secrets activities logged (e.g., secrets delivery, rotation, revocation)?
- Is there a roll-back procedure?
- Once a secret is revoked, how do you confirm it is deleted from volumes and that access permissions to resources are also revoked?
- If a secret is rotated or revoked, do you have an automated way to propagate the update or revocation to all relevant containers that use the secret?
- Can you ensure that secrets do not persist on the host once the container is spun down?
- How do you ensure that secrets are delivered and rotated based on your established security policies?
- Can routine rotation be done with no downtime to the running container and with no restarts?
- Can you ensure that only certified, designated running containers can retrieve secrets?

Aqua recommends mounting secrets as tmpfs volumes, where they're accessible to the application as a virtual "file" resident in memory. Most orchestrators use this delivery method for secrets to containerized applications.

Cloud account security

A single misconfiguration in a cloud account or IaaS can expose a system or your sensitive data to the outside world. Cloud security posture management (CSPM) solutions are specifically designed to protect against such misconfigurations, scanning for threats in cloud account and infrastructure as code (IaC) configurations to ensure compliance and best practices. To prosper in advanced, multi-cloud environments, it is critical to receive alerts on issues and fix or remediate the riskiest misconfigurations automatically.

Essential capabilities: Cloud account security

- □ Gain visibility into hundreds of configurations across multiple services in multiple clouds
- □ See configuration issues in popular IaC solutions (e.g., Terraform, AWS CloudFormation)
- □ View configuration issues in light of compliance requirements and best practice guidelines
- □ Detect risks and implement security features before infrastructure is launched
- □ Gain immediate visibility into the severity and impact of misconfigurations
- □ Set up alerts and notifications for issues as your cloud services evolve
- □ Set up auto-remediation to fix the biggest issues quickly via a RESTful API
- □ Integrate with third-party monitoring services

Key considerations: Cloud account security

- Is there a central place where you can view the riskiest misconfigurations across multiple cloud providers and service accounts (e.g., AWS, Azure, GCP, Oracle)?
- Do you have a remediation process for misconfigurations? Is it manual or automated? If automated, is it via a RESTful API?

  - How do you delegate specific permissions for each API key?
  - Can you track each API call?
- Are you aware of all the accounts you have on each cloud (e.g., AWS accounts, Azure subscriptions, GCP projects, Oracle Cloud compartments)?
- Can you integrate cloud service configuration issues into a CI/CD pipeline?
- What compliance and regulatory requirements do you have to adhere to with your cloud infrastructure? Can you access compliance reports via API?

Kubernetes security posture management

Kubernetes-based infrastructure is a complex system that is spread across an IT infrastructure. Some Kubernetes components are close to the OS system, some are internal configurations, and some run as workloads. Many Kubernetes settings are non-secure by default, making security difficult and confusing for both DevOps and security teams. Kubernetes security must address these configurations with policies across namespaces, nodes, containers, and network connections.

Essential capabilities: Kubernetes security

- □ Audit all Kubernetes events
- □ Gain visibility into vulnerable Kubernetes versions
- □ Conduct automatic penetration tests of your Kubernetes clusters against a variety of attack vectors
- □ Assess the Kubernetes environment according to the CIS Kubernetes benchmark, and provide daily scans and a detailed report with the findings
- □ Gain information into clusters, registries, hosts, namespaces, running containers, and the images the containers are based on automatically, with minimal user intervention
- □ Perform risk analysis on the Kubernetes environment without requiring root privileges on the host OS
- □ Provide continuous visibility into the Kubernetes environment and gain clear visibility on any policy violations
- □ Assess the security posture of Kubernetes clusters and prioritize risks for action in real-time

Key considerations: Kubernetes security

- Can you be certain that your environment complies with regulatory requirements?
- Can you easily align with the compliance requirements of common standards or best practices, such as CIS benchmarks, PCI, NIST 800-53, or HIPAA?
- Do you have a holistic view of the security posture of the Kubernetes environment across its entire architecture, including host components, internal configuration, management API, and workload pods?
- Can you ensure that the container engine versions are up to date and fully patched?
- Are you able to perform automated penetration tests on your Kubernetes clusters using external Kubernetes APIs?

Runtime protection

Cloud workload protection platforms, as defined by Gartner, should protect cloud native workloads across a mixed environment of VMs, containers, and functions, with purpose-built controls for each. For example:

- Runtime controls for containers should protect against any behavior that's not in line with the container's original intent. And enforcement should happen with minimal management overhead and no impact to runtime.
- Protection for cloud VMs, for example Amazon EC2 instances, requires a lightweight solution that is unified in a single dashboard with other cloud native security controls, without the usual drag on cloud resources.

Essential capabilities: Runtime policies and controls for container images and functions

□ Enable rescanning of deployed images on hosts

□ Enable revalidation of image status (allowed, disallowed) before instantiation

□ Create and enforce zero-configuration container behavioral profiles

□ Prevent stale images from running

□ Disable and enforce unauthorized executables, network connections, ports, and file paths with no container downtime

□ Visualize across running workloads grouped by hosts, pods, Kubernetes namespaces, image, uptime, and status (stop, run) to prioritize response

□ Track malicious behavior of functions during runtime

□ Block malicious executables in functions during runtime

□ Protect workloads running in host and serverless or Container as a service (CaaS) environments (e.g., AWS Fargate, Azure container instances)

□ Detect and prevent containers and Kubernetes clusters from running unauthorized images (e.g., Bitcoin miners)

□ Create cryptographic image fingerprinting for all layers

□ Detect any changes to containers (e.g., binaries, hash, system calls) against their originating images

- □ Detect and prevent container privilege escalation and centrally manage and enforce seccomp profiles
- □ Set and control container memory, CPU consumption, and running process limits to prevent DOS attacks
- □ Control user activity and enforce segregation of duties
- □ Apply and enforce custom runtime policies per environment (e.g., create disallowed executables per namespace, disallow unregistered images in a PCI cluster)
- □ Apply an added layer of runtime controls per image type (e.g., all Alpine containers) to all running containers
- □ Associate containers to source code for end-to-end traceability and tamper-proofing
- □ Enforce policies in offline mode

Key considerations: Runtime protection (containers and functions)

- Can you ensure that cloud native containers and functions are granted the minimum necessary permissions?
- Is every image deployed into production authorized? Can you ensure that the latest, authorized image is being instantiated? How do you know if rogue containers are running?
- Do you know how to protect your runtime applications against attacks that are not based on known vulnerabilities, such as zero-day attacks based on host configuration errors, privileged user error, or insider threat?
- How will you prevent executables or unauthorized processes that are not in the original image from running, without downtime?
- How will you ensure that the cloud native apps were not changed unintentionally or maliciously after being deployed?
- Do you have visibility into what is running inside a container?
- If the image configuration has changed, can you prevent it from running?
- When considering a shift in app stacks, can you look for the impact or the use of specific components across all running containers?
- If Kubernetes is misconfigured, can you stop the workload from running?

Essential capabilities: Cloud VM security

□ Monitor selected operations on Windows registry keywords, values, and path attributes

□ Automatically scan the host for compliance against CIS benchmarks (e.g., Docker, Kubernetes, Linux)

□ Automatically discover cloud native objects and their security posture

□ Alert on suspicious host activities, such as brute force login attacks

□ Monitor files, file attributes, and directories on your hosts for read, write, and modify operations (FIM)

□ Monitor both user and application activities that are running on your VM, evaluate and provide remediation steps

□ Prevent kernel operation (e.g., do not allow “chown”)

□ Protect the integrity of the Windows registry

□ Export audit data for proof of compliance

Key considerations: Cloud VM security

- Can you ensure that the OS is up to date and fully patched?
- Can you provide the same level of security to your Linux and Windows OS?
- Can you restrict and monitor all runtime activities?
- Do you know how to audit user activity on your host?
- Can you ensure the integrity of files on the host?
- Can you ensure that image packages and libraries are authorized (i.e., patched and up to date)?
- Can you be sure that access control, networking, and authentication are all in check?
- Can you monitor and protect Kubernetes nodes and virtual machines with the same solution?

Identity-based segmentation

It is critical to be able to discover, visualize, and define network connections in cloud native environments, detecting malicious network activity across VMs, containers, Kubernetes clusters, and pods.

Essential capabilities: Identity-based segmentation

- □ Automatically discover and visualize the workload attack surface, relationships between namespaces, deployments, pods, and network traffic, and provide continual updates based on actual workload activity
- □ Label container groups as sensitive, use security group use security group definitions from the orchestrator (e.g., PCI-sensitive) and apply firewall rules accordingly
- □ Detect and prevent unauthorized network connections such as open ports (on the same or across hosts and pods) based on automated policies
- □ Define service-oriented firewall rules, membership scoping, bypass scoping, and more with cloud attributes
- □ Define which inbound and outbound ports are accessible to or from which IPs or URLs for the workload
- □ Define zero-trust network connections based on service-oriented firewall rules, regardless of where the workload runs
- □ Manually modify communication rules and policies based on actual activity, without affecting workload performance and availability
- □ Automatically alert on or block unauthorized communication flows (no container downtime)

Key considerations: Identity-based segmentation

- Can you detect and alert on unauthorized network traffic?
- Can you block unauthorized inbound or outbound communication to and from containers? For example, can you block an outbound connection to a database container?
- Can you block container (e.g., PCI-related) access to a specific IP address?

Integrations and environment support

Larger organizations require compatibility with current solutions and support for desired environments.

Essential capabilities: Integrations and environment

- □ Use the cloud provider, secrets vault, CI/CD, SIEM and analytics, and orchestrator of your choice
- □ Integrate out of the box with the private and public registry or registries of your choice
- □ Integrate across the ecosystem with full REST API support
- □ Integrate across your on-premises and multi-cloud environments
- □ Integrate with Jira and notification feeds (e.g., Slack, PagerDuty) for tracking tasks and issues
- □ Use SSO authentication to make team access easier (e.g., SAML-based authentication, OpenID Connect)
- □ Run a private threat intelligence feed for air-gapped environments

Key considerations: Integrations and environment

- Can you integrate your security workflows with your chosen cloud platform(s) and tools?
- Do you have the choice across cloud platforms that you require to meet your business goals?

Demonstrating compliance for cloud native environments

Regulators are not yet clear and consistent about how to demonstrate compliance in cloud native environments. Many certifications do not include specific guidance for cloud native environments, so it can be difficult to know where to start. The good news is that concepts such as hardening, networking, and vulnerability management are all relevant in cloud native environments. This pushes the challenge of compliance toward demonstration with visibility, auditing, and control.

Essential capabilities: Audit and compliance

- □ Provide full user accountability and controlled super-user permissions
- □ Get real-time alerts on policy violations
- □ Generate granular audit trails of all access activity, and scan events and coverage, Docker commands, container activity, secrets activity, system events, blocked image executables, blocked user access and replacement of high risk containers
- □ View built-in alerts and reports for key compliance mandates (e.g., PCI, GDPR, HIPAA, NIST SP 800-190)
- □ Use CIS benchmark (Docker, Kubernetes, and Linux) assurance policies and reports
- □ Gain audit data around the impact of CVEs and vulnerabilities
- □ Ensure timeliness of scans
- □ Track key changes in vulnerability status
- □ Maintain vulnerability vs. remediation trends

Key considerations: Audit and compliance

- Can you measure remediation trends based on KPIs (e.g., open vs. remediated)?
- How will you log user and container activity?
- Are alerts set up to warn of any behavior deviation in container activity?
- Do you keep a history of security configuration changes?
- Is the history related to your secrets rotation logged and stored?
- Do you have compliance visibility across Kubernetes, images, and hosts all in the same solution?

Separation of duties across teams using cloud native security

Large organizations also need to securely manage a solution across multiple, matrixed teams including compliance, security, development, and DevOps. App development and security teams in large enterprises consist of multiple groups working on different projects with segregated artifacts such as images and workloads. These projects are defined by namespaces and cluster, and it's important to granularly define permissions across teams.

Essential capabilities: Separation of duties

- □ Provide separation of duties for your cloud workload protection platform to limit access to super-user permissions, tasks, and cloud native resources (e.g., images, containers, nodes, networks, pods, volumes, orchestrators)
- □ Define central policies that are read-only for subordinate teams
- □ Create permission sets per role that provide view or edit access to policies, events, assets, and system components
- □ Provide self-service portals for auditors, security admins and developers to maintain segregation of duties while fostering collaboration
- □ Derive user access privileges based on application definitions in the orchestration system
- □ Assign Docker subcommands to users on a specific host
- □ Assign Kubernetes master node operations to users by cluster, deployment, and node
- □ Derive roles and privileges from existing AD/LDAP groups and authenticate users
- □ Log all access activity for investigations and regulatory compliance
- □ Provide log-in authentication via SAML SSO (e.g., Okta, Microsoft ADFS, Google Identity servers)
- □ Monitor and alert on unauthorized user activity

Key considerations: Separation of duties

- Can you assign and enforce user access permissions to cloud native security tools?
- Can you customize the user’s role in your cloud workload protection platform per your organization’s requirements?
- Can you assign Kubernetes master node operations to users by cluster, deployment, and node?
- How do you alert on unauthorized user activity on a host?
- Can you assign Docker subcommands to users on a specific host?

Summary

Cloud native security can seem intimidating and overwhelming, especially when many teams are responsible for different pieces of the cloud native application life cycle. To be successful, and agree on steps to close the security gap, a few other nuances are helpful to keep in mind:

- Partnering strategically with those who invest in the open source community means benefiting first-hand from innovation in the space.
- Cloud native security can - and should - match your consumption model strategy, whether you prefer on-premises or SaaS solutions.
- Cloud native security will block business goals if it can't scale along with the application or applications it is protecting.

Aqua Security helps enterprises secure their cloud native applications from development to production, whether they run using containers, serverless, or virtual machines. Aqua bridges the gap between DevOps and security, promoting business agility, and accelerating digital transformation. Aqua's Cloud Native Security portfolio provides full visibility and security automation across the entire application lifecycle and

infrastructure, using a modern zero-touch approach to detect and prevent threats while simplifying regulatory compliance. Aqua customers include some of the world's largest financial services, software development, internet, media, hospitality, and retail companies, with implementations across the globe-spanning a broad range of cloud providers and on-premise technologies.

Copyright ©2021 Aqua Security Software Ltd., All Rights Reserved