TAG SECURITY

CLOUD NATIVE SECURITY WHITEPAPER

VERSION 2

Contributors: Brandon Krieger, Cole Kennedy (TestifySec), Fatih Değirmenci (Ericsson Software Technology), Frederick Kautz , Joel Bork, Marina Moore (NYU), Mateusz Pruchniak, Pushkar Joglekar (VMware), Raja Faisal, Savitha Raghunathan (Red Hat), Sayantani Saha

Reviewers: @jonzeolla, @nyrahul (Accuknox), Ragashree Shekar, Steven Hadfield, Kapil Bareja, Malini Bhandaru, Mikko Ylinen, Jonah Kowall (@jkowall), Kuang Dahu,Ariel Shuper, Eric Li (AlibabaCloud), Tanner Randolph (Applied Systems), Mark Dalton Gray (Microsoft), Ciara Carey, Brandon Lum (TAG Security Chair), Emily Fox (TOC Liaison) and Justin Cormack (TOC liaison)

CLOUD NATIVE COMPUTING FOUNDATION

Version 1 of the paper was published on Nov 2020:

https://www.cncf.io/blog/2020/11/18/announcing-the-cloud-native-security-white-paper/ which is followed by version 1 contributors and reviewers

VERSION 1 (NOV 2020)

Contributors: Aradhna Chetal - TIAA, Brandon Lum - IBM , Chase Pettet - Mirantis (Chase.Pettet@mirantis.com), Emily Fox - US National Security Agency (NSA), Gadi Naor - Alcide, Harmeet Singh - IBM, Jeff Lombardo - Independent, Jeyappragash JJ - Tetrate IO, (VMware) Pushkar Joglekar - Visa, Rowan Baker & Andrew Martin - ControlPlane, Trishank Karthik Kuppusamy - Datadog, Vinay Venkataraghavan - Prisma Cloud (Palo Alto Networks), Wayne Haber - GitLab, Mark Bower, Alex Chircop - StorageOS

Reviewers: @justincappos, @lumjjb, @whaber, @craigbox, @anvega, @magnologan, Alok Raj - XenonStack (alok@xenonstack.com), @nyrahul(Accuknox), @ranio1, @lizrice, @justincormack

TABLE OF CONTENTS

NEW IN V2 INDEX

Executive Summary

Purpose

The technology industry has shifted towards patterns of development and deployment that are seen as “cloud native”. Simultaneously, the ecosystem of technologies, products, standards, and solutions is expanding, challenging decision makers to remain abreast of complex designs. The CISO role in particular, has the evolving responsibility of illuminating business value propositions in this dynamic arena. Meanwhile, cloud native patterns have also encouraged changes in consumption and adoption of modern workflows that encourage integrated security practices.

Problem Analysis

Security concerns within this landscape are complex because of the explicit focus on rapid development and deployment. Additionally, the reliance on static identifiers such as network IP addresses in a traditional perimeter based security model is impractical. This complexity requires a paradigm shift to protect applications by migrating from a purely perimeter based approach to one where security moves closer to dynamic workloads that are identified based on attributes and metadata (e.g. labels and tags). This approach identifies and secures workloads to align with the scale of cloud native applications while accommodating constant flux. These paradigm shifts require the adoption of increased automation of security controls in the application lifecycle and secure by design architectures (e.g., Zero Trust). The tradeoffs for a secure implementation continue to involve multiple stakeholders within an organization, and significantly impacts productivity of developers and operators pursuing business objectives. Cloud native applications still require development, distribution, deployment, and runtime operations, but the paradigm dictates new security mechanisms by which these objectives are efficiently achieved. Cloud native development can be modeled in distinct phases that constitute the application lifecycle: “Develop,” “Distribute,” “Deploy” and “Runtime.” Cloud native security contrasts with traditional security approaches in that there is a tremendous opportunity to ensure that security is injected throughout these distinct phases instead of book ending the lifecycle with separately managed security informed interventions. Continuous learning of these concepts, tools, and processes, are critical for long term adoption and application.

Lifecycle Phases

Develop

Cloud native tools are meant to introduce security early in the application lifecycle. Security testing needs to identify compliance violations and misconfigurations early to create short and actionable feedback cycles for continuous improvement. This approach enables security failures to follow familiar workflows raised for other issues in the pipeline (e.g., bug fixes or continuous integration failures), which already require resolution before moving software further in the pipeline. The modern security lifecycle for this model revolves around the development of code that adheres to recommended design patterns (e.g., 12-factor1) and ensures the integrity of the development environment.

Distribute

Software supply chain safety is especially critical in models that enable faster software iteration. Cloud native application lifecycles need to include methods for verifying not only the integrity of the workload itself, but also the process for workload creation and means of operation. This challenge is amplified by the necessary, practical, and consistent use of open-source software and third party runtime images, including layers of upstream dependencies. Artifacts (e.g., container images) present in the lifecycle pipeline require continual automated scanning and updates to ensure safety from vulnerabilities, malware, insecure coding practices, and other malfeasance. Upon completing these checks, it is important to cryptographically sign artifacts to ensure integrity and enforce non-repudiation. Immutable image binary and immutable URL of image is also worth noting for secure distribution.

Deploy

Security integrated throughout the development and distribution phases allows for the real-time and continuous validation of workload attributes e.g. signed artifacts are verified, container image and runtime security policies are ensured, host suitability and trustworthiness can be validated via binary authorization policy in staging and production environments. Deploy time checks provide the last chance to validate, correct, enforce these checks before the workload starts running to serve its intended business needs. Secure workload observability capabilities, deployed alongside the workload, allow for logs and available metrics to be monitored with a high level of trust, complementing integrated security.

Runtime

The cloud native runtime environment can itself be broken down into layers of interrelated components with distinct security concerns[1] e.g. hardware, host, operating system, network, storage, container image runtime, orchestration. Container runtime consists of different implementations for various levels of isolation boundaries , e.g. shared kernel, micro-vm sandbox and trusted execution environment sandbox. It is critical to choose a runtime that satisfies the expected security requirements. For example, for an untrusted workload running in a multi-tenant environment, a VM-based sandbox could be considered. For privacy sensitive financial data processing, a trusted execution environment (memory encrypted hardware per process or VM) like confidential containers might be worth considering. Within the typical cloud native runtime environment, applications are often composed of several independent and single purpose microservices which communicate with each other via service layer abstractions which the container orchestration layer makes possible. Best practices to secure this interrelated component architecture involves ensuring that only sanctioned processes operate within a container namespace, prevention, and alerting of unauthorized resource access attempts, and network traffic monitoring to detect hostile intruder activity. Service Mesh is another abstraction that provides consolidated and complementary functionality for orchestrated services without imposing changes on the workload software itself (e.g. logging of API traffic, transport encryption, observability tagging, authentication, and authorization).

Recommendations

Cloud native security seeks to ensure the same or higher conditions of diligence, integrity, trust, and threat prevention as traditional security models while integrating modern concepts of ephemerality, distribution, and immutability. In these fast changing environments, prone to rapid iteration, automation in line with the development pipeline is required for secure outcomes. Organizations should rapidly adopt these cloud native concepts to assist in the creation of value-driven security outcomes in their cloud-native journey. By integrating security as early as possible throughout the development lifecycle, or even earlier with interactive developer training, security organizations can enable preventative security rather than reactive security (see also “9 Box of Controls’’). It is highly recommended that organizations evaluate the security stack against the relevant attack frameworks[2] to achieve clarity about which threats a defensive stack covers. Additionally, organizations need to adopt approaches and methodologies that shift security left[3], enable DevOps, and are flexible enough to work with future technology advancements.

Conclusion

Cloud native security, when executed strategically across an organization, can provide high availability, assurance, resilience, and redundancy at scale to ensure customers and developers have secure access to required resources at the velocity they expect. Security itself remains an interdisciplinary field that cannot be isolated from the development lifecycle or be treated as a purely technical domain. Developers, operators, and security personnel must all partner, exchange, and collaborate to continue to move the field and industry forward. As with any technical innovation, people who embark upon this journey driven through their passion are the ones who genuinely make the community and cloud native security possible.

Introduction

This paper intends to provide organizations and their technical leadership with a clear understanding of cloud native security, its incorporation in their lifecycle processes, and considerations for determining the most appropriate application thereof. Cloud native security is a multi-objective and multi-constrained problem space spanning many areas of expertise and practice. Nearly all Day 1 and Day 2 operations overlap with the security domain, ranging from identity management to storage solutions. However, cloud native security covers much more than these areas; it is also a human problem space, incorporating individuals, teams, and organizations. It is the mechanisms, processes, and intent by which humans and systems interact with and make changes to cloud native applications and technology.

Target Audience

Our target audience is the Chief (Information) Security Officer (CISO), or Chief Technology Officer (CTO) of a private enterprise, government agency, or non-profit organization who wishes to deliver a secure cloud native technology ecosystem. Additional organizational stakeholders may include Project, Product, and Program managers, and Architects responsible for designing and implementing secure, cloud native products and services. Apart from this, anyone with a keen interest in cloud native security can benefit from referring to this document.

Cloud Native Goals

Cloud native architecture is a set of patterns and designs that provide organizations with an effective, sustainable seamless abstraction that works across different cloud instances to make the application stack cloud agnostic. The adoption

and innovation involving containers and microservices architectures have brought with it its fair share of challenges. Security leaders are tasked with protecting assets, both human[4] and non-human, by adopting practices to prevent, detect, and respond to cyber threats while meeting strict compliance requirements. A common historical narrative has been that security implementations impede the speed and agility of DevOps teams. Therefore, security leadership must implement tighter integration and bidirectional understanding, empowering DevOps teams to create shared cyber risk ownership.

Organizations must be urged to adopt secure cloud native patterns and architectures throughout the modern application development lifecycle. Most importantly, highlighting the synergies of the security architecture with the organization's security objectives such as Zero Trust, Secure Software Supply Chain, and DevSecOps across the cloud infrastructure , should be emphasized as a top priority.

The concepts described throughout this paper are not designed to favor one service, component, or product over another and can be applied regardless of service selection.

This document does not intend to provide general education on security concepts or cloud computing concepts. It also does not recommend specific technologies or tools; however, it may cite examples of technology or tools that address the topic discussed.

Beyond the recommendations in this document, specific data security handling practices related to data protection and privacy regulatory mandates, e.g. GDPR, PCI DSS, may need additional regulatory-specific consideration. We recommend that readers consult appropriate independent resources for guidance on any such technical controls and compliance risk matters.

Assumptions

All terms, concepts, and mechanisms used are defined as per the Cloud Native Security Lexicon and Cloud Native Glossary. This paper does not seek to change these definitions or expand upon it.

As cloud native adoption and modern software development methodologies continue to evolve, the technologies that comprise an effective cloud native stack will continue to shift over time. Representations of this shifting stack are included in the all encompassing CNCF landscape.

Cloud Native Layers

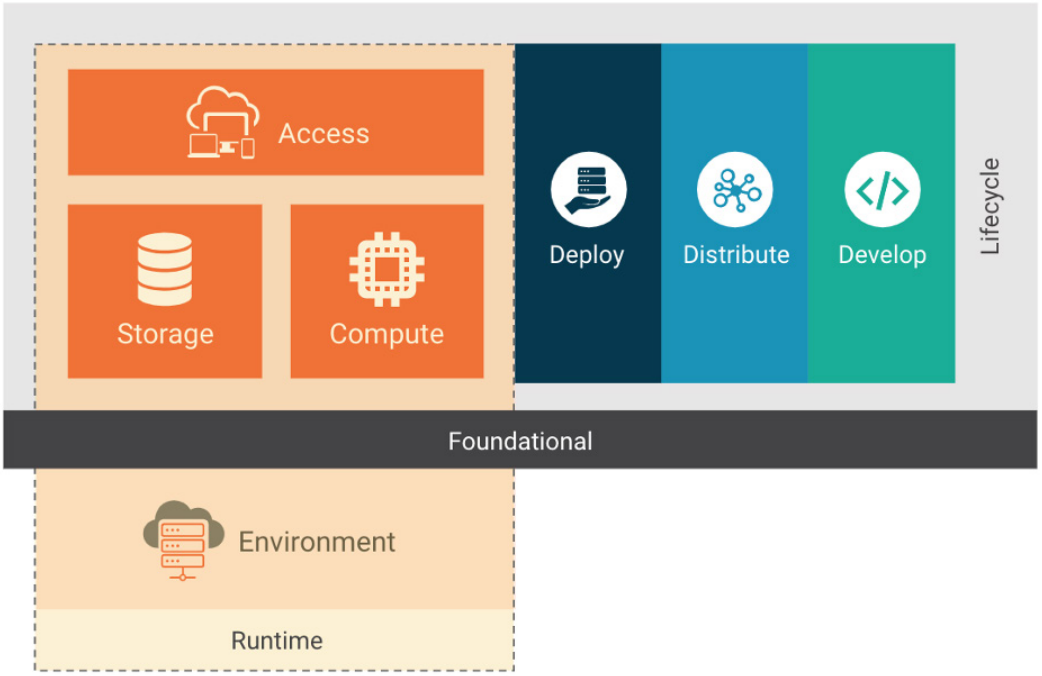

FIGURE 1

The cloud native stack is composed of the foundational layer, application lifecycle, and a runtime environment. The cloud native stack can be adopted using different deployment models: IaaS, PaaS, CaaS, and FaaS. Each deployment model provides additional abstractions that ease the management and operation of cloud native environments. As some of these models are considered well known and in use for years, we will focus on cloud native models only.

The Containers-as-a-Service (CaaS) model allows users to orchestrate and otherwise manage containers, applications, and clusters by leveraging a container-based virtualization platform, in conjunction with an application programming interface (API) or a web portal management interface. CaaS helps users construct scalable containerized applications with security policy embedded as code and run them on private cloud, on-premises data centers or in the public cloud. With platform-agnostic builds, microservices orchestration and deployments, it helps enterprises build and release software faster and allows portability between hybrid and multi-cloud environments, thus reducing infrastructure as well as operating costs. The CaaS model is cost saving, as it helps enterprises pay only for the CaaS resources they want and use. CaaS has containers as its fundamental resource, whereas for IaaS environments, virtual machines (VMs) and bare metal hosts are key.

Functions-as-a-Service (FaaS) is another cloud native deployment model, it is a type of cloud service that allows users to execute code in response to events without the complex infrastructure typically associated with building and launching microservices. Hosting a software application in the cloud usually requires provisioning and managing a virtual environment, managing the operating system and web components, etc. With FaaS, the physical hardware, virtual machine operating system, and web server software management are all handled automatically by the cloud service provider. In doing so, it allows users to focus on individual functions in the microservices code while paying only for the resources that are used while taking advantage of the resource elasticity that the cloud provides.

Lifecycle

Lifecycle in a cloud native context involves the technology, practices, and processes that enable resilient, manageable, and observable workloads to run natively in cloud environments. As depicted in Figure 1, the lifecycle is composed of four continuous phases; Develop, Distribute, Deploy, and Runtime. Each phase extends and amplifies the previous while permitting and supporting secure workload execution.

The next few sections provide a detailed analysis of the implications, tools, mechanisms, and best practices to integrate security throughout the application lifecycle.

Develop

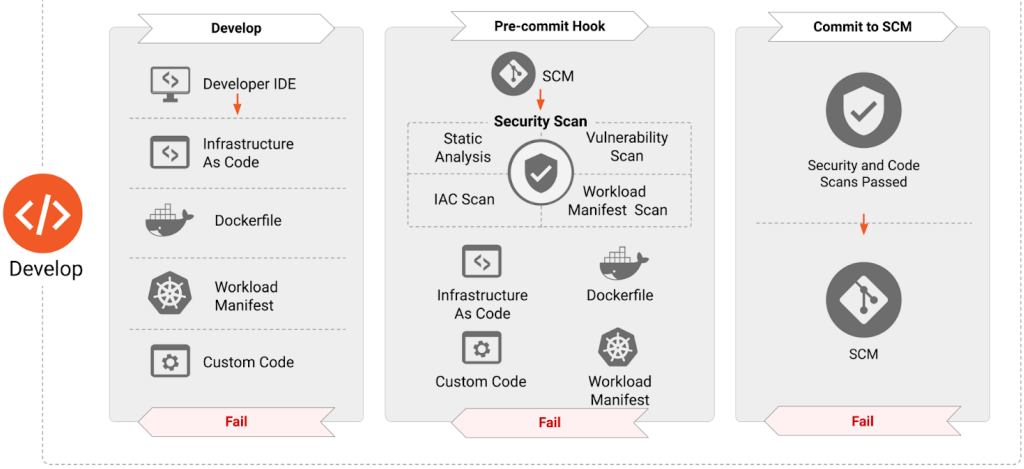

FIGURE 2

Security for cloud native applications needs to be applied throughout the entire lifecycle of an application. The "Develop" phase is the first in this cycle, resulting in the creation of the artifacts, such as Infrastructure as Code, application and container manifests, etc., that will be used to deploy and configure cloud native applications. Consequently, these artifacts have proven to be the source for numerous attack vectors that can be exploited in the runtime. The next few sections elaborate on the various security tools, processes, and checks that need to be instituted in this phase to dramatically reduce the attack surface of applications deployed in the runtime.

Security Checks in Development

Security hardening during the development phase forms a critical component in the deployment of applications. This means that security requirements must be introduced early in software development and treated in the same manner as any other design requirement. These requirements are typically based on business needs around risk and compliance and can be an outcome of a threat modeling exercise. Addressing these needs in the early phases prevents redoing work later in the lifecycle, which would otherwise slow down the DevOps pipeline, and increase overall costs[6]. DevOps teams must also leverage purpose-built tools to identify security misconfigurations and vulnerabilities before the deployment of these applications. Equally important is that these tools integrate seamlessly into existing and familiar tools leveraged by DevOps teams to complement agility with security and not impede it. For example, tools need to perform the scanning of Infrastructure as Code templates as well as application manifests within the developer IDE or when a pull request is made. They must provide rich and contextual security information which can be acted upon rapidly, easily, and early in the development pipeline by the owner of the code or application component. Adopting these steps ensures the absence of known vulnerabilities or high-risk configurations. Cloud native components should be API-driven, allowing for complex debugging tools to interact with the deployed primitive workloads that rely on the orchestrator.

Teams should deploy dedicated development, testing, and production environments to provide infrastructure and application developers with isolated environments to develop, test, and deploy systems and applications, container base images, VM golden images, and launch non-functional tests.

Development of Tests

Developers, operators, and security personnel should build tests for code and infrastructure that is business-critical, has a high threat-profile, is subject to frequent change, or has/is a historical source of bugs. Threat modeling can identify high-risk and high-impact code hotspots that provide a high return on investment (ROI) for developing tests. Tests may include deployment, operating system, infrastructure and database hardening, application testing (static and dynamic source code testing, such as fuzzing, container configuration), integration or system testing (acceptance of application and infrastructure components and their interaction), and smoke testing (post-deployment checks against a live system). Test authors should have access to comprehensive development and test environments that enable them to perform rapid test development while reducing continuous integration (CI) feedback loops. System test suites should be made available to run locally for authors as well as within a shared testing environment.

Code Review

Minor changes to a workload, or the infrastructure where the workload has been deployed, may have far-reaching security consequences. To mitigate the risk of unintended consequences, teams are encouraged to use the “four eyes” principle when conducting code review prior to changes being merged into the codebase (e.g. implementing a pull request in git workflow).

Distribute

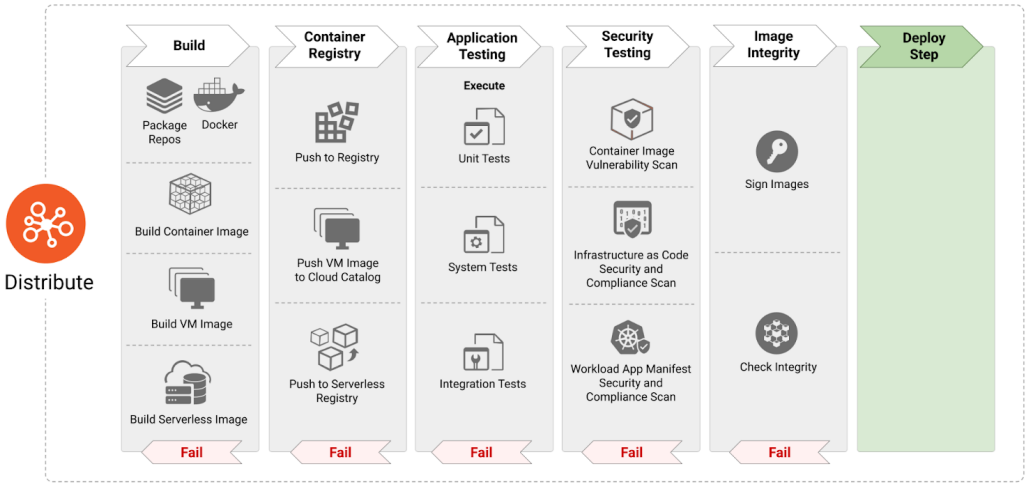

FIGURE 3

The “Distribute” phase is responsible for consuming image definitions and specifications to build the next stage of artifacts such as container images, VM images and others. In modern CI/CD pipelines, the “Distribute” phase consists of systematic application testing to identify bugs and faults in the software. However, the adoption of open source and reusable packages can result in the incorporation of vulnerabilities and malware into container images. It is therefore imperative to incorporate security-focused steps such as scanning the images for such threat vectors as well as for validating the integrity of the images to protect against tampering. Furthermore, organizations may wish to encrypt software artifacts if confidentiality is desired or needed.

Should software artifacts become untrusted due to a compromise or other incident, teams should revoke signing keys to ensure repudiation.

Build Pipeline

Continuous Integration (CI) servers should be isolated and restricted to projects of a similar security classification or sensitivity. Infrastructure builds which require elevated privileges should run on separate dedicated CI servers. Build policies should be enforced in the CI pipeline and by the orchestrator’s admission controllers.

Supply chain tools can gather and sign build pipeline metadata. Later stages can then verify the signatures to validate that the prerequisite pipeline stages have run.

The reader should ensure that the CI and Continuous Delivery (CD) infrastructure is as secure as possible. For example, security updates should be prioritized to be installed, and cryptographic keys should be protected from exfiltration via the use of HSM or Credential Managers.

Image Scanning

Scanning container images is one the most common ways of securing container applications throughout the lifecycle. It is vital to do the scanning in the CI pipeline before deploying the image to production. Additionally, continuous scanning of images of running containers is equally important to identify newly found vulnerabilities. Incorporating this capability ensures that developers, operators, and security professionals have detailed information on all known vulnerabilities and details such as the severity, the Common Vulnerability Scoring System (CVSS) score, and availability of mitigation/fixes. Coupling vulnerability scans of container images with pipeline compliance rules ensures that only sufficiently patched applications are deployed to production, reducing the potential attack surface. Scanning of container images also helps to identify the presence of malware in open source software packages or base image layers incorporated from open source image repositories. While container image scanning can provide teams with evidence of vulnerabilities or malware, it does not remediate vulnerabilities

or prevent malware. Organizations need to ensure that container scanning findings are acted upon, and that organizational compliance rules are enforced.

Image hardening

Container images must include security hardening that takes into consideration the threats to be mitigated while allowing some just-in-time configurations at the runtime phase to facilitate integration with the broader ecosystem.

Regarding the security assurance objectives, the following questions should be evaluated:

- Restricting the execution environment to a specific user?
- Limiting the access to resources?
- Kernel level restrictions on process execution?

Container Application Manifest Scanning

Application manifests describe the configurations required for the deployment of containerized applications. It is vital to scan application manifests in the CI/CD pipeline to identify configurations that could potentially result in an insecure deployment posture.

Container Application Manifest Hardening

As for container images, container application manifest hardening can be thought of and be implemented at build- as well as runtime.

With respect to security assurance objectives, the main question to answer is: What minimal constraints should the runtime execution ecosystem comply with?

Testing

Cloud native applications should be subjected to the same suite and standard of quality testing as traditional applications. These include the concepts of clean code, adherence to the Test Pyramid, application security scanning and linting through static application security testing (SAST), dependency analysis and scanning, dynamic application security testing (DAST), application instrumentation, and full infrastructure with tests available to developers in local workflows. Automated test results should map back to requirements for dual attestation (developer and tool) for real-time security assurance to security and compliance teams.

Once a security misconfiguration has been identified (e.g. an incorrect firewall or routing rule), if root cause analysis determines that it has a reasonable chance of recurring, the developers should write an automated test to prevent regression of the defect. At the test failure, teams will receive feedback to correct the bug, and with the next merge, the test will pass (assuming it was corrected). Doing so defends against regression due to future changes to that code.

Unit testing of infrastructure is a preventative control, and targets entities and inputs defined in Infrastructure as Code (IaC) configuration. Security testing of built infrastructure is a detective control and combines assurance, historical regressions, and unexpected configuration detection (firewall rules open to the world, loosely-privileged Identity & Access Management (IAM) policies, unauthenticated endpoints, etc.)

Hardening of infrastructure and workloads should be supported by comprehensive test suites, which allows for incremental hardening as the system matures. Tests to verify hardening has occurred should run during the build, but also at deployment to evaluate any changes or regressions that may have occurred throughout the lifecycle.

Static Analysis and Security Testing

Static analysis of IaC, application manifests, and software code provide linting, identification of misconfigurations, and vulnerabilities of the specific component and will not include context of the surrounding components in analysis. It is important to note that individual analysis is important, but should not be the only form of analysis. IaC code should be subject to the similar pipeline policy controls as are application workloads.

IaC is an increasingly popular way for organizations to deploy cloud and container infrastructure. Insecure configurations in IaC templates will naturally lead to security gaps in the deployed infrastructure. These templates should therefore be scanned for characteristics that compromise security, before deploying the application and infrastructure artifacts. Key misconfigurations to keep an eye out for include:

- Vulnerabilities contained within images specified in the application manifests
- Settings that do not respect the principle of the least privilege, such as containers that can escalate privileges or overly lax firewall rules
- Identification of the security contexts and system calls, which can compromise a system
- Resource limit settings

Dynamic Analysis

Dynamic analysis of deployed infrastructure may include detecting Role-based Access Control (RBAC) and IAM configuration drift, validating the expected network attack surface, and ensuring that a SOC can detect unusual behavior in dedicated test environments to configure alerting for production. Dynamic analysis is considered to be a part of testing, however, it is expected to occur in a non-production runtime environment.

Security Tests

Automated security testing of applications and infrastructure should be an integral focus within security teams. Test suites should be continuously updated to replicate threats in-line with the organizational threat model and can be reused for security regression testing as the system evolves. Automated security tests increase security and release velocity by removing manual security gates, such as validation and manual control implementation at a single checkpoint, which is time-consuming and inadequate. Automated security testing also demonstrates control efficacy on demand by explicitly attempting to carry out the threats, thus improving the system's security and adherence to any embedded compliance requirements in real-time.

Artifacts & Images

Registry Staging

Due to the use of open source components that are often pulled from public sources, organizations should create several stages of registries in their pipelines. Only authorized developers should be able to pull base images from public registries and store them in an internal registry for wide consumption within the organization. It is also advised to have separate private registries for keeping development artifacts per team or group, and finally a staging or pre-production registry for images ready for production. This enables tighter control over the provenance and security of open source components, while enabling different types of testing for stages in the CI/CD chain.

For any registry used, access control through a dedicated authentication and permission model must be implemented. Use mutually authenticated TLS for all registry connections (among other interactions within the architecture).

Signing, Trust, and Integrity

Digital signing of image content at build time and validation of the signed data before use protects that image data from tampering between build and runtime, thus ensuring the integrity and provenance of an artifact. Confirmation starts with a process to indicate that an artifact was vetted and approved. The trust confirmation also includes verifying that the artifact has a valid signature. In the simplest case, each artifact can be signed by one signer to indicate a single testing and validation process that the artifact has gone through. However, the software supply chain is more complex in most cases, and creating a single artifact relies on multiple validation steps, thus, depending on a conglomerate of entities' trust. Examples of this are:

- Container image signing - the process of signing a container image manifest
- Configuration file signing - signing of a config file, i.e. application config files
- Package signing - Signing of a package of artifacts, like application packages

For generic software artifacts such as libraries or OCI artifacts, signing these artifacts indicates their provenance of approved usage by the organization. Verification of these artifacts is equally crucial in ensuring that only the authorized artifacts are allowed. It is strongly recommended that repositories require mutual authentication to introduce changes to images in registries or to commit code to repositories.

Encryption

Container Image Encryption encrypts a container image so that its contents are confidential. The container image contents are encrypted to ensure that they remain confidential for promotion from build time through runtime. In the event of a compromised distribution, the image's registry contents remain secret, which can help for use cases such as protecting trade secrets or other confidential material.

Another common use of Container Image Encryption is to enforce container image authorization. When image encryption is coupled with key management and runtime environment attestation and/or authorization and credential distribution, it is possible to require that a container image can only run on particular platforms. Container image authorization is useful for compliance use cases such as geo-fencing or export control and digital rights media management.

Deploy

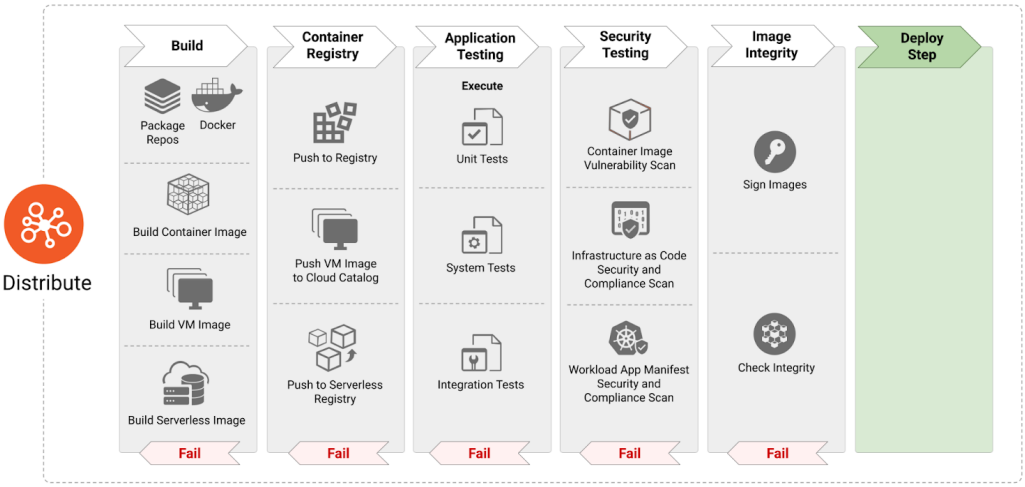

FIGURE 4

The "Deploy" phase is responsible for incorporating a sequence of 'pre-flight' checks to ensure that the applications that will be deployed in the runtime environment conform and comply with organization-wide security and compliance policies.

Pre-Flight Deployment Checks

Before deploying a container image, organizations should verify the existence, applicability, and current state of:

- Image signature and integrity
- Image runtime policies (e.g absence of malware or critical vulnerabilities)
- Container runtime policies (e.g absence of excessive privileges)
- Host vulnerability and compliance controls
- Workload, application, and network security policies

Observability & Metrics

Instituting observability and metrics into cloud native architectures delivers security insights, so appropriate stakeholders can resolve and mitigate anomalies appearing in reporting; tools in this area can help collect and visualize this information. Through the use of behavioral and heuristic analysis, teams can detect

and escalate outliers, suspicious events, and unexplained calls to appropriate stakeholders. Use of advanced machine learning and statistical modeling techniques that fall under artificial intelligence is encouraged to assist in behavioral and heuristic analysis development.

Incident Response & Mitigation

An application should provide logs regarding authentication, authorization, actions, and failures. The developer should include this capability as part of planning and design phases. These elements provide a trail of evidence to follow when an investigation takes place and a root cause needs to be established.

Forensics capabilities are an integral part of any incident response and mitigation activity. They provide evidence to determine the root cause of an incident and provide feedback for any mitigation to be put in place. The short-lived nature of the container environment requires a more agile tool set to capture and analyze any evidence. Integrating forensics capabilities into an incident response plan and procedures will provide the means to acquire and process evidence, decrease the time to determine root cause, and minimize exposure to a compromise.

Runtime Environment

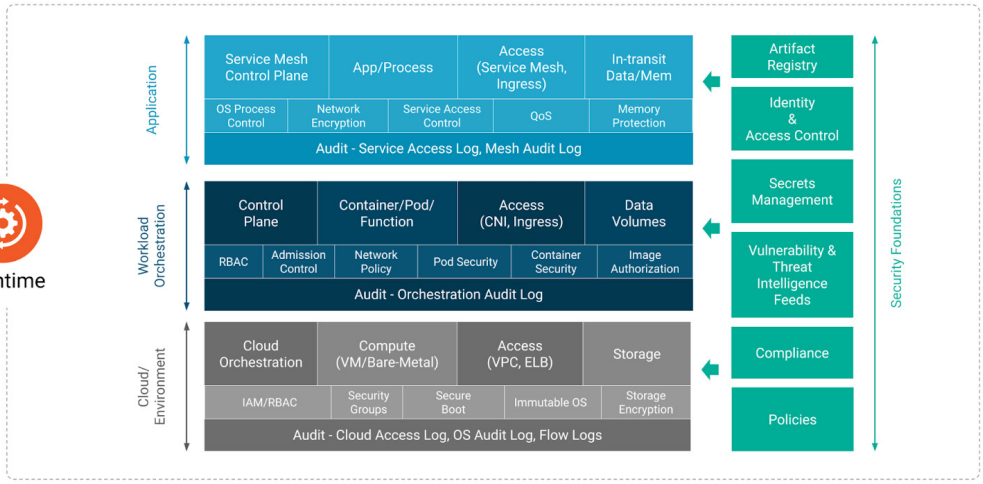

FIGURE 5

The Runtime phase comprises three critical areas: compute, access, and storage. While the runtime environment is dependent on the successful completion of the develop, distribute, and deploy phases, the security of the runtime is dependent on the efficacy of the security practices of the prior phases. The following paragraphs detail the security requirements and implications for each of these critical components.

Compute

Cloud native computing is highly complex and continually evolving. Without core components to make compute utilization occur, organizations cannot ensure workloads are secure.

Considering that containers provide software based virtualization for multi-tenant applications on a shared host, it is important to use a container specific operating system, which is a read-only OS with other services disabled. This helps in reducing the attack surface. This also provides isolation and resource confinement that enables developers to run isolated applications on a shared host kernel. To allow defense in depth, it's recommended to not allow disparate data sensitive workloads to be run on the same OS kernel.

In order for security to span all layers of container platforms and services, a hardware root of trust based in a Trusted Platform Module (TPM) or virtual TPM can be used. The chain of trust rooted in hardware can be extended to the OS kernel and its components to enable cryptographic verification of trusted boot, system images, container runtimes, and container images, and so on.

Secure enclaves (also known as Trusted Execution Environments or TEE) are at the core of confidential computing. Secure Enclaves are sets of security-related instruction codes built into new CPUs. They protect data in use, because the enclave is decrypted on the fly only within the CPU, and then only for code and data running within the enclave itself. TEE-based confidential computing ensures data security, integrity, and confidentiality. Encrypted data and code in the TEE is unavailable to other applications, the BIOS, operating systems, kernels, administrators, cloud vendors, and hardware components except CPUs. TEE-based confidential computing collaborates with sandboxed containers to isolate malicious applications and protect sensitive data.

Operating systems provide basic system components like crypto libraries used for remote connections and kernel functions that are used for process initiation, management etc. These can have vulnerabilities and, because they provide underlying compute baseline for the containers, they can impact all the containers and apps that run on these hosts. At the same time, improperly configured containers can impact the host kernel security and hence all the services running in containers running on that host.

Orchestration

Any orchestrator has several components that are separated into control and data planes. Occasionally, there is a need to have a higher-level multi-deployment management plane responsible for maintaining state across several control planes that co-exist independently of each other.

Any orchestration system has numerous threats that impact the overall security of the deployment and continued security at runtime. Malicious access to an orchestrator's API, unauthorized access and changes to the key-value store, orchestrator dashboard to control clusters, intercept control plane traffic, API misuse, intercepting application traffic, and so on are all potential threat areas. It is important to use best practices and configuration hardening for any orchestrator to prevent exposure to these threats[7]. It is also essential to monitor and detect any changes to the initial configurations made in runtime to ensure the continued security posture of the cluster. Other security best practices such as minimizing administrative access to the control plane, segregation of duties and principle of the least privilege should be enforced.

Security Policies

It is essential to consider the security features and various configuration options of your orchestrator to control the security privileges the container runtime can use to spawn containers. The use of higher level policy and governance constructs may enforce those security guardrails.

Resource Requests and Limits

Single misbehaving workload intentionally (e.g., fork bomb attack or cryptocurrency mining) or unintentionally (e.g., reading a large file in memory without input validation, horizontal autoscaling) can cause exhaustion of node and cluster level resources. Applying different object level resource requests and limits via cgroups helps prevent such a scenario.

Audit Log Analysis

Audit Log analysis is one of the most established methods to identify and correlate system compromise, abuse, or misconfiguration. Continued automation of audit log analysis and correlation is of paramount importance to security teams, as cloud native architectures are capable of generating more granular audit configuration and filtering than traditional legacy systems for workloads. Additionally, the interoperability of cloud native logs allows for advanced filtering to prevent overloads in downstream processing. What is critical here, as with traditional log analysis, is the generation of actionable audit events that correlate/contextualize data from logs into "information" that can drive decision trees/incident response.

Non-compliant violations are detected based on a pre-configured set of rules that filter violations of the organization's policies. To have the ability to audit actions of entities using the cluster, it is vital to enable API auditing that filters for a specific set of API Groups or verbs that are of interest to a security team or cluster administrators. Immediate forwarding of logs to a location inaccessible via cluster-level

credentials also defeats an attacker's attempt to cover their tracks by disabling logs or deleting their activity logs. These systems processing alerts should be periodically tuned for false positives to avoid alert flooding, fatigue, and false negatives after security incidents that were not detected by the system.

Control Plane Authentication and Certificate Root of Trust

The orchestrator administrators should configure all orchestrator control plane components to communicate via mutual authentication and certificate validation with a periodically rotated certificate in addition to existing control plane hardening. The Issuing Certificate Authority (CA) can be a default orchestrator CA or an external CA. Using an external CA may involve a non-trivial amount of work in maintaining the Certificate Authority Infrastructure, so this option should be selected with caution. Particular attention should be given by the administrators to protect the CA's private key. For more information on extending or establishing trust, refer to the Identity and Access Management section.

Secrets Encryption

It is possible to manage secrets in a container orchestration or deployment environment through use of an external secrets manager or natively using the orchestrator's secrets. When using a native secret store, it is crucial to be aware that several protection methods are available:

Encryption with an external KMS

- Leveraging a KMS is a secure way to protect secrets in the orchestrator secret store, where key encryption in an external KMS encrypts the Data Encryption Key (DEK) that encrypts the secrets stored at rest. This method does have an option to cache DEKs in memory to reduce the dependency on the availability of the external KMS and faster decryption of secrets during workload creation time.

Encryption fully managed by the orchestrator

- This methodology encrypts the secrets stored in the orchestrator, but the encryption key is also managed by the orchestrator (i.e. a config file of the orchestrator)

No encryption

- For example, with some orchestrators, secrets are base64 encoded and stored in clear-text in the key-value store by default

Using an external secrets manager can limit the risks of using unencrypted secrets and ease the complexity of key management. Typically tools are provided as controllers, drivers or operators that can inject secrets at runtime and handle their rotations transparently.

Runtime

The runtime environment of a container needs to be monitored and secured from a process, file, and network perspective. Only sanctioned capabilities and system calls (e.g. seccomp filters), should be allowed to execute or be invoked in a container by the host operating system. In some cases, the usage of sandboxing container runtimes is worth consideration to enable more strict host isolation. Changes to critical mount points and files should be monitored and prevented. Configuration must prevent changes to binaries, certificates, and remote access configurations. The configuration must also prevent ingress and egress network access for containers to only what is required to operate. Additionally, network traffic to malicious domains should be detected and denied.

Conversely, full workloads or parts of the workload that process privacy-sensitive data in memory during runtime can be executed in trusted execution environments. This enables confidential compute and protects workload data from external threats.

Microservices and Eliminating Implicit Trust

The perimeter for containerized applications deployed as microservices is the microservice itself. Therefore, it is necessary to define policies that restrict communication only between sanctioned microservice pairs. The inclusion of zero trust in the microservice architecture reduces the blast radius by preventing lateral movement if a microservice is compromised. Operators should ensure that they are using capabilities such as network policies to ensure that east-west network communication within the container deployment is limited to only that which is authorized for access. There is some initial work done to provide strategies for microservices security through NIST SP 800-204 and may serve as a guide for implementing secure microservice architectures.

Image Trust & Content Protection

Utilization of a policy agent to enforce or control authorized, signed container images allows organizations to provide assurance of the image provenance for operational workloads. Further, inclusion of encrypted containers allows for the protection of sensitive sources, methods, or data that exist within the container.

Service Mesh

A service mesh provides connectivity between the services that adds additional capabilities like traffic control, service discovery, load balancing, resilience,

observability, security, and so on. A service mesh allows microservices to offload these capabilities from application-level libraries and allows developers to focus on differentiating business logic. To effectively ensure secure communications between services in cloud native environments, organizations should implement a service mesh to eliminate implicit trust within and across workloads, achieved through data-in-motion encryption. Utilization of a service mesh also resolves identity issues where traditional static identities, like IP addresses, no longer cleanly map to workloads. Service mesh provides not only service level isolation and security but also network-level resiliency capabilities such as retry, timeout, and implementing various circuit-breaker capabilities. Streaming platforms can benefit from a service mesh for added security by using workload level authorization to set access rules for topics or brokers.

It is important to note that implementation of a service mesh can help reduce the attack surface of a cloud native deployment, and provide a key framework for building zero trust application networks.

Runtime Detection

Monitoring deployed workloads should provide teams with validation that the true operational state is the expected state. Organizations cannot forgo periodic security scanning and monitoring within their environments without turning their workloads into an unsupervised playground for attackers. Utilization of components that detect, track, aggregate, and report system calls and network traffic from a container should be leveraged to look for unexpected or malicious behavior.

While regression testing and security tests can help prevent known, expected issues from moving to production environments, they cannot stop everything. Workloads should be dynamically scanned to detect malicious or unexpected behavior for which no known occurrence yet exists. Events such as an extended sleep command that executes data exfiltration from a data store after the workload has been running for X amount of days are not expected in the majority of environments and therefore are not included in security tests. The aspect that workloads can have time or event delayed trojan horses is only detectable by comparing to baseline expected behavior, often discovered during thorough activity and scan monitoring.

Further, workloads will become vulnerable at the time of, or after they are deployed. Organizations should continuously scan their environments to detect which workloads are now vulnerable. Understanding the make-up or software bill of materials for each workload can help organizations quickly identify where vulnerabilities lie. Additional information about those vulnerabilities, such as exploit maturity, and vulnerable path in use are critical to determining the actual risk to workloads and can help organizations prioritize updates to at-risk applications.

Functions

Serverless functions are susceptible to various attacks and therefore need to be appropriately protected. Processes must execute only functions explicitly defined in an allow list. Additionally, functions should not be allowed to change critical file system mount points.

The functions must have restrictions that only allow access to sanctioned services, either through networking restrictions or least privilege in permission models. Additionally, the egress network connection must be monitored by administrators to detect and, where possible, prevent access to C&C (command and control) and other malicious network domains. Ingress network inspection must also be considered to detect and remove malicious payloads and commands that can be used in exfiltration. For instance, SQL injection attacks can be detected using inspection.

Serverless functions have a number of threats, and controls available for tenants are limited. Broken authentication and insecure API integrations with dependent services are some of these issues. Ensuring all serverless functions are run in tenant-based resource or performance isolation for similar data classifications may assist in resolving this, however, they can impact the performance due to limitations in the address space available to the isolation environment.

Bootstrapping

Trust needs to be bootstrapped in the compute nodes to ensure that workloads and configurations are run on the correct nodes. Bootstrapping ensures that the compute is in the correct physical and logical location and provided with the ability to authenticate itself. These steps are usually part of the cloud provider's provisioning. However, methods are available to verify trust, with limited reliance on a third party.

Storage

Cloud Native Storage covers a broad set of technologies that are bucketed into presented storage and accessed storage. Presented storage is storage made available to workloads such as volumes and includes block stores, file systems and shared file systems. Access storage is storage that is accessed via an application API, and includes object stores, key value stores, and databases.

Storage systems contain a data access interface that defines how applications or workloads store or consume data that is persisted by the storage system or service. This interface can be protected by access controls, authentication, authorization, and potentially encryption in transit.

Storage systems also contain a control plane / management interface which is typically an API protected by authentication and TLS, although finer grained access may be available. In general, the control interface is only accessed via a service account by an orchestrator or service broker.

Storage Stack

Any storage solution is composed of multiple layers of functionality that define how data is stored, retrieved, protected and interacts with an application, orchestrator and/or operating system. Each of these layers has the potential to influence and impact the security of the storage system. A common example may be a file system that persists files or blocks to an object store. It is equally important to protect every layer in the topology, and not just the top layer where data is accessed.

Orchestration

Most orchestrated systems will implement a variety of abstraction and virtualization layers that may include file systems (such as bind mounts), volume managers, and the application of permissions at a user or group level based on orchestrator policies. As with many components of containerization and microservice architectures, protecting volumes and storage will always rely on the protections in place from other in-cluster capabilities. If a user can escalate their privileges within the orchestrator or container runtime to root they can wreak havoc within the environment including to the underlying storage systems. The implementation of zero trust, least privilege, and access control and enforcement are linchpins in successfully securing storage in cloud native architectures.

System Topology & Data Protection

Understanding a system's storage topology is key to secure both the data access path to the storage system and the intra-node communication in distributed topologies.

Common topologies include centralized models where all compute nodes access a central storage service, distributed models that distribute the function over many nodes, and hyperconverged models where application and storage workloads are combined on the same nodes. The selection of specific, layered security mechanisms to protect data in storage and in transit between storage locations is driven on the topology in use by the system.

A key function of any storage system is to provide protection of the data that is being persisted in the system or service. This protection is implemented first through availability of the data to authorized users and should exist as a transparent layer in the system. This can include technologies such as parity or mirroring,

erasure coding or replicas. Protection is next implemented for integrity, in which storage systems add hashing and checksums to blocks, objects or files. The hashes are primarily used to detect and recover from corrupted data, but can also add a layer of protection against the tampering of data.

Caching

Caching layers, often fully fledged separate systems, are implemented to improve the performance of storage systems, especially file systems, objects, and databases. The appropriate access controls and security policies need to be applied to the caching layer, as the cache will be fronting the access to the actual storage back-end.

Data Services

Storage systems typically implement several data services which complement the core storage function by providing additional functionality that may be implemented at different layers of the stack and may include replication and snapshots (point-in-time copies of data). These services are regularly used to move copies of data to remote locations, and it is important to ensure that the same access controls and security policies are applied to the data at the remote location.

Physical or Non-Volatile Layer

Cloud native storage security is not restricted to virtual cloud native architectures as cloud native capabilities can be deployed on-premises, and even virtual offerings have a physical presence. It is essential to remember that storage systems will ultimately persist data on some form of physical storage medium which is generally non-volatile. Modern physical storage such as SSDs often support security functions such as self encryption, as per the OPAL standards, and rapid/secure erasure functions. Secure erasure is important when devices that contain data need to leave a secure physical location (e.g. to be returned to a vendor after developing a fault).

Storage Encryption

Storage systems can provide methods to ensure confidentiality of data through data encryption. Data encryption can be implemented for data in transit or data at rest, and when implemented the storage system can ensure that encryption is done independent of the application. Encryption functionality is often dependent on integration with a key management system.

Encryption can have an impact on performance as it implies compute overhead, but acceleration options are available on many systems which can reduce the overhead. When selecting the kind of encryption for data, consider the data

path, size, and frequency of access as well regulations, compliance or additional security protections that may require more secure encryption algorithms to be used. Additionally, teams should not neglect the use of caches when considering encryption requirements for their architecture.

Encryption services can be implemented for data in transit (protecting data as it traverses the network) and for data at rest (protecting data on disk). The encryption may be implemented in the storage client or storage server and granularity of the encryption will vary by system (e.g. per volume, per group or global keys). In many systems, data in transit is protected with TLS (which has the added benefit of providing an authentication layer via certificates[8]. Older protocols (such as iscsi) may be harder to secure in transit (although more complex solutions such as IPsec or encrypted VPNs[9] can be used). Data at rest is generally protected using standard symmetric encryption algorithms such as AES, and may be deployed with specific modes of encryption such as XTS for block devices.

Public Cloud storage that includes Block, Shared File System and Object Storage, may support data encryption with CMK and BYOK natively.

Persistent Volume Protection

Protecting access to volumes is critical to ensure only authorized containers and workloads may leverage volumes provided. It is imperative to define trust boundaries for namespaces to cordon access to volumes. Leverage existing or create new security policies that prevent groups of containers from accessing volume mounts on worker nodes and ensure only appropriate worker nodes have access to volumes. It is especially critical as privileged containers can gain access to a mounted volume in a different namespace, so additional precautions are needed.

Specifying the UID or GID of the volume still permits access by container in the same namespace and will not provide data protection. Network file system version 3 (NFSv3) assumes the client has already performed authentication and authorization and does not perform validation. It is critical to consider where authentication and authorization occur and whether validation of that action exists when implementing protections.

Artifact Registries

Registries should accommodate technologies to sign and verify OCI artifacts. It is also important to ensure that the caching and distribution tools also provide the capability to sign, encrypt and provide checksums to ensure that the caching layer can detect tampering or attempts to poison the dataset.

The CNCF Storage Whitepaper provides additional background on the concepts, terminology, usage patterns and technology classes of cloud native storage.

Access

Identity and Access Management

A comprehensive identity and access management (IAM) solution for cloud native architectures requires service identity at a minimum. Organizations maintaining or operating on-premise or hybrid clouds need user and device identity management. For applications and workloads distributed across multi-cloud environments, identity federation is critical to a successful implementation.

Applications and workloads should be explicitly authorized to communicate with each other using mutual authentication. Due to the ephemeral nature of cloud computing, key rotation and lifespan need to be frequent and short to maintain the demands of high-velocity capabilities and control and limit the blast radius in case of credential compromise.

The utilization of identity management services from cloud providers is dependent on industry-specific use cases. Users, independent of the cloud-provider, should generate and manage credentials and keys for sensitive workloads such as health or finance information.

For the client and server to bi-directionally verify identity via cryptography, all workloads must leverage mutual/two-way transport authentication.

Authentication and authorization must be determined independently (decision point) and enforced (enforcement point) within and across the environment. Ideally, secure operation for all workloads should be confirmed in real-time, verifying updated access control and file permissions where possible as caching may permit unauthorized access (if access was revoked and never validated). Authorization for workloads are granted based on attributes and roles/permissions for which they have been assigned. It is strongly recommended organizations use both Attribute-Based Access Control (ABAC) and Role-Based Access Control (RBAC) to provide granular authorization enforcement in all environments and throughout their workload lifecycle. Such a posture can enable defense-in-depth, where all workloads are able to accept, to consume, and to forward the identity of the end user for contextual or dynamic authorization. This can be achieved through the use of identity documents and tokens. Not enforcing this limits an organization's ability to truly perform least privilege access control on system-to-system and service-to-service calls.

It is critical to note, application or service identity is also essential in the context of microservices, where the identities of apps are primarily subject to be spoofed and impersonated by a malicious service. Utilization of a strong identity framework and service mesh can help overcome these issues.

All human and non-human cluster and workload operators must be authenticat-

ed and all their actions must be evaluated against access control policies that will evaluate the context, purpose, and output of each request. To simplify the authentication process, identity federation can be configured to allow usage of enterprise capabilities such as multi-factor authentication. Authorization must then be enforced with access control mechanisms mentioned in this section.

Credential Management

A credential management solution gives organizations the power to efficiently manage both hardware and software-based credentials that access digital and physical resources. Deploying a secure credential management system is a critical step in the process of securing your systems and information.

Hardware Security Modules (HSM)

Whenever possible, the reader should use technologies such as HSMs to physically protect cryptographic secrets with an Encryption Key that does not leave the HSM. If this is not possible, software-based credential managers should be used.

Credential Management Cycle

Cryptographic secrets should be generated securely within either an HSM or a software-based secrets management system.

Secrets, whenever possible, should have a short expiration period or time to live after which they become useless. Secret management should be highly available and have high ease of generation, as these characteristics are prerequisites for the short-lived secrets. While not recommended, if organizations are using long-lived secrets, appropriate processes and guidance should be established for periodic rotation or revocation, especially in case of accidental disclosure of a secret. All secrets must be distributed in transit through secure communication channels and should be protected commensurate with the level of access or data they are protecting.

In any case, secrets should be injected at runtime within the workloads through non-persistent mechanisms that are immune to leaks via logs, audit, or system dumps (i.e. in-memory shared volumes instead of environment variables).

Availability

Denial of Service (DoS) & Distributed Denial of Service (DDoS)

A denial-of-service attack (DoS attack) in the context of cloud native applications

is a class of cyber-attacks. The perpetrator seeks to temporarily or indefinitely make the cloud native application unavailable to its intended users (human or automated). The perpetrator may do this via disrupting critical cloud native application components (such as microservices), disrupting the orchestration layer responsible for keeping the microservices running, or disrupting health monitoring systems responsible for scaling the application. A denial of service is typically accomplished by flooding critical microservices or resources with superfluous requests to overload systems and prevent some or all legitimate requests from being fulfilled.

A distributed denial-of-service attack (DDoS attack) typically involves a high volume of incoming traffic flooding the cloud native application services or the upstream networks to which they depend. Typically, the attack is mounted from many sources. Volumetric attacks are mitigated by detecting and deflecting the attacks before they reach the cloud native application.

Security Assurance

Security is fundamentally a risk management process that seeks to identify and address risks posed to a system. The iterative and perpetual hardening of systems will mitigate, reduce, or transfer risk depending on the component's or organization's risk profiles and tolerances. The predisposing concepts of hardening, while legacy at their core, can still be applied to a security conscious team by evaluating components and their makeup against minimal, yet flexible, functionality. For instance, as teams determine an updated base image, considerations for additional ports, permissions, and packages added with an update should be reviewed and either accepted, altered, or restricted.

In contrast, compliance standards form principles of controls to ascertain or create requirements definitions by which systems are assessed against. The outcomes of the assessment are binary (pass or fail) but may contain Type 1 (false positive) or Type 2 (false negative) errors and should be evaluated as the result of tests from a CI/CD pipeline, akin to the results of any testing in a pipeline. Thus, compliance and security assurance are complementary processes but are not interchangeable. A compliant system is not guaranteed to be secure, nor a secure system is guaranteed to be compliant.

Threat Modeling

For organizations adopting cloud native, a primary mechanism for identifying risks, controls, and mitigations is to perform threat modeling. While there are many threat modeling techniques, they share several core characteristics. All start with building a scoped representation of a system's architecture. This begins with

identifying all important processes, data stores, and security boundaries. Once boundaries have been established and the relevant elements of the system are partitioned within them, the next step is to model how these elements interact, with special attention paid to any interactions that cross security boundaries.

The below guidance is an enhancement of the four-step OWASP threat modeling recommended for cloud native capabilities.

End-to-end architecture

A clear understanding of the organization's or individual's cloud native architecture should result in data impact guidance and classifications. This helps teams organize data distribution within the architecture, as well as the additional protection mechanisms for it later on. Cloud native diagrams and documentation should not only include the core components of the overall system design. This includes the location of the source code, the storage mechanism in use, and any additional aspects of the software development cycle. These are all areas that must be considered when initiating threat modeling for cloud native.

Threat Identification

When considering threats specific to an organization's cloud native capabilities, it is recommended to leverage a mature, well-used model of threats such as STRIDE or OCTAVE. Common threats organizations may wish to consider for their cloud native architectures includes, but is not limited to:

- **Spoofing** a cluster admin by stealing the authentication credentials via a social engineering attack
- **Tampering** of an API server config file or certificate could result in failed API server restart or mutual TLS authentication failures
- **Repudiation** of actions of an attacker because of disabled or misconfigured API auditing could result in a lack of evidence of a potential attack
- **Information disclosure** is possible if an attacker compromises a running workload and can exfiltrate data to an external entity
- **Denial of Service (DoS)** resulting from a workload that does not have resource limits applied therefore consumes the entire node level CPU and memory, worker node is then lost
- **Elevation of privilege** could happen if a workload is running with unrestricted or higher privileges or by modifying the security context of a workload or a container

Threat actors to consider for cloud native security are consistent with existing threat modeling practices:

- **Malicious insider** - An actor with malicious intent and with authorization to perform actions within the modeled system.
- **Uninformed insider** - An actor with authorization to perform actions within the modeled system (assuming anyone can be duped).
- **Malicious outsider** - An actor with malicious intent and outside the system, able to launch attacks via the internet, supply chain, physical perimeter etc. without explicit authorization to perform actions within the modeled system.

There are other actors that may interact with the modeled system (e.g. uninformed outsiders) and they can be included for completeness. It's likely that controls for their actions will be a subset of those for the primary actors listed above.

As with any cloud native process, it is important to iterate and provide feedback. In the context of threat modeling, this means re-evaluating if the existing measures, mechanisms, and matrices accurately reflect the operational state given the continual changes to the architecture.

Threat Intelligence

Cloud native applications are a collection of multiple dynamic components compromised from first-party and third-party code and tools, which means threat intelligence must be applied for network activity and cloud native application components. Cyber threat intelligence is information about threats and threat actors that helps mitigate harmful events. Threat intelligence in cloud native systems would make use of indicators observed on a network or host such as IP addresses, domain names, URLs, and file hashes which can be used to assist in the identification of threats. Behavioral indicators, such as threat actor tactics, techniques, and procedures can also be used to identify threat actor activity in cloud native components. The MITRE ATT&CK framework can be leveraged as a starting point for establishing and validating threat activity.

Threat Matrix for Containers (New in v2)

The threat matrix for containers by ATT&CK is an excellent starting point in evaluating and modeling threats for your systems. ATT&CK's Threat matrix for containers is focused mainly on adversarial behaviors they exhibit during a successful attack on a system.

The threat matrix of ATT&CK has rows & columns, where the rows consist of the techniques and the columns consist of the tactics. Understanding what might be the end goal for the attacker can help us build better security and defend against

it as developers and platform operators. Let's look at various techniques explained in Threat Matrix of Containers through that lens:

- **Initial Access:** This is the very first step for an attacker to successfully exploit a container environment. Public facing applications have vulnerabilities that can be exploited by the attacker, which might lead to the adversary getting host access. So as a developer, it is important to implement mutual authentication for externally facing services and limit sharing host resources such as mounting host file systems where possible.
- **Execution & Persistence:** The adversary who succeeded in gaining initial access into the environment will continue running malicious code, to maintain their control across system reboots. This can typically happen through an attacker owned malicious image that is deployed just like any other benign workload to evade detection. So as a platform operator, it is important to limit registries that are accessible from the cluster, implementing a secure image promotion process and auditing image pulls in the clusters so that anomalous events such as pulling from an unknown registry can be alerted on.
- **Privilege Escalation:** This is when the adversary will attempt to get the root or administrative privileges. Adversaries may break out of a containerized environment to gain access to the underlying host. Therefore as a developer it is useful to follow the least privilege principle when setting permissions for your workload which makes it harder for an attacker to break out of the runtime isolation.
- **Defense Evasion:** Once the adversary has established control in the environment, it will try to actively evade the system's defenses. Therefore, as platform operator auditing for shell commands executed on the host or container exec calls will allow detection of such methods.
- **Credential Access:** If the adversary has come this far, it uses a brute force approach to gain access to container & container orchestration accounts. Therefore as a platform operator, giving developers access to short lived credentials will limit the value of a compromised credential as it would be useless once expired.
- **Discovery:** The adversary tries to understand the container environment and discover the available resources such as containers or components deployed on a cluster and try to understand its assigned permissions. Auditing GET calls of an API gateway/server and commands executed by unknown users on hosts is a great feature to provide as a platform operator for detecting attacks in the discovery phase.
- **Impact:** In this stage the adversary executes its objective, which may involve performing Denial of Service (DoS) attacks to degrade or block the availability of targeted resources. This has the goal of co-opted systems to solve

resource intensive problems which may impact system or hosted service availability. Therefore, as platform operators it is important to have well documented incident response playbooks and by default applying soft and hard resource limits to workloads sharing host resources.

Static metadata such as IPs, Domains, and Hash values will change across different environments, but it is difficult to change the mind of an adversary, this is the core motivation behind building MITRE ATT&CK Threat Matrix for Containers. Several other mitigations for the techniques and tactics described in the Threat matrix are explained in greater depth through all four phases of cloud native app lifecycle in this paper.

Incident Response

An organization with existing incident response and triaging workflow, special attention should be paid to how that can be applied to cloud native workloads which may not always conform with some underlying assumptions. These include node isolation (new workload instances could run on a different server), networking (e.g. IP addresses are assigned dynamically) and immutability (e.g. runtime changes to container are not persisted across restarts). Hence, it's important to revisit these assumptions and reapply or update the incident response playbook as needed. Observability and forensic tools need to understand cloud native specific constructs such as pods and containers so that the state of a compromised system can be maintained or recreated. Evidence mishandling can sometimes be unintentional in intent-based orchestrators, which are built to treat workloads as "cattle, not pets". As a side note, building an incident response and triaging strategy from the ground up, although possible, is out of scope for this document.

Use case: Ransomware (New in v2)

Identifying, modeling and implementing mitigations for threats can be a daunting task. To make it a bit more approachable, let's look at a concrete example of the Ransomware threat in a cloud native context.

Ransomware is malware that employs encryption to hold a victim's information at ransom. A user or organization's critical data is encrypted so that they cannot access files, databases, or applications. A ransom is then demanded to regain access to the encrypted data. Ransomware is often designed to spread across a network and target database and file servers, and can quickly paralyze an entire organization. It is a growing threat, generating billions of dollars in payments to cybercriminals and inflicting significant damage and expenses for businesses and government organizations.

Even though numerous types of ransomware exist, a few actions remain consistent for these attacks. We see these ransomware attacks using malware that iden-

tifies and disables or removes multiple processes on the endpoint that operators could have used to detect the execution or even help with recovery post-infection as a first step post-compromise. A ransomware attack typically looks like system event logs being disabled and removed along with volume shadow copies, recovery partitions, and any system backups being deleted before the encryption phase occurs.

What then occurs is what is known as the encryption phase, which is where the malware typically is directed towards specific file system directories. Ransomware strains will then look for certain file types and enumerate the system, looking for any remote file shares or other endpoints sharing resources. The ransomware will then perform its encryption functions and deliver a ransom note with follow-on means for communication and payment.

RansomCloud refers to a particular type of ransomware attack that targets data in the cloud. That data is increasingly valuable as many businesses move their operations into public and private clouds.

Preventing Ransomware attacks

Ransomware prevention begins with following best practices and developing mature security capabilities. Foundational capabilities such as establishing secure baselines, patching vulnerable software, and mature configuration management practices are crucial to prevention. Observability platforms and well-tested disaster recovery capabilities are vital to minimizing the impact and recovery time.

Having regular security assessments, vulnerability scanning, and penetration testing as part of your ongoing strategy is essential to prevent a ransomware attack proactively. Understanding your current security posture e.g. controls in place to limit a successful social engineering attack and mitigating critical unpatched vulnerabilities that could be compromised will be crucial to avoiding the worst impacts of a ransomware attack.

When malware reaches the encryption stage, very little can be done to prevent your devices from being impacted. Averting ransomware events from occurring, needs malware detection in earlier phases of the MITRE ATT&CK framework. To accomplish this, relying solely on signature-based detection capabilities and indicator-based threat intelligence is not a complete solution. Enterprises will need to perform defense in depth strategies that include micro-segmentation and behavioral analysis for internal and cloud network segments and any external related traffic.

Developing a secure software factory and deployment pipeline will significantly reduce the risk of ransomware by reducing the attack surface by controlling the number of deployed vulnerabilities and mandatory code/configuration management. A software factory is ideal for implementing code scanning, image scanning, code review, and validating the supply chain provenance.

Risk is further reduced by treating configuration changes as code that must traverse through the secure software factory, including scanning and code reviews. Configuration management can be tracked through the pipeline and audited by an external observability platform.

Anomalies, including administrative actions rarely performed, must be identified and tracked. Expected anomalies should be tracked and labeled for auditing purposes. The observability platform should tag unexpected anomalies for additional review. Rule engines and AI/ML may automate some anomaly detection for scalability, but automated detection should not yet replace humans who can reason about more complicated scenarios.

Deployments must follow the Principle of Least Privileges. This principle is critical to reducing the blast radius of a compromised deployment. Operators should isolate databases from the workloads, with minimal privileges allowed. Best practices include using views, prepared statements, disabling updates/deletes when not needed. Backups should be maintained and regularly tested. For more advanced protection, enable ledger capabilities of the underlying storage and database, such as object versioning.

Protecting data encryption keys is also vital. A ransomed cryptographic key can be just as devastating as ransomed raw data. Production systems with sensitive data should store their keys in KMS or HSM. Cloud environments offer high-quality KMS services that are FIPS 140-2 certified.

Finally, it is essential to limit the paths of communication between systems. Operators can do this in a few ways. If you are practicing Zero Trust, you can ensure only systems with approved and valid cryptographic identities are capable of communicating through encrypted channels such as Mutual TLS. For applications unaware of cryptographic identities, it is essential to establish encrypted tunnel network policies and provision next-generation firewalls to protect against malicious attacks.

Ideally, steps taken to prevent a ransomware attack work as expected, in keeping an organization away from being a victim of a successful attack. However, these measures take time to implement though, and while they should make an organization harder to compromise and more able to recover from attack, it is not full proof and there are never any guarantees.

Ransomware Incident Response

As per NIST Incident Response Guide, these steps are involved in managing a ransomware incident:

Preparation

Establish an incident response plan that has gone through several rigorous table-top exercises with your team to understand how your organization will respond to a potential Ransomware attack. This will include whom to contact if your organization is a victim of a Cyberattack. If applicable, this will include emergency contact numbers in your organization, carrier, breach coach, DFIR company, and MSP/MSSP. You will need to activate the team as quickly as possible to start the next stage in the life cycle.

Detection & Analysis

In this stage, you will want to quickly and efficiently detect the suspicious/malicious to contain and eradicate it. In this process, you will need to do your best to maintain digital forensics evidence as much as possible. This way, you can investigate the incident to find artifacts that will provide crucial information on how the threat actor compromised your IT infrastructure/cloud environment. You will also want to know if they moved laterally throughout the environment and what data the threat actor gained access to.

Through this stage, you will also if you do not already have an Endpoint Detection and Response solution in place, you will want to deploy one as quickly as possible. This will give you visibility to your endpoints to detect, quarantine, or kill any suspicious or malicious activity. This way, you can start working on containing the active threats.

Containment, Eradicate & Recovery

Working on containment is essential to eradicate the active threats so that your team can start working on recovering from the Cyberattack.

Containment could be disconnecting the endpoint(s) that have been identified as compromised from the network without shutting these systems off. Remember, we want to preserve digital forensics evidence.

Next is to eradicate the active threat and confirm the Threat Actor is no longer in the environment. This is essential because Threat Actors are known to hold organizations hostage and request greater demands when they realize they still have control of the environment.

As soon as you feel comfortable that you have contained the active threat, it appears to be eradicated from your IT/Cloud environment. You will now start work-

will also want to know if they moved laterally throughout the environment and what data the threat actor gained access to.

Through this stage, you will also if you do not already have an Endpoint Detection and Response solution in place, you will want to deploy one as quickly as possible. This will give you visibility to your endpoints to detect, quarantine, or kill any suspicious or malicious activity. This way, you can start working on containing the active threats.

Containment, Eradicate & Recovery

Working on containment is essential to eradicate the active threats so that your team can start working on recovering from the Cyberattack.

Containment could be disconnecting the endpoint(s) that have been identified as compromised from the network without shutting these systems off. Remember, we want to preserve digital forensics evidence.

Next is to eradicate the active threat and confirm the Threat Actor is no longer in the environment. This is essential because Threat Actors are known to hold organizations hostage and request greater demands when they realize they still have control of the environment.

As soon as you feel comfortable that you have contained the active threat, it appears to be eradicated from your IT/Cloud environment. You will now start working on recovery. This next point is crucial to responding to a Ransomware attack and could save you millions of dollars.

It is essential to have a backup program that has been tested and protects your backups. It is recommended that you have an off-site backup solution for all your critical systems and data. These backups should be scanned for malware and stored in a secure location. These backups will be essential to restoring business continuity and not having to negotiate payment with a Threat Actor for your data, potentially.

Post-Incident Retrospective

Here is where a team debriefs after a Ransomware attack to understand the successes and challenges that occurred through the incident. This is a great time to evaluate the Incident Response Plan, Administrative and Technical Controls, Disaster Recovery Plans, Back-ups, Endpoints, Change Management, External and Internal Communication plans in case of breach.

Having this new insight into having gone through a Ransomware attack will change how a business thinks about its operations and day-to-day activities. This should not be short-lived, and this new understanding needs to be implemented into existing business practice and security program.

Security Principles

Secure Defaults (New in v2)

A strong security system in its default state is possible, cost effective and transparent. Building or transitioning towards such a system involves following these guidelines in cloud native context:

1. Make security a design requirement

2. Applying secure configuration has the best user experience

3. Selecting insecure configuration is a conscious decision

4. Transition from insecure to secure state is possible

5. Secure defaults are inherited

6. Exception lists have first class support

7. Secure defaults protect against pervasive vulnerability exploits

8. Security limitations of a system are explainable

For more details on these guidelines, please refer to this page: Secure Defaults: Cloud Native 8

Least Privilege

Least privilege is just as important, or perhaps the most important, aspect of cloud native architectures, and must be considered at all parts of the stack where an authentication or authorization decision is made. Traditionally Least Privilege has been thought of at the account layer whether that account is a human or a service.

In cloud native, least privilege must be applied at every layer of the stack. It should also be considered when evaluating the specific tooling responsible for fulfilling each layer's execution. Organizations may find, as they explore various products and capabilities, that many containers have privileged-by-default deployments or required root privileges to operate. As a result, additional measures may need to be taken to isolate those elevated privileges from the rest of the workload. Organizations should consider all areas to employ isolation and least privilege in their workloads and deployments; from cgroups and system calls in the runtime environment to artifact management and rootless builds.

To consistently reduce the potential attack surface and corresponding blast radius, organizations need to implement the principle of least privilege at every level of their architecture. This not only applies to the individuals performing various functions within their roles, but also to the services and workloads

executing in a given environment. Rootless services and containers are vital to ensuring that if an attacker does get into an organization's environment, they cannot easily traverse between the container they gain access to and the underlying host or containers on other hosts.

Mandatory Access Control (MAC) implementations (e.g. SELinux and AppArmor) can limit the privileges beyond those set to the container or namespace. Additionally, they provide container isolation at the host level to prevent container breakout or pivoting from one container to another, to escalate privileges beyond those permitted by the access control in place.

Roles and Responsibilities

When moving to cloud native architectures and deployments, organizations should expect to see adjustments in legacy security roles and responsibilities and create new security roles specific to the cloud. With the rapid onset of modern development methodologies and better alignment of IT activities with business needs, security must be adaptive, commensurately applied with actual risk, and transparent. It is unreasonable to expect developers and operations to become security experts. Security practitioners need to partner with developers, operations, and other project life elements to make security and compliance enforcement fully integrated with process modernization efforts and development lifecycles. Doing so means findings are reported in real-time through the tools in use by developers for habitual resolution, akin to how build failures are resolved at notice.

The blurred lines that often occur in DevOps environments should not replace clear separation of duties (SoD) when it comes to managing security in cloud native environments. While developers will be a lot more involved in implementing and executing security measures, they do not set policy, need not gain visibility into areas that aren't required for their role, etc. - this separation should be implemented between roles and across product and application teams in accordance with the organization's risk tolerance and business practices. It is understood this becomes difficult with smaller organizations when individuals perform many duties to keep the business thriving. Nevertheless, implementing a distinct role to permission alignment can assist in enforcing SoD as the organization continues to grow and cognitively forces a mental switch in the activities being performed by the individual. Ultimately, allowing for reorganization roles to be reassigned to new individuals without increasing scope of access with the new assignment.

Organizations will need to reevaluate their asset risks as products and services migrate to the cloud. With ownership and management changes of the technology in use and its deployment stack, executives should expect significant risk posture changes. Shared responsibility between providers and teams

will require changes to thresholds in risk acceptance, transference, and new mechanisms for mitigation.

Supply Chain Security (New in v2)

The security of a system is only as good as the supply chain that it relies on. Defending the supply chain requires securing how software and hardware are designed, implemented, distributed, configured, stored, and verified.

An effective supply chain strategy relies on mature policies and procedures to mitigate risks associated with first and third-party software creators, integrators, and distributors. Each party must communicate relevant information accurately and effectively. A supply chain policy should contain diverse providers with controls designed to limit harm from a compromised supply chain. It provides provenance of systems, software, and configurations with the ability to trace the origin and validate the integrity of both the artifact and the chain that produced it.

The software supply chain consists of source code, second- or third-party code, build pipelines, artifacts, and deployments. Each of these stages must be performed by an authenticated trusted party such that it can be verified cryptographically and automated where possible. All trusted entities should have limited authorization scope to reduce the impact of a compromise.

Software Bill of Materials (SBOMs) are a critical first step towards discovering what software components you have so then you may correlate them with known vulnerabilities. SBOMs should be generated using standardized formats such as but not limited to SPDX, Cyclone DX, SWID, at build-time from the source code and should link to the SBOM of imported libraries and tools.The source code and component metadata included in the SBOM may also be used by developers and operators to identify tampering of the software supply chain. The CI/CD system should sign both application and container images with signatures reproduced in the SBOM. Operators may use post-build SBOM generators from binaries or images to validate the accuracy of the build-time SBOM. End to end attestations may be used to validate the processes used by software creators and suppliers. These attestations should be added to every step in the software supply chain.

In some cases, it may be challenging to identify the software that is impacted by CVEs and implement fixes since SBOMs can contain thousands of dependencies and do not identify if they have a CVE (out of scope of the SBOM). In these types of situations, generating delta reports during the build process and storing them with the corresponding SBOMs can help organizations to identify the actual vulnerable software (together with its version) much faster and with less effort or potential error. An additional benefit of delta reports is that they can help to accurately identify non-vulnerable but impacted (dependent) software also.

A secure CI/CD system should generate SBOMs and attestations. Attestations should include the CI step's process, environment, materials, and products. Evidence should be cryptographically verified when possible. The software producer should use trusted documents such as signed meta-data documents and signed payloads to verify the authenticity and integrity of the built environment.

It's equally important to keep track of dependencies of the CI/CD's supply chain in this process. Suppliers should provide proof of assessments and reviews of their components and dependencies. Suppliers should provide timely notifications of vulnerabilities, whether they are affected by those vulnerabilities or breaches. Upcoming standards such as VEX will provide a common framework for exchanging information on vulnerabilities.

Operators and security teams should store all of the above information in a queryable inventory system to discover vulnerable or non-compliant systems quickly.

A mature and automated SBOM, CVE, and VEX program may provide relevant information to other security and compliance controls. For example, the infrastructure may automatically report non-complying systems to an observability platform or deny providing a necessary cryptographic workload identity, effectively quarantining it from compliant systems in Zero-Trust environments.

The CNCF has produced the Software Supply Chain Best Practices White Paper to assist you with designing a secure supply chain process. This whitepaper provides more details about securing the software supply chain and discusses relevant CNCF projects that developers and operators can use to secure various stages of the supply chain.

GitOps (New in v2)

GitOps is code-based infrastructure and operational procedure that rely on Git as a source control system. It is an evolution of Infrastructure as Code (IaC) and a DevOps best practice that leverages Git as the single source of truth, and centralized control management for creating, updating, and deleting IT system architecture. GitOps allows separating deployments from development and use full advantage of the immutable declarative infrastructure. Every element of the environment can be deployed as often as needed with the same result, instances are redeployed instead of restoring from multiple unique configurations and versions.

Traditional processes mostly rely on human operational knowledge, expertise, and actions performed manually but in case of GitOps all changes are made as interaction with Git repository. Therefore, the Git repository and GitOps process become crucial to secure and should be secure by design. Immutability of infrastructure protects from making changes from outside the main deployment process and easier to detect and reverse environment changes based on the declarative state in the Git repository.

Usage of IaC and GitOps increase the overall security of the infrastructure itself by limiting manual operations, providing an audit of all changes, a declarative single source of truth, policy enforcement via the necessary controls and gates on processes to ensure that security requirements are met. Using GitOps tools and technologies, organizations can mitigate different vectors of attacks, i.e. by reducing the number of people and machines that have access to the target system.

GitOps processes are responsible to deliver changes to the production environment and if that process is compromised, then the adversary may open infrastructure backdoors or may introduce harmful software to production environments. Some noteworthy guidelines to follow based on least privilege principle and separation of duties are:

- Restrict access to repository and branches
- Never store unencrypted credentials or secrets in the Git repository and block sensitive data being pushed to Git
- Enforce strong identity with GPG Signed Commits, to give accountability and traceability
- Require linear history and maintain a commit history by disallowing force pushes
- Enforce branching policy, especially protect the main branch and require code review before merging
- Monitor for vulnerabilities, and keep Git and GitOps tools up to date
- Rotate SSH keys and Personal Access Tokens, block unauthorized access to Git repositories
- Utilize a dedicated non-user technical account for access where credentials are frequently rotated and short-lived
- Limit users who can elevate permissions to remove security features to cover their tracks via deletion of audit trails and silencing of alerts

In summary, GitOps can enable elimination of vulnerabilities long before any code is deployed to production through quality and security policy gates when needed.

Zero Trust Architecture

Zero trust architectures mitigate the threats of lateral movement within a network through fine- grained segmentation, micro-perimeters, and removing implicit trust to data, assets, applications, and services (DAAS) with verification and enforcement policies. Most common implementations of zero-trust architecture rely on cryptographic concepts to create zero trust. This is primarily based on the ability to have specific key material secured in hardware or tokens and managed in a way

where they can be securely communicated to a platform.

The foundational building block that zero trust architecture usually consists of several aspects:

- Each entity can create proof of whom the identity is
- Entities can independently authenticate other identities (i.e. Public Key Infrastructure)
- Communications between entities remain confidential and untampered

The zero trust framework creates the zero trust building blocks by leveraging a strong root of trust: the ability to tie a tamper-resistant trust to an entity or process is the foundational building block. It then requires attestations: the ability to attest, validate, and prove the identity of an entity. For the example of container services, how do I check that this container is who it claims to be. This needs to be verified with the orchestrator, but to trust the orchestrator, we need to ensure it is running untampered, which can only be ensured if we are running a trusted OS, BIOS, etc. Attestations are usually a chain as well.

Zero trust also requires secure communication between entities. While network segmentation provides value to zero trust architectures and should be considered, is not an end all solution to zero trust implementation. Orchestrator network policies as well as use of a service mesh are all components of a comprehensive zero trust solution. More information on zero trust concepts is available widely online.

Security Stack (New in v2)

Implementation of these security assurances across the four lifecycle phases are explored in depth in Cloud native security map that can be found here: https://cnsmap.netlify.app. The one side effect of security implemented through these tools across the stack is that they help in compliance needs of a cloud native environment.

Compliance

Designing a system with the appropriate set of security controls that address regulatory, and compliance guidance makes cloud native resources more secure. Doing so may also make certification by relevant regulatory bodies and auditors easier, particularly if the system design and planning is done to allow automated compliance to various regulatory bodies through a plugin model. While compliance often requires utilization of security benchmarks (e.g. NIST Application Container Security Guide, Center for Internet Security (CIS), NIST Security Strategies for Microservices-based Application Systems, and OpenSCAP) it is important to note that utilization of machine-readable compliance control frameworks and languages are recommended.

Adoption and implementation of these benchmarks enable teams to test for a hardened baseline and deploy secure-by-default workloads. However, they cannot consider data flows and custom usage of the platforms under testing. Security practitioners should implement them as a guide rather than a checklist.

Regulatory Audits

Many financial, health, government, and other entities need to comply with a specific set of requirements to protect the system. The users trust the systems to keep their interactions private and secure. Every organization should evaluate which regulatory standards apply to them (e.g., PCI-DSS, HIPAA, FedRAMP, GDPR, etc.). They should then determine how specific requirements apply to their cloud native systems and how they will implement those standards' real-world implementation. This evidence-gathering mechanism supporting adherence to specific standards should be automated with non-repudiation guarantees whenever possible.

Personas and Use Cases

The focus is on security, protection, detection, and auto-response wherever possible. It is not necessarily development tooling alone, but security tooling that integrates transparently into the development process to enforce security policies where fast feedback and most immediate actions to remediate can occur. For specific information on cloud native security use cases, refer to the TAG-Security's use cases listing.

Industries

Enterprise

Core areas of concern for Enterprise to adopt a cloud native model are maintaining the current process and procedures while meeting the business objective. Keeping the interoperability, data loss or leakage, and security risk exposure at a minimum when new standards and practices are introduced throughout the organization.

Microbusiness

Core areas of concern for Small businesses to adopt a cloud native model are the ability to focus on short term goals and foster innovation to meet intense competition. The lack of resources, budget, technology depth, and best practice hinders their ability to adapt to cloud native solutions. Small business requires repeatable patterns and a small IT footprint to solve the challenges.

Finance

Core areas of concern for financial and insurance industries that are essential to successful cloud native adoption are legal framework compliance, data localization requirements, immutable audit logging, unauthorized disclosure of information, and fraud detection.. Fraud, being somewhat unique to the sector, can have a direct impact on fund availability, making the integrity of financial transactions of paramount importance.

Healthcare

Core areas of concern for healthcare industries that are essential to successful cloud native adoption are unauthorized disclosure of information, timeliness, and availability of records, and accuracy of records. Due to the nature and practices of the healthcare industry, the availability of records and their associated content is the basis by which medical decisions are made.

Academia and Education

Core areas of concern for educational institutions for successful cloud native adoption can be dependent upon the intended end user. Institutions catering to minors may have additional legal requirements to protect the confidentiality of minors, and thereby making access control critical. Beyond this, institutions should focus on the availability of educational content to end users.

Public Sector

Core areas of concern for Public Sector organizations that are essential to successful cloud native are security, data sovereignty, compliance, and vendor lock-in. The barriers emerge from agencies placing regulations to protect the public interest. In the public sector, it is essential to maintain harmony and trust between public and government entities. Additionally, timeliness of deployments and features may also be a strong consideration. The adoption of cloud native, along with modern methodologies, can increase organizational velocity, which is critical to many areas of the public sector.

Use case: Securing Financial Institutions under EU regulations (New in v2)

Cloud Native architectures in public and private clouds have become the standard solution for modern IT for fast innovation, delivering more value to their customers with dramatically less effort. This is a big challenge, especially for regulated sectors such as finance due to the complexity of their legacy systems, and compliance challenges including concerns raised by regulatory institutions. Let's look at three main authorities in case of European Union:

- **EBA** - The "European Banking Authority" is an independent authority that works to ensure effective and consistent prudential regulation and supervision across the EU banking sector.
- **EIOPA** - The "European Insurance and Occupational Pensions Authority" is a European Union financial regulatory institution.
- **ESMA** - The "European Securities and Markets Authority" works in the field of securities legislation and regulation to improve the functioning of financial markets in Europe, strengthening investor protection and cooperation between national competent authorities.

Each of the above authorities pay particular attention to:

- Flexible and secure multi-cloud strategy,
- Solid foundations for portability and interoperability,
- Right to access and right to audit,
- Security of data,
- Exit strategy,
- Risk assessment and concentration risk.

Risk assessment should be conducted considering the expected benefits, security, costs, business continuity, legal, compliance, operational and concentration risks. The risk assessment when performed lends itself to a properly designed solution to mitigate risks and strengthen operational resilience. Additionally, financial institutions need to develop comprehensive exit plans that are documented and sufficiently tested. These plans should be updated as needed, including in case of any service changes.

Cloud Native architecture helps to meet the above regulations and requirements by microservices architecture, service meshes, modern design, containers, backend services, and a high level of automation. Microservice architecture has enabled the loosely coupled integration of service interfaces and the interoperable

systems by creating an abstraction layer based on web technologies. Containers help software to run reliably when moving it from one environment to another (build once and run anywhere!). Container Orchestrators have provided abstractions to make a common multi-cloud environment and an industry-standard (CNCF-conformant) container management platform to avoid any proprietary software layer lock-in in the cloud. Reuse of the same tools for logging and monitoring across existing cloud environments is also possible with native integrations.

Evolution of Cloud Native Security

Container technologies are a continuously evolving space with rampant adoption. The threat landscape for cloud native technologies and the corresponding security challenges in mitigating and resolving these threats evolves as well. These, in addition to a complex ecosystem for secure container platforms, require a fully formulated, well-thought-out security strategy, with technical controls and automation for security policy enforcement, response, and operational discipline.

Containers provide enormous security benefits when appropriately implemented. They provide greater transparency, modularity, reduced attack surface, easier application components updates, and a consistent environment for application components to run. This consistency allows for parallel security to thrive in development, test, and production runtime environments. They also reduce the impact of enterprise wide security incidents when enabling proper isolation built between applications (essentially enabling micro segmentation in enterprises which may have a flat network) as part of a layered defense-in-depth security strategy.

With all the current challenges in security, the number of security tools needed, and the shortage of skills and talent in the market, securing a container platform is a monumental challenge. We expect to see increased migration to the cloud as container service offerings by cloud providers become more mature, with more cloud native security & intelligence tooling integrated over incompatible specifications. These offerings reduce the overhead for enterprises as part of the shared responsibility model.

The threat landscape, however, generally remains the same, with top weaknesses consistently being exploited by the same sets of actors. The most significant changes we see are the manner and mechanisms by which attackers target cloud native organizations and applications. Any attacks on container orchestrators and deployments are increasing, as seen with the increase in cryptomining attacks through infiltrated or trojan horse images. As with any innovative technology beginning to reach market saturation, it was only a matter of time for malicious actors to exploit any low hanging fruit.

As these attacks become more prevalent, more intricate, and expand, cloud native security has to evolve to put a more significant focus on enterprises and

DevOps teams than where it currently resides. We are seeing an increase in the use of security policies as code, but there is a lot of room for evolution and increased automation in security policy enforcement, detection and response. It's evident that immediate and automated security intelligence and responses will be essential to thwart the attacks, and even self-heal from them. Perhaps even adapt and integrate [^10] as they occur.

Container forensics tools and technologies will need to evolve to keep pace with where cloud native is headed. This is particularly critical as the number and complexity of incidents increase in the context of infrastructure-as-a-service and other as-a-service models.

Conclusion

In the past 15-20 years, the community has seen increasing adoption in cloud services and technology, with a recent significant push towards cloud native models. Innovators continue to poke and push this technology forward for mature adoption and testing.

It is critical that organizations at the brink of mature adoption earnestly analyze and apply core security concepts to alleviate the lag in hardening and environmental control for Day 2 operations.

While security-specific guidance and controls may not yet exist for most innovations we see today and coming in the future, core security concepts in cloud native architectures can be consistently applied while designing, developing, and deploying new capabilities.

These core security concepts are:

- Protection from unauthorized access (person and non-person entities) - Ephemerality reduces asset exposure to unauthorized entities by consistently rebasing from a known good state.
- Immutability to preserve the integrity of content and code.
- Availability of services, tooling, and content - Distributed nature of cloud native architectures provides resilience and redundancy.
- Auditing and Accountability - Ensure that detection of irregularities and keeping track of authorized changes is possible.

Appendix

Learning from First Version (New in v2)

Each release of the whitepaper triggers a retrospective process that assesses what worked well, what we should do more and opportunities where we can improve. The first retrospective led to the creation of a survey to gauge the success of the whitepaper through answers to questions regarding the content, engagement, usefulness, relevance, reach, and distribution of the paper.

A quick summary, raw data and feedback of the survey is documented under github.com/cncf/surveys/security

Changes since first version

Several new sections in Security Assurances, Security Principles and Compliance were added. Feedback from the first version retrospective was addressed throughout the paper.

Have Feedback For Us? (New in v2)

If you have feedback for us please open an issue here: https://github.com/cncf/tag-security/issues/new?assignees=&labels=suggestion%2C+triage-required&template=suggestion.md&title=%5BSuggestion%5D+some+descriptive+title. Don't forget to mention the name of the whitepaper and its version number.

SSDF v1.1 Mapping (New in v2)

Cloud Native Application Lifecycle	SSDF Practices/Tasks
1. Develop	
Security Checks in Development	PO.3.1 PO.3.2 PO.3.3 PO.5.1 PS.1.1
Development of Tests	PO.5.2 PW.1.1 PW.8.1 PW.8.2
Code Review	PW.2.1 PW.7.1 PW.7.2
2. Distribute	
Build Pipeline	PO.3.1 PO.3.2
Container Image Scanning & Hardening	RV.1.1 RV.1.2 RV.1.3 RV.3.1 RV.3.4
Testing (SAST, DAST, Security Tests)	PW.7.2 PW.8.1 PW.8.2 RV.1.2
Artifacts Registry & Staging	PW.1.3 PW.4.2 PS.3.1
Signing, Trust & Integrity	PS.1.1 PS.2.1
Encryption	PO.5.2

3. Deploy	
Pre-Flight Deployment Check	PW.9.1 PW.9.2 PS.2.1
Observability and Metrics	PO.5.1 PW.1.3
Incident Response and Mitigation	PW.1.2 RV.1.3 RV.2.1 RV.2.2 RV.3.1 RV.3.2
4. Runtime	
Compute	PO.5.1
Storage	PO.5.1
Access	PO.5.1

References

1. NIST SP 800-204 Security Strategies for Microservices-based Application Systems - https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf
2. NIST SP800-190 Application Container Security Guide - https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf
3. NIST SP 800-218 Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities
4. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf
5. CIS Kubernetes Benchmark
6. Threat Modeling: 12 Available Methods
7. https://owasp.org/www-community/Application_Threat_Modeling
8. MITRE ATT&CK Matrix For Kubernetes
9. 12-factor
10. 9 Box of Controls
11. Cloud Native Security Lexicon
12. Cloud Native Glossary
13. CNCF landscape
14. Four eyes principle
15. Common Vulnerability Scoring System
16. Test Pyramid
17. software bill of materials
18. OPAL - https://trustedcomputinggroup.org/wp-content/uploads/TCG_Storage-Opal_SSC_v2.01_rev1.00.pdf
19. CNCF Storage Whitepaper
20. security boundaries - https://www.oreilly.com/library/view/cissp-certified-information/9780470276884/9780470276884_security_boundaries.html

21. OWASP threat modeling
22. STRIDE
23. OCTAVE
24. ATT&CK's Threat matrix for containers
25. NIST Incident Response Guide
26. Secure Defaults: Cloud Native 8
27. Software Supply Chain Best Practices White Paper
28. Cloud Native Security Map - https://cnsmap.netlify.app
29. Center for Internet Security (CIS)
30. OpenSCAP
31. TAG-Security's use cases listing
32. European Banking Authority - https://eba.europa.eu/documents/10180/2170121/Final+draft+Recommendations+on+Cloud+Outsourcing+%28EBA-Rec-2017-03%29.pdf
33. European Insurance and Occupational Pensions Authority - Guidelines on outsourcing to cloud service providers
34. European Securities and Markets Authority - CLOUD OUTSOURCING GUIDELINES
35. github.com/cncf/surveys/security
36. Feedback
37. https://techmonitor.ai/technology/cybersecurity/ransomcloud
38. https://www.mcafee.com/enterprise/en-ca/security-awareness/ransomware.html
39. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-61r2.pdf

Citations

[1] Another model to consider is Cloud, Clusters, Containers, and Code: https://kubernetes.io/docs/concepts/security/overview/

[2] Example - MITRE ATT&CK Framework for Kubernetes

[3] Shifting security left often leaves organizations to lapse operational security monitoring. It is important that security exists in all parts of the lifecycle and organizations continually evaluate other aspects of their business and technology processes where they may reach beyond modern security paradigms to embrace security as a culture and habit.

[4] Human capital is a vital asset necessary to the success of any organization, the corresponding intellectual property and relational capital brought as a result is equally in need of protection.

[5] https://blog.aquasec.com/malicious-container-image-docker-container-host

[6] According to Applied Software Measurement, Capers Jones, 1996 and adjusting for inflation - 85% of defects are introduced during coding with a cost of $41 to fix compared to a post release fix cost of $26,542.

[7] cisecurity.org maintains a listing of benchmarks for hardening

[8] It is critical to note that while authentication is available for use, mutual authentication is the preferred mechanism to not only verify the client but also the server (outsider versus insider).

[9] Utilization of a VPN does not guarantee encryption.

[10] The concept of regression proofing is best explained as a facet of antifragile behaviors within technology environments. Instead of remaining resilient and robust against adverse conditions and attacks, technology can proactively adapt and thrive when subjected to them.

Acknowledgements

This white paper is a community effort driven by the members of the CNCF Security TAG. Thank you to everyone for their outstanding contributions. Special thanks to Emily Fox and Jeyappragash JJ.


CLOUD NATIVE
COMPUTING FOUNDATION