Buyer's Guide for Securing Cloud Native Workloads

The Purpose of This Guide

DevOps managers, security professionals, compliance managers, and security practitioners should use this guide[1] for the purpose of implementing and deploying security controls throughout their cloud workload environments. The guide will support the effort for assessing specific capabilities, identify gaps in workload security, and select the right solution to fill those gaps

Why cloud workload security should be important for your organization?

On the average, containers tend to live two days[2] and some live no more than 12 hours. This introduces the following challenges:

- *Fast-paced environment that is constantly changing*
- *Increased attack surface:*
  - Due to the exponential growth in the number of objects
  - Images are often built from open source software
- *Vulnerabilities within images are more common and can easily find their way into your production environment, due to the rapid development process*
- *The sprawl of numerous functions across cloud accounts on the one hand, coupled with event-driven, very short execution durations presents a new security challenge from the runtime aspect*

A compromised image becomes a compromised container, that container can gain access and compromise other containers, on the same host, or on other hosts.

How can you assess a cloud workload security solution?

- *A clear view of all active workloads and a snapshot of the security posture of your applications*
- *Combine image and host scanning with runtime protection, detection, and response capabilities*
- *Gaining insight on the image content, as well as the container and functions expected activities, controlling container communication*
- *Define user access based on roles and permissions*

These will provide your incident response teams the ability to quickly detect and respond to unauthorized activities, implement automation often automatically, and at the speed required to stop attacks in large scale, dynamic environments.

[1] If you would like to receive updates to this guide, please register here

[2] Source: Docker Adoption

What is covered in this guide?

Cloud workload security can be challenging for organizations, you are facing new and different range of threats, multiple technologies, and processes. It can be difficult to understand the features and capabilities that are important for your environment. This guide focuses on:

- *Vulnerabilities in container images, serverless functions and cloud VMs*
- *Secrets management*
- *Hardening the host, running VMs, and orchestrator environment*
- *User access permissions*
- *Controlling the runtime environments*
- *Demonstrating compliance for cloud workloads*
- *Understanding if your solution is enterprise ready*

In each section, we cover why this topic is important and review essential capabilities that you need to look for in your cloud workload security solution. We will also guide you with presenting questions that you need to ask when assessing the security posture of your workloads.

Vulnerabilities in Container Images

It is essential that developers and IT operations only use authorized images which were scanned and available from a trusted registry. In practice, security takes a back seat when pitted against speed. Developers often push images to the registry before they can be validated. When building a container image, developers could start from a base image, often Ubuntu or Alpine, or a common application (such as NGINX or JBoss). Frequently, the base image will be based on another base image and may even contain CVEs to begin with. If vulnerabilities are found in the image, it's quite possible they may have originated in a base image, or perhaps in that base image's own base image. Therefore, remediation should be applied at the point of origin.

Essential Capabilities for Vulnerabilities Management in Containers

Execute Real-Time Image Risk Assessment
Discover and maintain an up-to-date inventory of image repositories
View a live map of all running workloads and identify the security posture of each component
Automatically scan private registries upon image push across any platform
Automatically detect, and prevent, attempts to exploit vulnerabilities in a non-intrusive way
Use multiple resource feeds for scans (public CVEs, vendor-issued, proprietary vulnerability data streams, and malware) to achieve refined results
Scan images against configuration standards
Provide an image bill of materials (packages, files, OSS license information, and layer history)
- **View the relationship** between an image and its base image
- **Detect vulnerabilities** in open source components
- **Auto-scan images** that did not originate from registries in your environment
- **Scan** OS packages (e.g., RPM, Deb, Alpine)
- **Scan** 40+ languages and binaries including C++, PHP, NodeJS, Golang, .NET, Java, Python
- **Use custom compliance** checks (e.g., via SCAP, custom scripts) to scan for sensitive data
- **Detect and enumerate** sensitive data and secrets
- **Detect if an image** user is defined as root
- **Detect vulnerabilities** in base-images for fast remediation across all subsequent image builds
- **View vulnerability** origins throughout the image hierarchy/layers and validate the chain of custody
- **Acknowledge security** risks and propagate them to upstream images

- **Enforce effective** security hygiene into the CI/CD build process via scan plugins to any CI tool (e.g., Jenkins, VSTS, Bamboo)

- **Provide actionable** remediation information on detected vulnerabilities

View timeline and scan results of both registered images and user CI image scans

Image Assurance Policy Settings & Enforcement

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 Docker images)

Use image labels to restrict usage in certain environments (e.g., production/non-production)

Execute Function Risk Assessment

Automatically discover and maintain secure function inventory

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 IAM privilege issues associated with functions

Detect secrets embedded in functions

View actionable remediation information on detected vulnerabilities

Generate granular audit trails of all scan events, vulnerability status, scan timelines, and remediation trends

Identify abnormal usage trends in function runtime duration and invocation frequency

Detect a function's attempt to run executables during the function's invocation from /tmp and block it

Block malicious code injection

Consider the following key image-related security concerns:

- *Is every image used in the CI/CD pipeline scanned?*
- *Do you scan all image registries and hosts 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?*
- *Does the image include sensitive data, or embedded secrets (e.g., API key, SSH key)?*
- *Does the image include only executables required for its on-going operation?*
- *Are there any vulnerabilities in the base image itself? Are there any vulnerabilities in open source components that were added?*
- *Can your team know when an exploit attempt was made and automatically shield your environment?*
- *Can you have a clear single pane of glass for reviewing all your running workloads and assess the security posture of each component?*
- *Is the image configured with root/admin account by default?*
- *Can you stop images from being used based on CVE severity and risk scores?*
- *Can you automatically detect, and prevent, attempts to exploit the vulnerabilities in your system?*
- *Can you easily search for a specific vulnerability or a software component in many running containers?*
- *Can you prioritize vulnerabilities in your environment based on the impact that they present to your running workloads?*

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. There are numerus challenges when addressing secrets:

- *Developers usably store secrets inside a container image or a function which leads to the risk of exposing secrets to anyone with access to that object, to the registry, CI/CD pipeline, and to potential intruders*
- *When secrets are embedded in an image, the image lifecycle will be coupled with the secrets' lifecycle. If you want to rotate a secret, you must build a new image*
- *Providing secrets as an environment variable when running a container is not recommend since it is a common practice for software to log its entire environment, evidentially exposing the secret to end users*

"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 secrets delivery method to containerized applications"

Essential Capabilities for Vulnerabilities Management in Containers

Secrets Management
Secrets injection/rotation in runtime (no container downtime/restart)
Manage and monitor container secrets activity
Encrypt secrets in transit
Provide secrets store integrations (e.g., CyberArk, HashiCorp, AWS KMS, Azure Vault)

Consider the following key secrets related security considerations:

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

Overprovisioned User Access Permissions

Limiting the scope of user permissions can reduce the impact of errors or malicious activities. Knowing which users (e.g., developers, admins, auditors) can access which container resource and their set of permissions (command level) is a critical step in maintaining the Segregation of Duties and monitoring user activity. Setting granular role-based user access permissions (e.g., start a container) per container resource or services (cluster of containers) on the host, enables you to automatically identify misuse or abuse of these permissions.

In addition, logging all activity and generating alerts on unusual user, can help security analysts quickly review and investigate potential incidents.

Essential Capabilities When Assessing User Access

User Access Management
Provide Role Based Access Control (RBAC) to limit super-user permissions, tasks, and container resources access (e.g., images, containers, nodes, networks, pods, volumes, orchestrators)
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 sub-commands 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 MFA via SAML SSO (e.g., Okta, Microsoft ADFS, Google Identity servers)
Monitor and alert on unauthorized user activity

Key access-related security considerations are:

- *Can you assign and enforce user access permissions to container resources?*
- *Can you assign Docker sub-commands to users on a specific host?*
- *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?*
- *How do you ensure role-based access control to the container runtime environment?*

Hardening the Host, Running VMs, & Orchestrator Environment

Securing both the hosts, and the orchestrator components, is an integral part of your organization's workload security initiative. A vulnerable host that resulted from an unpatched operating system, for example, can expose a large attack surface to exploits. Attackers could change system configuration, install malware, or extract unsecured secrets stored in the memory. To prevent a container security breach from impacting the host, you need to secure the host OS.

For additional recommendations please refer to Aqua's free, open-source tool, Kube-bench, for Kubernetes nodes assessment against CIS Kubernetes benchmarks.

Essential Capabilities for Securing the Host and Orchestrator

Host Security
Automatically scan hosts for OS vulnerabilities, malware, and login attempts
Evaluate the scanning findings and provide remediation steps
Protect both Linux and Windows hosts
Automatically scan the host for compliance against CIS benchmarks (Docker, K8s, & Linux)
Automatically scan hosts for malware
Control and monitor container access rights to OS and host resources
Control and monitor user logins to host
Monitor files, file attributes & directories on your hosts for read, write, & modify operations
Alert on suspicious host activities, such as Brute Force login attacks
Label hosts and containers as production/non-production, sensitivity level, purpose, etc. to enforce segmentation by the orchestrator

Examine the following host related security considerations:

- *Can you ensure that the OS, as well as the container engine versions, are up-to-date and that they are fully patched?*
- *Can you provide the same level of security to your Linux and Windows OS?*
- *Can you ensure that image packages & libraries are authorized (e.g., patched & up to date)?*
- *Can you make sure that your Kubernetes environment is properly secured?*
- *Can you be sure that access control, networking, and authentication are all in check?*
- *Can you be certain that your environment complies to CIS benchmarks?*
- *Do you know how to audit user activity on your host?*

Real-time Visibility and Control for Runtime Environment

The host environment is architected so that the shared OS runs the container engine and the engine runs the containers themselves. The OS is "aware" of the container engine, but it is "unaware" of which containers are running. The container engine, on the other hand, "knows" which containers are running, but it is not "aware" of container activity. The containers themselves, conversely, are not "aware" of each other, but they access OS resources directly. If you are running application firewalls or host-based intrusion prevention systems (HIPS) to monitor the OS, you'd be lacking both the visibility into container activity and the ability to monitor and control the same host containers' traffic.

Tracking containers' activity is mandatory for real-time attack detection and disruption. There are numerous ways in which containers can slip through the cracks via misconfigurations, or through explicit actions by attackers through namespace manipulations. By conducting on-going container behavioral analysis, controlling network traffic, and creating a container whitelisting policy, organizations can automatically detect unauthorized activity and block these attacks.

Essential Capabilities for Gaining Control and Visibility into Containers Runtime Environments

Image Assurance and Image-Container Drift Prevention (pre-deployment)
Encrypt images to protect data and IP
Enable re-scan of deployed images on hosts
Enable re-validation of image status (allowed/disallowed) before instantiation
Disable unneeded executables, network connections, ports, and file paths
Create cryptographic image fingerprinting for all layers
Detect and prevent containers running from unauthorized images (e.g., Bitcoin miners)
Prevent stale images from running
Prevent running images where their integrity changed
Prevent running processes that are not part of the original image (e.g., Bitcoin miners)

Workloads Firewall

Automatically discover & visualize the workload attack surface, relationships between namespace, deployments, pods, and network traffic, providing continuous updates based on actual container activity

Define which containers' inbound/outbound ports are accessible to/from which IPs/URLs

Label container groups as sensitive /inherit security group definitions from the orchestrator (e.g., PCI-sensitive) and apply firewall rules accordingly

Define container network connections based on a set of service-oriented firewall rules, regardless of where the container resides

Manually modify communication rules/policy based on actual activity, without impacting container performance and availability

Automatically alert on or block unauthorized communication flows (no container downtime)

Runtime Protection

Protect workloads running in host and Serverless/CaaS environments (e.g., AWS Fargate and Azure container instances)

Create zero-configuration container behavioral profiles

Detect and prevent container privilege escalation

Block unauthorized executables from running with no container downtime

Prevent kernel operation (e.g., do not allow "chown")

Centrally manage and enforce Seccomp profiles

Set & control container memory, CPU consumption & running process limits to prevent DOS attacks

Control user activity and enforce Segregation of Duties

Associate containers to source code for end-to-end vulnerability visibility and traceability

Detect any changes to containers (binaries, hash, system calls)

Detect and prevent unauthorized network connections such as open ports (on the same or cross hosts/ pods) based on automated policies

View running workloads grouped by hosts, pods, Kubernetes namespaces, image, uptime, status (stop/run)

Apply and enforce custom runtime policies per environment (e.g., apply black-listed 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

Enforce policy changes without impacting running applications

Provide real-time container activity monitoring

Ongoing policy enforcement in offline mode

Security considerations for runtime protection:

- *Do you know how to protect your runtime applications against attacks that are not based on known vulnerabilities, such as zero-days attacks based on host configuration errors, privileged user error, or insider threat?*
- *Do you have the means to detect an attack in the container's environment?*
- *Is every image deployed into production authorized? Can you ensure that the latest, authorized image is being instantiated? Do you know if any rogue containers are running?*
- *Can you ensure that the container image was not changed unintentionally or maliciously altered once on the host?*
- *Can you review only the vulnerabilities that are relevant to my running workloads?*
- *How can you priorities all the issues and assess the impact of each vulnerability in your environment?*
- *Can you ensure that containerized apps are granted the minimum necessary permissions and use secrets securely?*
- *Do you know if the app is the correct patched version?*
- *Do you have visibility of what's running inside a container? Do you know what executables are used for its intended purpose?*
- *Can you detect and alert on unauthorized network traffic?*
  - *Can you block unauthorized inbound/outbound communication from/to containers? For example, can you block outbound connection to a database container?*
  - *Can you block container (e.g., PCI-related) access to a specific IP address?*
- *Can you stop a container's unauthorized processes with no downtime to the running container itself?*
- *Can you prevent images, whose configuration has changed, from running?*
- *Can you perform instant impact analysis and list all the containers that have specific vulnerabilities?*
- *When considering a shift in app stacks, can you look for the impact or the use of specific components across all running containers?*
- *Can you immediately locate all containers that have a certain package, e.g., Python?*
- *Can you prevent executables that are not in the original image from running?*

Demonstrating Compliance for Cloud Workloads

In the event of a breach, demonstrating compliance is a useful means to avoid and reducing fines. Demonstrating compliance is best defined in the cloud workload environment as blocking vulnerable images from being used. These images are withheld and kept as compliance evidence, to prove that the remediation process was performed. Compliance evidence can also contain additional logged events, including the replacement of high-risk containers, blocked image executables, blocked user access, etc.

Regulatory Compliance Support

Audit & Compliance
Provide full user accountability and controlled super-user permissions
Real-time alerts on policy violations
Generate granular audit trails of all access activity, scan events and coverage, Docker commands, container activity, secrets activity, and system events
Built-in alerts and reports for key compliance mandates (PCI, GDPR, HIPAA, NIST SP 800-190)
CIS benchmark (Docker, K8s, and Linux) reports
Impact of CVEs and Vulnerability
Timeliness of scans
Ability to track changes in vulnerability status
Maintain vulnerability vs. remediation trends

Consider the following key compliance related evidence:

- *Can you measure the remediation trends based on KPIs (e.g., 'open' vs. 'remediated')?*
- *Do you log user and container activity?*
- *Do you alert on any behavior deviation in container activity?*
- *Do you keep a history of security configuration changes?*
- *Do you keep secrets rotation history?*

Enterprise-Grade Security

As organizations continue to adopt containers and serverless, and to a greater number of applications within those organizations, it is not simply about managing nodes. Many organizations run multiple applications, across disparate teams, running on-premise or multi-clouds and orchestrators, resulting in a growing requirement for scalable security that is easy to manage.

Integrations
Secrets store agnostic
Cloud platform agnostic
Orchestration agnostic (K8s, Red Hat OpenShift, Pivotal Cloud Foundry, Docker Swarm, AWS ECS, Mesos, Rancher)
Scanner plugins to multiple CI/CD build process (e.g., Jenkins, GoCD, TeamCity, Bamboo, GitLab, CircleCI, Azure DevOps
SIEM and analytics integrations (ArcSight, AWS CloudWatch, Datadog, Elasticsearch, Google SCC, Logentries, Loggly, Microsoft OMS, Splunk, Sumo Logic, Syslog, QRadar)
Full REST API support
Out-of-the-box integrations with multiple private and public registries (e.g., Docker Hub, Amazon ECR, Google GCR, CoreOS Quay, JFrog Artifactory, Azure ACR)
Integration with Jira
Notification feeds (Slack, PagerDuty)
LDAP/Active Directory/SAML authentication
SSO Authentication (SAML-based authentication, OpenID Connect)

Environment Support
On-premises & cloud installation available
Runtime protection for Windows & Linux containers
Runtime protection for on-demand Container-as-a-Service (CaaS) cloud environments
Private deployment of Cyber Center (feed) for air-gapped environments
Multi-tenancy management for MSP/multi-department deployments

APPENDIX

It is equally important to consider developers as part of this equation. DevOps methodologies are “left shifting” operations and security-related decisions into the hands of developers. Decisions of which operating system to use, image configuration (privileged/non-privileged) and image components used (e.g., unauthorized open source components) are now made by developers, who are not (nor should they be) security experts. Therefore, security best practices should also shift left via automated security controls embedded into the development process. This makes it easy for developers to run image scans, view scan results, and apply corrective measures – all according to the policy set by the security team. This will result in significantly reduced risks without hindering any development efforts.

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.

Contact

contact@aquasec.com
www.aquasec.com
@AquaSecTeam
AquaSecTeam

US HQ: 800 District Avenue, Suite 510, Burlington, MA 01803
Intl. HQ: 2 Ze'ev Jabotinsky Rd., Ramat Gan, Israel 52520