11 Recommended Security Practices to Manage the Container Lifecycle

Adopting a DevSecOps approach for modern apps

vmware®

Table of contents

vmware®

> The challenge for teams is to prevent vulnerabilities from being introduced, both during development and eventual deployment, and to validate that every container deployed to production is compliant and secure.

Introduction

Most organizations today are on a path to modernize the way they build and deliver applications and services. Whether such applications and services will be used by customers or internal constituencies, the benefits of modernizing are profound. Organizations can deliver revenue-generating features faster and be more responsive to their customers. They can reduce technical debt (and the potential security vulnerabilities hidden there), as well as take advantage of cloud efficiencies. Application packaging, specifically containerization, plays a key role in modernization. However, development, security and operations (DevSecOps) leaders must also modernize the way they think about and practice security, governance and compliance so as to instill confidence in faster release cycles.

This paper details recommended practices for DevSecOps teams looking to move to a more modern application lifecycle methodology. Eleven recommended security practices are discussed, each highlighting the various stages of a container's lifecycle, from how the application is built to how it is stored, deployed and run.

It is not meant to be an exhaustive list. Rather, it is meant to provide a better understanding of DevSecOps practices over the lifecycle of a container that every team must consider when modernizing. Each section of this paper is written to be consumable by itself, so that it can be useful to a wide variety of readers.

Why modern applications require a modern InfoSec approach

Many applications today are running as monoliths, which means all of their core functionality is co-located within the same deployable application bundle. Monoliths are optimized to run for long periods, usually on virtual machines (VMs) or bare metal servers. As DevSecOps teams move to manage policies and security up the layers of abstraction—first from bare metal to VMs, then to containers and orchestration platforms—the complexity only increases.

Containers used as part of a modern software development process can greatly accelerate application development and ultimately help a business create differentiated digital experiences. Containerized applications are typically built as small, individual services rather than large, monolithic applications. This smaller, more modular codebase allows development teams to deliver innovation to users more quickly than they would without the use of containers because they are not slowed down by the rest of the codebase. The challenge for teams is to prevent vulnerabilities from being introduced, both during development and eventual deployment, and to validate that every container deployed to production is compliant and secure.

To run containerized applications at scale, organizations should adopt orchestration platforms like Kubernetes. By using a powerful, common runtime platform across all infrastructure, Kubernetes helps organizations realize a true geographically distributed application deployment model. The automation tooling associated with orchestration platforms allows applications to be run and managed at scale across these distributed environments. Meanwhile, infrastructure management tooling keeps these discrete production systems uniform, predictable and working together as a whole. But these new practices present a challenge for security practitioners.

Many of the tools that DevSecOps teams have come to rely on for lower layers of abstraction are no longer optimal in a modernized world. Virus scanners, patching practices, perimeter-based security and the like are too computationally expensive and are not built for an environment of ephemeral, lightweight containers. Instead, security needs to be built into the development process and baked into the operation of the runtime platform—an end-to-end approach.
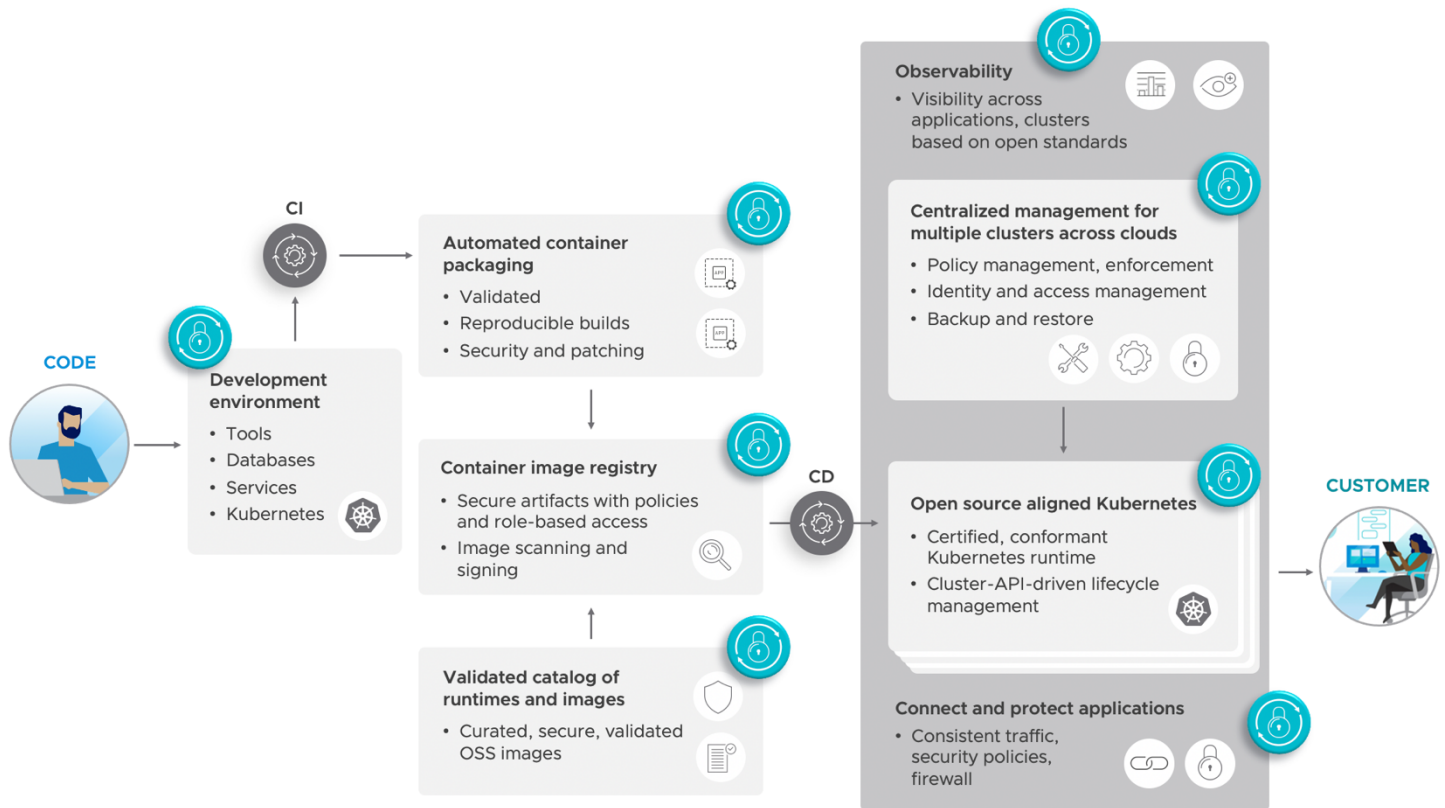
vmware®

FIGURE 1. Security touchpoints across the container lifecycle—from code to customer

As organizations adopt new ways of deploying containerized applications, they need to change the way they implement and manage their DevSecOps policies. Containerization often goes hand-in-hand with automation. And so, like testing, integration, and deployment, security also needs to be built in at the ground level of an application and automated as much as possible.

This baking of security into an application early in the container lifecycle is the so-called "shift left" of an organization's security model. Security teams can kick off the process by using existing policies that align with the necessary governing and compliance frameworks of their organization while also updating the policies to consider the new container and application lifecycle and tools. Development and delivery teams are then responsible for the implementation of those practices, performing the day-to-day decision-making around the security of their application and providing evidence showing that they are meeting the organization's policies back to the security teams.

The result is that security will no longer be an "add-on" to a deployed application—or a hurdle for a development team to overcome as an afterthought. Secure practices will be just "the way things are done."

Thankfully, there is a precedent for this "shift left" thinking. Few teams have questioned the need for regression testing, code reviews and the like. Practices that were normally assigned to separate teams were simply integrated and automated into the development flow. In this same way, security needs to span development, testing and deployment practices. In order for a business to thrive, in other words, its organizational culture around security needs to change.

vmware®

Monolithic, long-running applications are left largely intact while updates to application code or dependencies are applied. This is not, however, a recommended practice for containerized applications.

The following sections lay out 11 recommended security practices, roughly ordered to align with the stages of a containerized application lifecycle.

#1. Make developing secure, cloud native software easier

Many programming language frameworks make adopting recommended security practices and patterns easier for developers. The most prominent of these is Spring, but frameworks for .NET Core and others exist as well. Using these frameworks will help development teams create secure applications by default.

Many of these frameworks include configuration management within the framework itself. Additionally, many can connect to an external configuration service for the centralized management of policies. And numerous security features are included in these frameworks, such as:

• Authentication integration through SSO, SAML, OAuth, and LDAP

• mTLS API communications enabled by default

• Circuit breaker patterns that gracefully shut off communications if a service is unable to be reached

• Protection against attacks like session fixation, clickjacking, cross-site request forgery

These frameworks also make integration with other external services easier. Systems for aggregating log data are useful for operations and security teams alike. And plugins for monitoring systems can allow the same monitoring tools used for the infrastructure layer to send alerts based on application behavior.

Going serverless is one way to abstract away much of the underlying complexity of running an application and allow the function to run in an opinionated and secure abstraction layer. This allows developers to focus on creating features and services. It also enables security teams to implement low-level policies for how those functions are run without disrupting the application or its development.

#2. Build modular container images with layers that can be individually updated

With traditional, monolithic applications, updates are applied directly to the application as it's running. This process is known as "patching" or "patching in place." These long-running applications are left largely intact while updates to application code or dependencies are applied. This is not, however, a recommended practice for containerized applications.

Functionally, container images are built in layers. Whenever one of the layers changes, rather than accessing the running application in place and applying the update, the whole container image should be rebuilt from scratch. Depending on the practices of the organization, however, this process can be heavy-handed and use up a lot of time as developers verify that the application is built correctly and passes all the necessary tests within the application delivery pipeline.

The process of updating a container is a far cry from patching in place. Indeed, it results in an entirely new container image that still needs to be verified, tested and deployed. However, creating a new container does not necessarily require the compiling and rebuilding of the application itself. If, for example, only system libraries need to be updated, in an effectively modularized container only that layer needs to be rebuilt. This reduces the burden on testing and validation practices, as the application code and any other code layers should still be the same as they were in previous builds. And that, too, should be verifiable.

vmware®

Container bloat can result in not only a container that uses more resources than necessary—making the container slower to boot and more expensive to run—but an increased attack surface area for vulnerabilities.
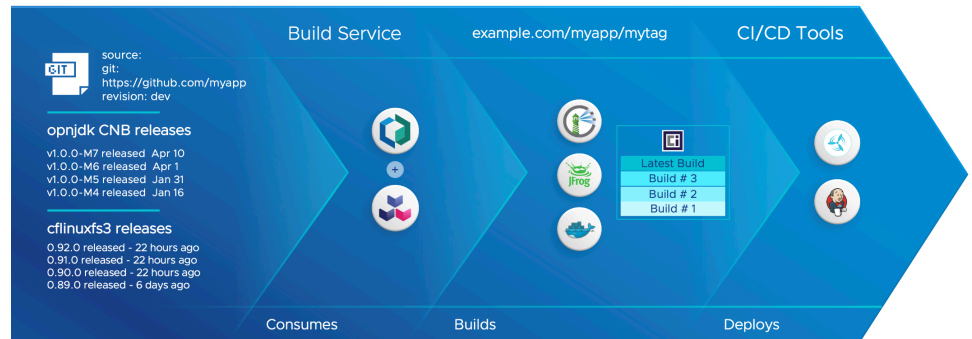

FIGURE 2: Rebase container images with only changed layers using VMware Tanzu™ Build Service™

Every library within the image should be given a digital fingerprint to prove that it has not changed since the time it left its source. Hashing is a tried and tested method for ensuring data has not changed throughout a process, but it can be time-consuming to manage. Instead, metadata within each image can be programmatically inspected for changes and rejected if any are found. Security and compliance team reports can also be programmatically and automatically generated whenever there is a change to an image so that proof of provenance of every bit of code running in production can be verified.

Finally, the pipelines that deliver such images into the production environment can be set up to perform these checks every time a new image is pushed. Container testing and scanning tools can also be integrated at this layer to provide further verification and reporting, but this should not be the only line of testing undertaken. These practices enable organizations to build a robust deployment and updating practice for containerized environments, one that allows for continuous updating and offers verifiable proof of provenance backed by auditable container metadata and automated reporting for security and compliance teams.

#3. Build secure, custom containers

To build a container image, all code, libraries and tools must be explicitly pulled in from other locations. Most teams will pull these in as part of the container image's base operating system (OS) and dependency layers. But for applications written in different languages and frameworks, organizations are presented with a trade-off: managing either a huge number of customized base OS and dependency images, each hardened with only the dependencies needed to run a specific application, or a smaller number of images, each packed with everything that any application might need. In the latter case, container images can become quite large. This is known as "container bloat." In many cases the base OS image may not even be hardened or updated on a regular basis.

Container bloat can result in not only a container that uses more resources than necessary—making the container slower to boot and more expensive to run—but an increased attack surface area for vulnerabilities. The more code you have in your container, the higher the likelihood that there is also an unpatched security flaw from the Common Vulnerabilities and Exposures (CVE) list or malware embedded in a library. So how can organizations verify that only the code necessary to run an application is included in an image, eliminating the need to manage individual base OS and dependency layers for every application?

Some organizations will use a hardened version of a base OS, like Alpine Linux or Photon OS. Others will choose to build their own base OS layer for use within their containers. In either case, all of the code used in the base OS, as well as application dependencies necessary to run the application, should be verified via metadata compiled and included within the container image. The name of the included library, the version, as well as the

While images from a public container image registry are widely used, they can leave an organization exposed to vulnerabilities.

open-source license and version of each package (if applicable) and any known CVEs included within it would be useful for DevSecOps teams. All of this data is available for public consumption and should be included within the container. A hash value for every layer and package within the container should also be provided to verify that neither the container nor its contents have changed since they were initially created.

Metadata like package names, versions and CVEs will allow DevSecOps teams to set policies around what types of applications are deployable into production environments. (Perhaps older versions or packages under a certain open-source license should not be deployed, for example.) This data also gives external audit teams easy, programmatic ways to verify and report on the status of governance and compliance frameworks to which an organization might be required to adhere. What's more, hashing values provide a way to confirm that an included package has not changed (to possibly embed malware or the like) since it was verified at the source of the code repository.

#4. Secure use of third-party containers

Custom applications are just one piece of the puzzle. To run them, organizations leverage other containerized applications from external repositories and third parties, such as databases, as well as from caching, logging, monitoring and other similar services. All of them need to be treated with the same scrutiny as containers built using source code written in-house—perhaps more so, since organizations do not typically control the code within them.

Traditionally, if a developer needed to use one of these services as part of an application stack, they would choose an image from a public container image registry. But while these images are widely used, they can leave an organization exposed to vulnerabilities.

That's because such images provide little in the way of verification methods to confirm that all the code within the container is safe to use in production. Updates may also be sporadic, and any included libraries will be outside of the control of the organization and as such might add to the container's overall attack surface. Different development teams might deploy the same service from different providers, further complicating the management and security of the overall production system for operations teams. How can organizations give development teams the flexibility they need to deploy these services while preserving the ability to maintain the environment?

They need to set up a single, private container registry where they can place all approved container images and base OS images. Their operations teams can then implement cluster policies that only allow container images that come from these approved sources to be deployed. But how can they build and manage these containers, keep them up to date with the latest patches and features, and determine whether CVEs are being mitigated regularly?

Like custom applications, these containers are also constructed in layers—from the base OS to the build dependencies to the application code. And any time one of these layers gets updated, especially if there is a critical CVE patched or a new feature an organization wants to take advantage of, the organization should be notified, and a new container built automatically. Then the organization will have the ability to deploy that new container into production if desired.

The images should also provide meta information detailing the inventory of libraries and tools within the container, what was last updated, how the image was hardened, and any vulnerability scan details that might be produced with the image.

Organizations that allow teams to pull and run applications from any registry into production are vulnerable to threats, whether from unpatched CVEs or from malware embedded within the image layers.

#5. Adopt control policies for images in your environment

A private container image registry allows teams to store container images securely and privately so they can be pulled and run on a container runtime platform later on. This is in contrast to a public image registry, where container images are accessible by the general public. With a private registry, operations teams can validate images, set usage policies, and keep images refreshed and updated so that developers can leverage them.

Most organizations use a private container registry for their developed applications. This represents a portion of the organization's intellectual property, and protecting it is of the utmost importance. But many organizations don't use private registries for either their third-party or open source applications. Instead, they pull images from public repositories and run them on critical production infrastructure. Organizations that allow teams to pull and run applications from any registry into production are vulnerable to threats, whether from unpatched CVEs or from malware embedded within the image layers. If your organization can't guarantee that the source of an image is safe, assume that it is not.

Instead, a private container registry of application images should be made available to developers. For open source services, teams should adopt policies and practices to build these applications internally, and lock down their deployment to validated versions only. In this way, they can create a more uniform, trusted and verifiable environment. The development teams should also have all application containers created through automated pipelines. This way, the automated build system can take any changes or vulnerabilities and rapidly release new versions while also automating the variety of security, testing and compliance checks the organization requires

Additionally, with a private registry, access controls can be applied to limit users or services that can push images into it. Recommended practices suggest that only automated pipelines be allowed access to push images to a registry. The operations teams can manage the authentication credentials for these automated users, which can help prevent malicious code from being integrated by a bad actor.

For the pulling and running of container images, it is further recommended to disallow the pulling of older versions of an image. For example, employing a rule that only N-1 versions of application images can be pulled means that only the most current version or at most, one version back can be pulled and run, limiting the exposure of older, unpatched code running in production.

Of course, teams can go around these policies. Theoretically, they can create a container repository without any of the aforementioned restrictions. That is why it is important for security, development and operations teams to work closely together. If there are burdensome hurdles causing teams to go around approved practices, discuss those as a team. Adversarial relationships will only lead to a breakdown of the overall process.

As a way to enhance compliance of even the most agreeable teams, some cluster management tools do allow the images deployed on them to be restricted—only allowing containers to be run if they are from a certain registry, for example, or signed with a valid certificate. These practices can help organizations verify that only containers from a trusted registry, pushed from an authorized, automated and secured pipeline with restricted individual access can run in their production environments.

#6. Secure the container runtime platforms

Kubernetes clusters provide a relatively easy platform on which to create and to run applications. Securing them, and ensuring they include ubiquitous and uniform policies, however, is a major challenge for security teams—one that requires collaboration with development and platform operations teams.

Organizations should implement a zero-trust, role-based access control policy for accessing the runtime platform itself. For most use cases, automated pipeline workflows

along with centralized management dashboards can be used to make changes as opposed to granting direct access to the platform itself. In this way, security teams can validate that all changes are performed only by pipelines that have been rigorously tested, and that any changes that are made are adequately tracked so as to provide a detailed audit log.

Credentials for these users should be stored securely off of the platform, preferably in a centralized management system for all runtime environments. These credentials should be treated just as they would be for access to any other business-critical system. There are many options available, but their integration capabilities for orchestration systems like Kubernetes can vary widely. In most cases, a Kubernetes-native (in other words, one built specifically for Kubernetes) secrets management system is preferred.

For applications, explicit control policies can be put in place on the platform to only allow containers from trusted sources to be able to run on production clusters. This can be done by using embedded certificates within the container or by whitelisting sources at the DNS level (or both). Platform operations teams can then validate that the containers running in their environments are from trusted sources, which themselves have also validated the application dependency supply chain back to the source code. The automated pipelines that manage the containers should also automatically enforce these controls, preventing unauthorized containers from being pushed or run within the production platforms.

The containers and production platforms are also a product of infrastructure as code. This means that operators, security and audit teams can easily validate that only authorized configurations for the platform and running containers are in place. It can also allow for easy identification of the history of previous changes as well as new versions that are pushed, ensuring that all changes are authorized, meet the organization's governance requirements, and are allowed in the production environment.

#7. Use a centralized control plane to manage container runtime sprawl

Many organizations want to centrally manage all of their clusters from one location, despite how challenging that can be. Indeed, the ability to view every cluster under management including their status, version, running workloads and security policies— across all types of infrastructure—is a growing necessity. This becomes even more important for an organization as the number of clusters they have continues to grow.

A centralized system can enable an organization to continue self-service access to new Kubernetes clusters while maintaining the overarching security policies set by their security team. These policies could include controls over what versions of Kubernetes can be deployed, which can help mitigate the deployment of older, patched CVEs within the container orchestration platform itself. They could restrict the applications that can be deployed to only those from trusted sources. They could also include role-based access control policies, to control who can access the cluster directly.

Setting strong access control policies on every cluster can help mitigate privilege escalation attacks. In those situations, attackers use an initial vulnerability to gain access to a container, then through a series of attacks, continue to increase their privileges by authenticating as other users and workloads, gaining access to the node, cluster—even the infrastructure-as-a-service hosting system.

Kubernetes upgrades can become burdensome for platform teams, which prevents them from deploying the most recent updates and patches consistently. This can leave an organization vulnerable to attacks.
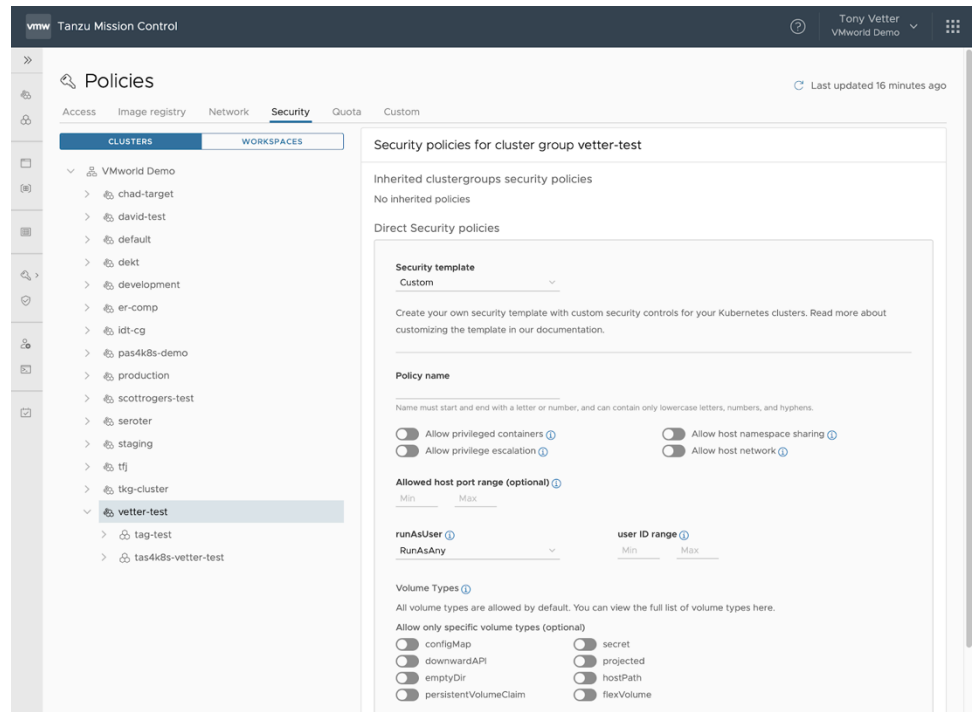

FIGURE 3. Centralize policy management across clusters with VMware Tanzu™ Mission Control™

A centralized management system gives operations and security teams the ability to manage policies on every cluster uniformly along with the flexibility to set policies for individual clusters by teams, as necessary. This, in turn, allows development teams the flexibility they need to run custom services on the cluster, while the security teams can manage the overarching policy for that individual cluster as easily as they would for the entire fleet.

A centralized management system can also provide a central point to monitor for compliance cluster-hardening requirements from CIS and others. When an organization's clusters are centrally monitored, with CIS reporting across all of them, it becomes possible to add assurances that the Kubernetes clusters running critical applications are not only up to date but are hardened according to industry best practices.

#8. Continuously update the infrastructure layer

Major open source Kubernetes upgrades are released about every six months, with a consistent stream of patches coming in between. These upgrades can become burdensome for platform teams, which prevents them from deploying the most recent updates and patches consistently. This can leave an organization vulnerable to attacks.

Organizations should apply a centralized system for Kubernetes management to make the process of creating new and fully up-to-date Kubernetes clusters easy and automated. That way, when a new patch or major version becomes available, operations, development and security teams can work together to create a new cluster, and then use automated pipelines to start testing the deployment of applications onto the new version. A centralized system also makes it easier to identify when clusters across the organization require upgrades to limit security risk or exposure from unpatched clusters.

To streamline this process across Kubernetes platforms, all clusters need to be managed using a uniform API such as Cluster API, kubeadm, KOPS or a host of others. These API layers allow for individualized management of a single Kubernetes cluster for actions like

An application is broken up into services, each of which must, in turn, communicate with one another to facilitate the look and feel of a full application to the end user. These communications can be vulnerable to outside threats and should be secured and monitored.

bootstrapping and upgrading, but also more complex events like backup and recovery. They should plug into an overarching management system wherein operations teams can implement global policy once and allow the management system—along with the individual API layers for each cluster—to take action.

This practice will help eliminate old deployments and misconfigurations on the Kubernetes cluster at regular intervals. It will help disrupt any persistent threats running on the cluster, like command-and-control (C&C) botnets or data exfiltration attacks. From an operational standpoint, creating a new cluster beside the older version will limit or even eliminate downtime, as traffic can be redirected to the new cluster as soon as it is ready.

#9. Secure inter-container communications

Unlike VMs or other ways of packaging monolithic applications, containers will likely not contain all of the necessary components to run a full application stack. Instead, an application is broken up into services, each of which must, in turn, communicate with one another to facilitate the look and feel of a full application to the end user. These communications can be vulnerable to outside threats and should be secured and monitored.

Most services will use APIs to perform this communication. A Kubernetes environment can use a secure application service mesh to enable them to communicate, through which traffic can be encrypted. The inclusion of service discovery allows new services to be added securely and automatically into the communications mesh so their traffic can be routed appropriately.
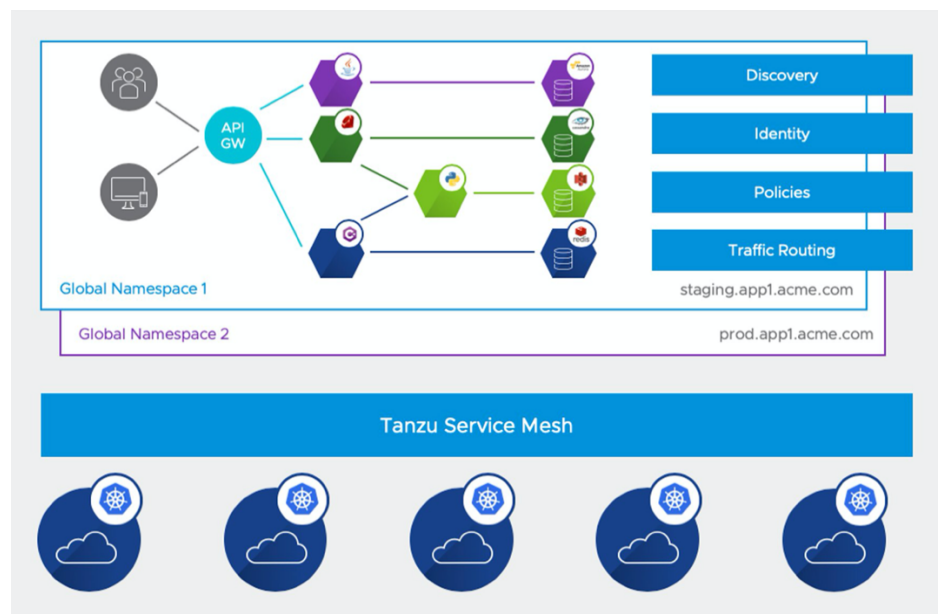

FIGURE 4. Manage grouped applications by namespace using VMware Tanzu™ Service Mesh™

Access policies for the network need to be generated and ubiquitously and uniformly implemented on every Kubernetes cluster under management, much like securing the platform itself. Using a centralized Kubernetes control plane can help operations teams verify that every new cluster created will have embedded features that bring the cluster up to date with the latest policies.

Malicious workloads exploit vulnerabilities in the DNS protocol used by Kubernetes to route traffic in and out of the cluster. A robust observability platform can be used to monitor DNS traffic and alert against such anomalies.

Thresholds on these communications can also be added at the individual application firewall level, where rate limits can be set for communications between individual containers. This can help limit the risk of a bad actor trying over and over again to access adjacent containers. Restrictions can also be set around where these communications can be accepted from, preventing bad actors from jumping from one container to another. Securing these communications is a lot like setting up access permissions: Containers should only be allowed to talk to the containers they need to do their job—nothing more, and nothing less.

Using VMs as Kubernetes nodes to further segment traffic and secure application boundaries is another good practice. Containers themselves will share resources with the host but will not provide the same level of segmentation as a VM does. If an attacker gains access to an individual node, this practice will help prevent them from easily accessing the rest of the cluster.

Monitoring the service mesh is also crucial. Observing traffic patterns is a useful practice for operations teams, as highlighting anomalous behavior could draw attention to an attacker trying to gain access or a persistent threat already running within the cluster. Traffic thresholds should be set to alert when a pattern arises so that the offending workload can be investigated as either a malfunctioning service or a bad actor within the system. But care must be taken to not set thresholds in a way that also produces false positives. Too many false alerts can lead to security teams ignoring errors, and subsequently missing any true issues until it's too late.

#10. Observe everything

Observability extends beyond the network layer and into every layer of the application and infrastructure stack. It is therefore crucial to verify that the security policies put in place are doing what they were intended to do. For example, C&C workloads are a major category of risk in Kubernetes environments. These malicious workloads exploit vulnerabilities in the DNS protocol used by Kubernetes to route traffic in and out of the cluster. A robust observability platform can be used to monitor DNS traffic and alert against such anomalies.

Other malicious workloads can be spun up at the node level. These containers might not be managed directly by Kubernetes, and so avoid detection. Attackers can spin up these workloads by compromising the Kubernetes admissions controller, then using it to create containers or spin up sidecar applications for containers that are already running in order to exfiltrate data. Monitoring workloads and resource utilization can alert security teams to the presence of such threats.

Reports generated from these observability platforms can also aid in auditing processes for compliance requirements. Custom reports should be generated regularly to provide security and compliance teams with the information they need to document compliance and perform audits. Teams can save time by using the observability platforms to automatically generate as much data as possible whenever it is needed.

Monitoring applications has significantly changed, with applications now split across many microservices and clusters. When an organization grows to hundreds or thousands of nodes or containers, it is no longer possible to monitor individual components using legacy systems. With attackers going after modern distributed systems, a detailed observability system will be able to identify attackers before they get a foothold in the environment. For example, applications can log input validation errors into a central repository. Then, through programmatic observability of the microservice application logs, attackers scanning or trying to exploit applications will be easily identified. Operators and security teams can then monitor and secure the infrastructure while the development teams patch the possible compromises the attackers were trying to leverage.

#11. Get a trusted DevSecOps partner

Even the most technologically mature organizations might need help implementing these security practices. By partnering with a trusted vendor, organizations can leverage that partner's cumulative experience alongside their own internal expertise. This is not about outsourcing the problem, but engaging in side-by-side collaboration, in-depth reporting, and pairing between engineers to build a more robust, secure system across the container lifecycle.

Professional services organizations will be able to help avoid any practices that seem best in the short term but might have far-reaching implications for other systems or teams. They can also foster better collaboration among security, operations and development teams, and can help integrate systems for more robust automation and easier manageability.

Leverage regular check-ins with your vendor partner to verify everything is running smoothly. Build a staged plan with future features that they've implemented and share it internally. This early-and-often internal collaboration will foster goodwill across teams, and when they're given enough time to plan, those teams can ready themselves for new capabilities yet to be added. Then teams can iterate on that plan together while continuing to build more secure platforms that will keep organizational as well as customer data safe and secure.

Embrace DevSecOps with VMware Tanzu™ Advanced Edition

In order to achieve the level of oversight and automation needed to build and maintain a secure container lifecycle, organizations need to have the right tools and capabilities. The VMware Tanzu Advanced edition is meant to help organizations create a secure container lifecycle from start to finish—from automated container builds to applications running safely in production on hardened Kubernetes clusters to the management of policies across clusters and clouds.
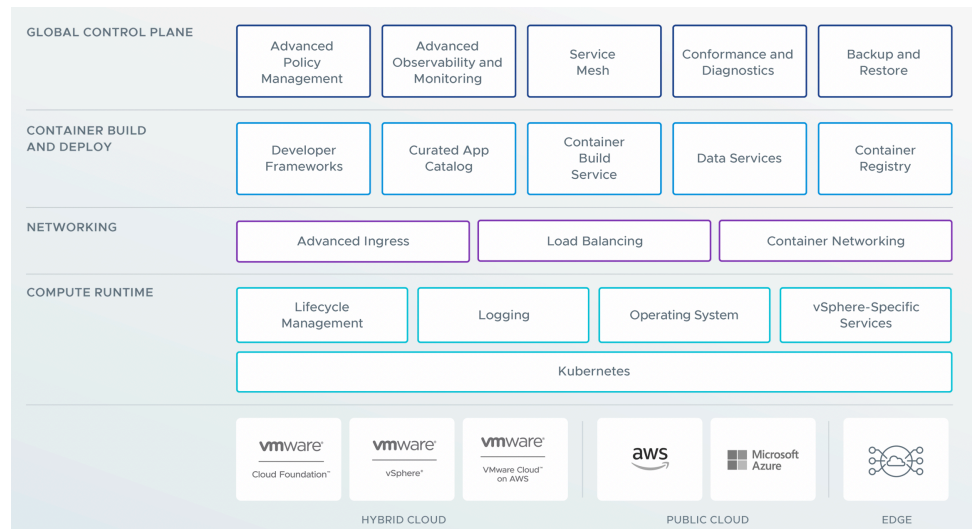

FIGURE 5. VMware Tanzu Advanced offers full-stack DevSecOps

Here are some key Tanzu Advanced capabilities that enable a secure container lifecycle:

• With the popular VMware Spring Runtime® framework, developers have access to the entire ecosystem of Spring services (including Spring Cloud Gateway and Spring Cloud Data Flow) to help them build secure, maintainable and differentiated Java applications and deploy them in a cloud native way. In addition to the power of Spring, support for

ADDITIONAL RESOURCES

- *No YOLO: Containers, Governance, and You* (webinar featuring Stephen O'Grady, Industry Analyst at Redmonk, Joe Beda, Principal Engineer at VMware, Graham Siener, VP of Product at VMware)
- *Three Essentials for Delivering Containers at Scale: A Real DevSecOps Approach* (webinar introducing VMware Tanzu Advanced edition)
- *The Future of Software Delivery Needs DevSecOps* (webinar featuring Sandy Carielle, Principal Analyst at Forrester)
- *Streamline Open-Source Security Compliance on Kubernetes* (VMware blog)
- *Best Practices For Container Security* (a report published in July 2020 by Forrester)
- VMware Tanzu Editions website
- VMware Tanzu Labs website

OpenJDK and Apache Tomcat—two widely used open source building blocks for Java applications—is also provided.

- Developers can use container building blocks from a secure, hardened and frequently updated inventory of curated images via VMware Tanzu Application Catalog™. The catalog, which is deposited into an organization's private container repositories, delivers a set of pre-packaged container images and Helm charts from the Bitnami collection. Each container is continuously updated and comes with metadata proving the trustworthiness of the software within.
- Tanzu Build Service improves an organization's security and compliance posture with centralized control of container updates and patches. It automatically builds and continuously maintains containerized applications. Through integration with existing CI pipelines, any change to upstream code, application dependencies or base OS triggers rebuilding and retesting of containers. Because new images are delivered automatically to the registry whenever a configuration changes, organizations can have consistent and always up-to-date container images. What's more, code provenance is clear: Custom applications are signed to maintain their validity and integrity.
- With the Harbor™ container registry, organizations can consistently and securely manage artifacts across cloud native compute platforms like Kubernetes. It secures artifacts with policies and role-based access control, scans images for vulnerabilities, and signs images as trusted.
- Tanzu Mission Control hardens Kubernetes clusters by creating a logical organization of clusters and access management and enforcing policies like those dedicated to access, security, image registries, networks, and quotas. Security teams can set overarching security policies on every cluster and operations teams can set access and deployment policies. Meanwhile, development teams retain the ability to spin up new infrastructure that complies with all organizational policies with just a couple of clicks.
- When applications are running in production, Tanzu Service Mesh built on VMware NSX® will monitor and secure all application components. It provides the granular authorization and encryption features necessary to secure communications and protect data in transit.
- VMware Tanzu Observability™ by Wavefront gives operations and security teams insights into every layer of the production stack. Observations via custom dashboards can highlight not only process and application bugs, but security concerns. If traffic or compute patterns start behaving abnormally, operations teams have all the necessary data to highlight and investigate these issues in collaboration with external development and security teams.
- The VMware NSX® Advanced Load Balancer™ Intelligent Web Application Firewall delivers high-performance web application security using the CVE catalog of known threats as a blueprint.
- VMware Tanzu Kubernetes Grid™ provides an open source-aligned Kubernetes runtime for your data center, public clouds and edge. It integrates seamlessly with Cluster API and Tanzu Mission Control to simplify the provisioning, upgrading and management of Kubernetes clusters and cluster groups. Operators can manage the lifecycle of any cluster or groups of clusters provisioned with Tanzu Mission Control.

Organizations might also need additional expertise and development support to build out a Kubernetes-based platform with DevSecOps principles in mind. VMware Tanzu™ Labs™ is a team of expert Kubernetes and software development practitioners who have completed thousands of engagements. The Tanzu Labs team can introduce organizations to the cloud native patterns and practices that enable DevSecOps, as well as support the design and build of a custom platform leveraging Tanzu Advanced components that will drive meaningful business outcomes securely and at scale.

vmware®