



e-book

Mastering Kubernetes Security

Exploring the core principles and components
of Kubernetes security.



Introduction

Kubernetes has solidified its position as the leading orchestrator and manager of containerized applications. As businesses continue to transition their operations to the cloud, Kubernetes provides the scalability and flexibility necessary to accommodate their evolving needs. However, this expanded capability is accompanied by the challenge of ensuring robust security within Kubernetes environments.

Securing a Kubernetes environment is a complex task that requires a thorough understanding of various aspects, including compliance, vulnerability management, admission controls, real-time detections/auditing, authorization, access, and secrets management.

This “Mastering Kubernetes Security” eBook is designed to equip you with the knowledge and tools to effectively secure your Kubernetes environment. Whether you’re a business leader trying to grasp the importance of Kubernetes security or an IT professional seeking practical guidance on implementing security measures, this eBook is a great resource to get you started on your journey.

1

Chapter 1: 10 Reasons Business Leaders Should Care About Kubernetes Security

Kubernetes security is not just an IT concern but a strategic business issue. The security of your Kubernetes environment is directly linked to your company's financial health, reputation, and customer trust. The first chapter provides ten compelling reasons business leaders should prioritize Kubernetes security. From protecting against financial losses and safeguarding reputations to ensuring compliance, securing intellectual property, enabling digital transformation, supporting business continuity, facilitating innovation, enhancing competitive advantage, reducing risk, and driving customer confidence, effective Kubernetes security offers many critical business benefits.

2

Chapter 2: Simplifying Compliance in Complex Kubernetes Environments

Compliance in Kubernetes environments is a complex task but an essential one. It involves adhering to industry standards, regulatory requirements, and best practices. Non-compliance can lead to penalties, reputational damage, and even business loss. However, maintaining compliance in a dynamic Kubernetes environment doesn't have to be daunting. This chapter explores the criticality of compliance and emphasizes the importance of regular auditing to identify non-compliance issues and take prompt corrective actions. Learn how Uptycs unified CNAPP and XDR platform automates compliance checks, offers real-time reporting, enforces policies, and enhances your Kubernetes environment's security posture.

3

Chapter 3: NSA Hardening Guide

This chapter explores hardening Kubernetes environments using resources like the NSA Hardening Guide as a reference point to maintain adherence to industry standards and best practices. The importance of regular auditing to identify non-compliance issues and the need for prompt corrective actions are emphasized. Uptycs unified CNAPP and XDR platform is highlighted for its role in detecting and responding to security threats. It automates compliance checks, offers real-time reporting, and enforces policies, thus enhancing your Kubernetes environment's security posture.

4

Chapter 4: Vulnerability Management with Runtime and Shift Left Controls

Vulnerability management is an essential aspect of Kubernetes security. This chapter details how Uptycs integrates with CI and registry scanning tools to scan images for vulnerabilities, malware, and secrets. The platform's capabilities in vulnerability prioritization, KPI measurement, and continuous improvement in fixing vulnerabilities are explored.

5

Chapter 5: Runtime Admission Controls

This chapter highlights the significance of Runtime Admission Controls (RACs), which add a layer of protection by ensuring compliance and verification post-admission controller's approval. It introduces open-source solutions like OPA Gatekeeper and Kyverno for implementing admission control in Kubernetes. It discusses how Uptycs integrates with these tools to act as an actual admission controller, providing visibility into admission failures and audits.

6

Chapter 6: Authorization, Access, and Secrets

Chapter 6 delves into managing authorization, access, and secrets in Kubernetes environments. It explains the Role-Based Access Control (RBAC) mechanism for managing access to Kubernetes resources. The native admission control provided by Uptycs for image and runtime deployments, which allows blocking images with critical or high vulnerabilities and enforcing policies based on business needs, is discussed. This chapter also covers Uptycs' integration with registry scanning to prevent the use of vulnerable images for container deployments.

7

Chapter 7: Real-time Detections & Auditing

This chapter emphasizes the importance of real-time detection and auditing capabilities for Kubernetes environments. It elaborates on Uptycs' single console view for triaging and generating evidence for compliance issues. The platform's ability to perform continuous posture checks on existing and new infrastructure, providing remediation information and real-time evidence, is detailed.

8

Chapter 8: Securing Your Kubernetes Environments via Network Policies

The final chapter discusses securing Kubernetes environments using network policies. It explores Uptycs' ability to provide visibility into pod-to-pod communication, detect unauthorized network connections, and monitor traffic for security threats. The platform's integration with open-source solutions like Calico, Cilium, and Istio is detailed for implementing and managing network policies in Kubernetes environments. Overall, this chapter highlights how Uptycs can enhance your Kubernetes environment's security posture by providing comprehensive network security capabilities.

Kubernetes Security:

10 Critical Considerations for Business Leaders



Kubernetes security may not be part of the everyday interactions of non-technical business leaders, but it directly affects their responsibilities and the organization's overall health.

The security risks associated with Kubernetes environments are two-fold. First, they are due to the complexity of Kubernetes, which can make it challenging to maintain and secure. Second, they are also due to increased cyber threats aimed at Kubernetes deployments. Kubernetes applications run in a distributed environment that spans multiple nodes, and all of these nodes must communicate with each other to enable container orchestration. As a result, criminals view insecure Kubernetes environments as a prime target for attacks.

Kubernetes' security can be a significant challenge for business leaders. Given the complexity of Kubernetes environments, it can take considerable resources to adequately secure it. Security issues can result in substantial downtime, financial loss, legal costs, lost productivity, data loss, and a damaged reputation.

Here are ten reasons why these leaders should prioritize Kubernetes security:

1. **Financial Impact:** Security breaches can cause severe financial losses, ranging from immediate expenses like fines and legal fees to long-term consequences such as lost business and decreased customer trust.
2. **Regulatory & Compliance Implications:** Many industries operate under strict regulations that demand robust data protection and security. Failure to comply due to a security breach can result in heavy fines and legal repercussions.
3. **Reputation Management:** A company's reputation is invaluable. Security incidents, especially those that compromise customer data, can damage an organization's reputation and make it challenging to regain customer trust.
4. **Business Continuity:** Security vulnerabilities can disrupt business operations. If services are taken offline due to a breach, it leads to revenue loss and hampers overall business processes.
5. **Strategic Planning:** Kubernetes frequently plays a role in an organization's digital transformation or modernization strategy. Understanding the security implications of Kubernetes is crucial for effective planning and decision-making.
6. **Risk Management:** Modern businesses constantly evaluate risks and devise strategies to mitigate them. Kubernetes security is a significant aspect of the broader IT risk landscape, and leaders need to comprehend its importance in order to make informed decisions.
7. **Investor & Partner Confidence:** Investors and business partners seek assurance that the organization responsibly manages

its technological assets. Prioritizing Kubernetes security can strengthen stakeholder confidence.

8. **Informed Decision Making:** Even if non-technical leaders are not directly involved in tech decisions, understanding the security posture of the company's technology stack empowers them to make better-informed decisions in other domains, such as finance, marketing, and strategy.
9. **Fostering a Security-First Culture:** Leaders play a crucial role in shaping organizational culture. By emphasizing the significance of Kubernetes security, leaders can foster a broader security-aware culture and prioritize it at all levels of the organization.
10. **Future-Proofing the Business:** As technology evolves, security will become increasingly vital. By grasping and prioritizing Kubernetes security now, business leaders can better equip their organizations for future challenges.



Simplifying Compliance in Complex Kubernetes Environments



Maintaining compliance in dynamic Kubernetes environments is a formidable challenge that compliance officers and security managers face in today's ever-evolving technology landscape.

The constant flux of containers, the complexity of various regulatory frameworks, and the need for continuous visibility make it a daunting task. Compliance in Kubernetes environments is a complex affair due to several factors:

- **Dynamic Nature of Kubernetes:** In Kubernetes environments, containers are frequently created, updated, and destroyed. This dynamism makes it difficult to track and ensure compliance across all containers and pods.
- **Compliance Requirements:** Compliance officers and security managers must adhere to various regulatory frameworks and industry standards, including PCI DSS, HIPAA, and GDPR. These standards demand the implementation and monitoring of intricate technical controls and policies in Kubernetes environments.
- **Lack of Visibility:** Traditional compliance processes often rely on manual audits and assessments, which are time-consuming and prone to errors. In Kubernetes environments, where containers are constantly changing, maintaining visibility and ensuring compliance becomes even more challenging.

So, how can we overcome these challenges? The answer lies in adopting a unified CNAPP (Cloud Native Application Protection Platform) and XDR (Extended Detection and Response) platform like Uptycs.

Streamlining Compliance with Uptycs

Uptycs offers comprehensive compliance solutions that automate compliance checks, provide real-time reporting, and enforce policies. Here's how it works:

Automated Compliance Checks: Uptycs automates compliance checks against various regulatory frameworks and industry standards, continuously monitoring the Kubernetes environment to ensure all containers and pods meet the required compliance controls.

Real-Time Reporting: Uptycs provides real-time reporting on compliance status, giving compliance officers and security managers instant visibility into their Kubernetes environments' compliance posture. This allows them to identify and address compliance issues promptly.

Policy Enforcement: Uptycs enables organizations to define and enforce policies across their Kubernetes environments, ensuring that all containers and pods adhere to these policies, thereby reducing the risk of non-compliance.

Integration with Existing Tools: Uptycs integrates seamlessly with existing security and compliance tools, allowing organizations to leverage their current investments. It consolidates compliance data from various sources, providing a unified view of compliance across the entire Kubernetes environment.

The Benefits of Simplifying Compliance Management with Uptycs

By leveraging Uptycs, organizations can not only streamline their compliance processes but also reap several benefits:

Reduced Risk: Automated compliance checks and policy enforcement minimize the risk of non-compliance.

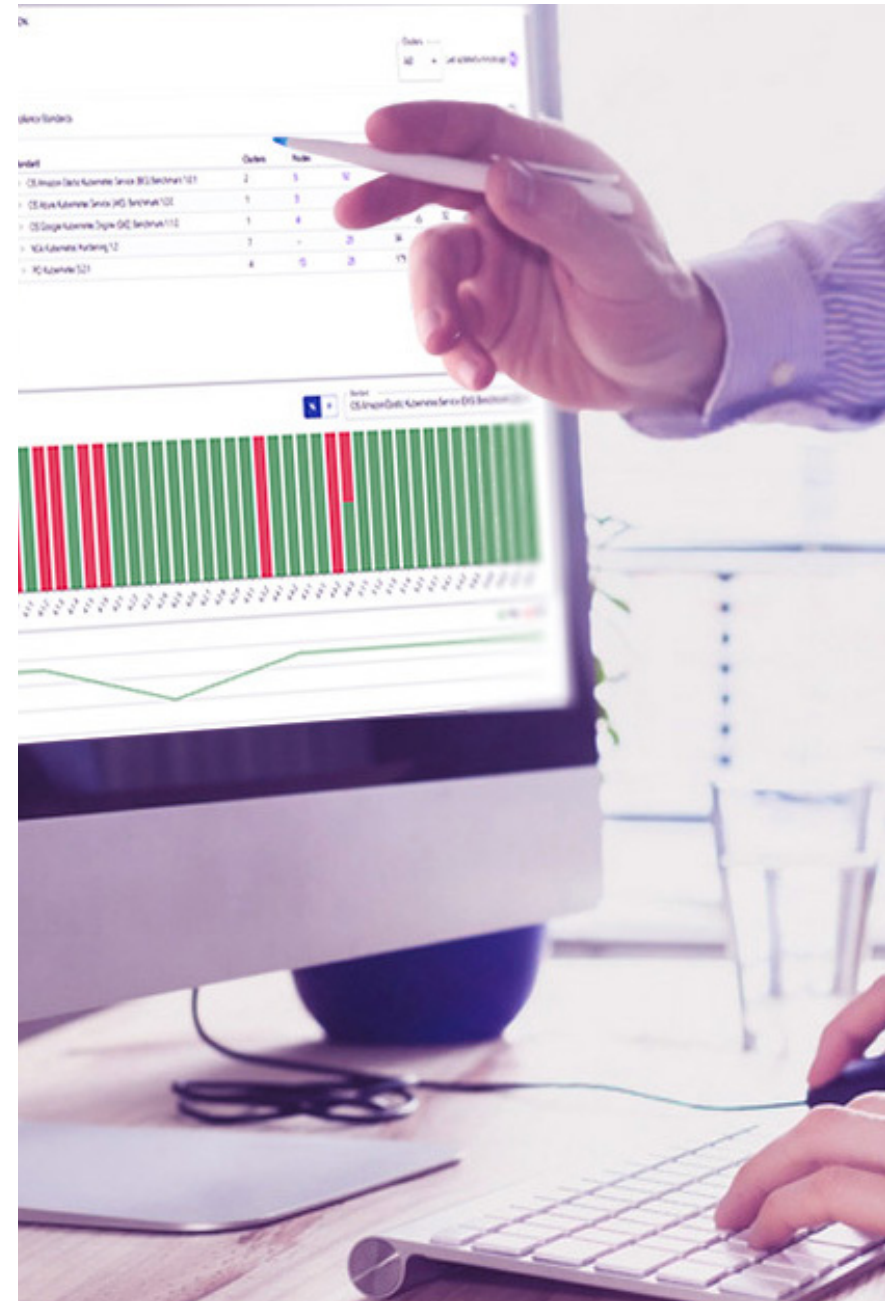
Improved Efficiency: Real-time reporting and integration with existing tools enhance operational efficiency.

Time and Cost Savings: Automation eliminates manual audits and assessments, saving considerable time and resources.

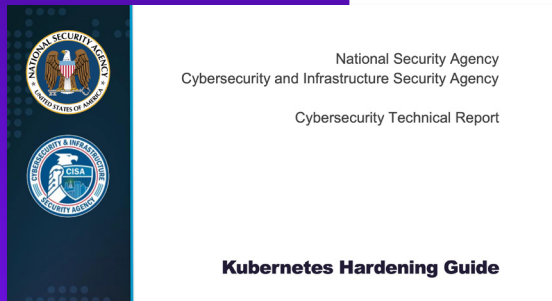
Peace of Mind: With Uptycs, compliance officers and security managers can rely on a trusted platform to ensure continuous compliance in dynamic Kubernetes environments.

Conclusion

Maintaining compliance in dynamic Kubernetes environments is indeed a complex task. However, with Uptycs' compliance solutions, this task becomes manageable. Through automation, real-time reporting, policy enforcement, and seamless integration with existing tools, Uptycs simplifies compliance management in Kubernetes environments.



NSA Hardening Guide



Organizations across the globe are embracing Kubernetes and harnessing its power to deliver cloud-native applications on top cloud providers like Amazon Web Services (AWS), Azure, and Google Cloud Platform. However, Kubernetes' default security limitations mean safeguarding your cluster and data should be a top priority.

Securing Kubernetes can be a challenge, particularly since clusters are a common source of misconfigurations in the cloud, can be shared between multiple teams, and the learning curve for Kubernetes is steep. But there is help available. Last August, the National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA) released an updated version of the Kubernetes Hardening Guide.

The guide helps organizations solve those key misconfigurations and provides guidance to enterprises on solving key misconfigurations such as:

- **Access controls.** The most secure access follows the zero trust model, i.e., only give what is absolutely necessary. However, in Kubernetes, many users can end up getting cluster role binding privileges, which gives them access to the entire cluster when they don't necessarily need or shouldn't have that - especially in shared multi-tenant cluster use cases.
- **Network Security.** Since all pods can talk to each other by default, this increases the lateral attack surface inside the cluster. The NSA guide gives great insight into how to solve this using Kubernetes network policies.

- **Container runtime issues.** Examples of this include overprivileged containers and containers with mutable file systems, which can lead to vulnerabilities and can lead to breakout.

While the insights the guide provides are a fantastic first step, there are some key challenges we saw in the market when it came to using the recommendations:

1. **Operationalizing the Insights and Misconfiguration Failures:** While the guide was a great start, it was hard for customers to keep up with an entire single PDF guide and operationalize it as part of their day to day processes without codifying the rules. Non-compliant infrastructure should be visible in a single pane of glass, and any failures should be easily assignable to DevOps and platform teams responsible for the infrastructure.
2. **Enabling continuous posture checks on existing and new infrastructure:** Every time a new cluster is deployed or a new namespace is provisioned, the same NSA checks should be done on new infrastructure without manual intervention. Moreover, without a system in place, it is hard for customers to do daily checks, leading them to do it only on a quarterly basis, which can lead to significant blind spots, especially since assets are ephemeral. For example, a container can be spun up and down after doing something non-compliant in minutes. Therefore, continuous checks and alerting are key.

3. **Remediation info and real-time evidence:** Sometimes, while it can be great to have the information as a CISO, in order to make sure your dev and platform teams understand the risk of the issue, real-time evidence is needed. If remediation steps are provided, that leads to less friction in fixing the issue as the devs have a path forward on what to do.

While these three challenges can seem daunting, the Uptycs platform streamlines visibility into key compliance risks and provides real-time evidence and remediation steps that are continuous, actionable, and insightful. Let's see an example of this in action!

Step 1: Assessing the State of the System

When you land on the compliance dashboard, you get a single pane of glass view into the different standards being run across your cluster fleet. Any cluster you enroll, whether EKS, GKE, AKS, or even ones you manage yourself, can undergo the same compliance scanning process. Compliance scanning is done every 6 hours and is configurable. This gives the user a good understanding that their assets are being continuously monitored for compliance issues and an overall sense of the state of their system at any present moment.

In the example below, we see that roughly 40% of our assets are compliant when it comes to NSA Hardening, and 60% are non-compliant. We can see that 63 rules passed while 101 failed, and below, we get a breakdown across the different NSA Hardening categories of our pass/fail rate.

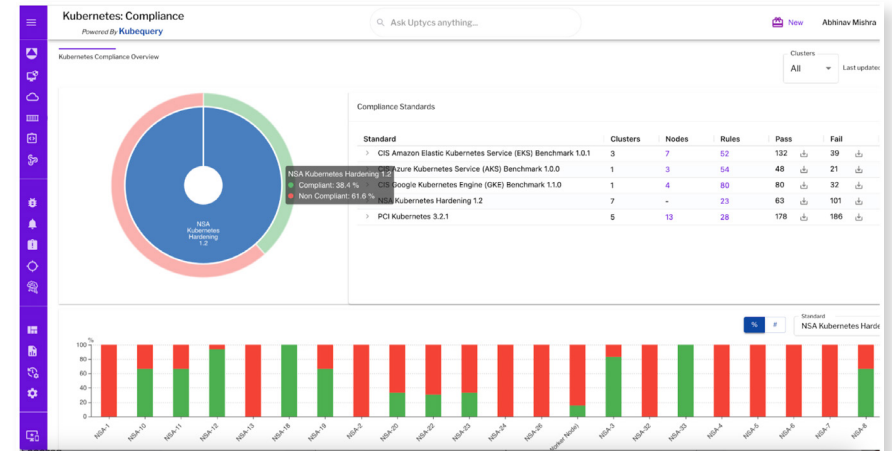


Figure 3-1: Uptycs Kubernetes Compliance

Next, let's look at what actual rules are failing so we can appropriately triage.

Step 2: Triage Through the Rules

Next, we can click the list of rules and get an overview across different categories of what is passed, what is failing, and how many assets for the given rule are passing/failing. The good thing is that just like the NSA Hardening Guide, Uptycs also logically organizes it into the relevant sections so you can focus on a specific security task that is most relevant to you instead of traversing through tables of 100s of rules.

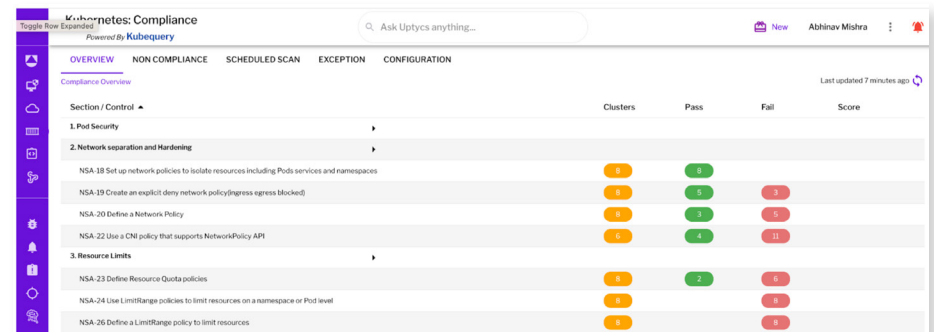
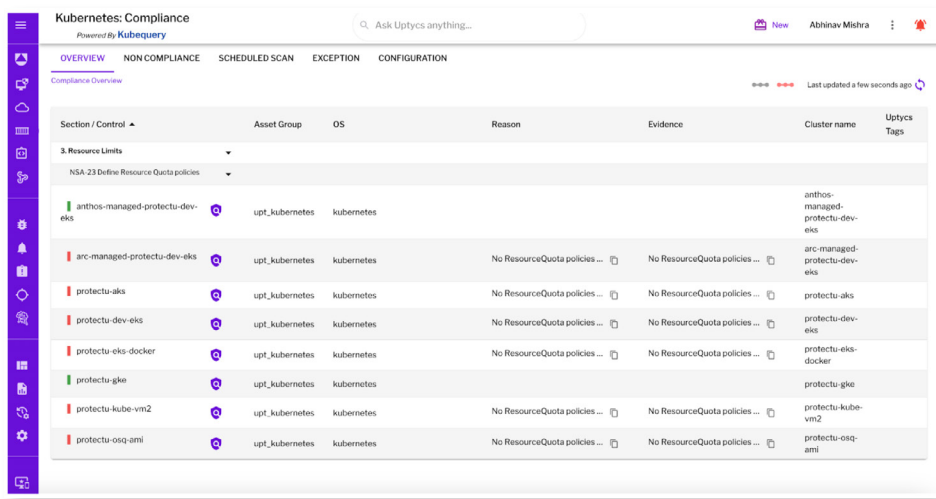


Figure 3-2: Uptycs organizes categories into relevant sections so you can focus on a specific security task that is most relevant to you.

Clicking a specific rule, for example, NSA-23 - Defining Resource Quota Policies, gives us the clusters that are passing and failing this rule. In this case, most of my clusters do not have resource quota policies defined for their namespaces. This can be problematic, especially in shared cluster environments where one user assigned to a namespace can over-inflate their namespace and limit other applications and pods from getting deployed. This can be especially problematic if you have important software add-ons for security, observability, cost metrics, and more that constantly need to be running on the cluster.



The screenshot shows the 'Kubernetes: Compliance' dashboard. The 'Compliance Overview' tab is active, displaying a table of clusters. The table has columns for Section / Control, Asset Group, OS, Reason, Evidence, Cluster name, and Uptycs Tags. The 'NSA-23 Define Resource Quota policies' rule is selected, showing a list of clusters that are non-compliant. The clusters listed are: anthos-managed-protectu-dev-eks, arc-managed-protectu-dev-eks, protectu-aks, protectu-dev-eks, protectu-eks-docker, protectu-gke, protectu-kube-vn2, and protectu-osq-ami. All clusters have a 'No ResourceQuota policies defined' reason and evidence.

Section / Control	Asset Group	OS	Reason	Evidence	Cluster name	Uptycs Tags
3. Resource Limits						
NSA-23 Define Resource Quota policies						
anthos-managed-protectu-dev-eks	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	anthos-managed-protectu-dev-eks	
arc-managed-protectu-dev-eks	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	arc-managed-protectu-dev-eks	
protectu-aks	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-aks	
protectu-dev-eks	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-dev-eks	
protectu-eks-docker	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-eks-docker	
protectu-gke	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-gke	
protectu-kube-vn2	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-kube-vn2	
protectu-osq-ami	upt_kubernetes	kubernetes	No ResourceQuota policies defined	No ResourceQuota policies defined	protectu-osq-ami	

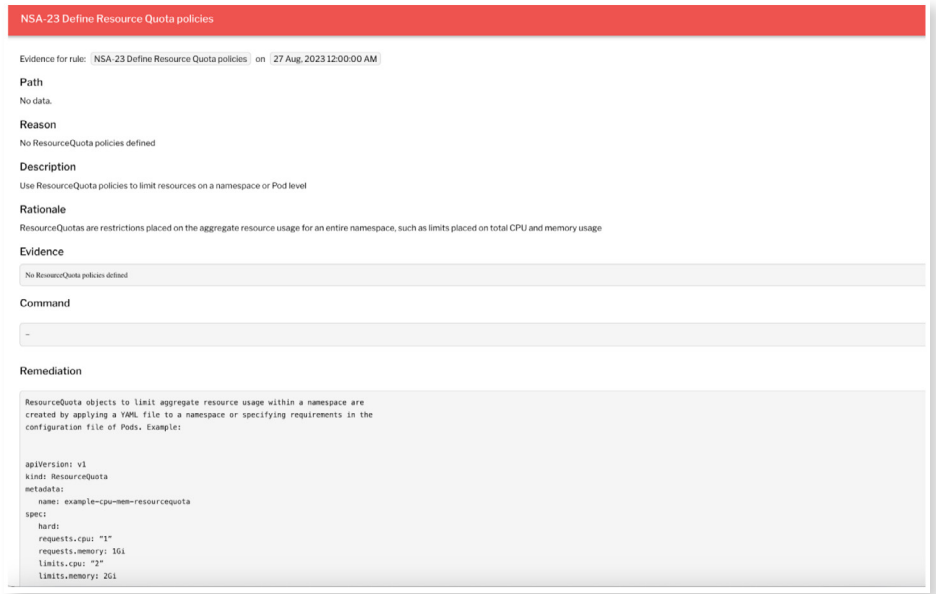
Figure 3-3: Checking a specific rule

Suppose you assign this to a DevOps or platform engineer to take a closer look at. Let's see how we can use the evidence-based view to get more details.

Step 3: Generating the Evidence and Remediation To Take Action

Many times, as a security ops engineer, you may get pushback asking why this check is so important. By clicking the purple magnifying glass, you get an evidence-based view that gives the real source of why this issue is so important, the evidence behind it, and the remediation steps. For example, for the no resource policies defined issue, you can see exact remediation steps of

how one can define a ResourceQuota with CPU and memory limits for a given namespace.



The screenshot shows the 'NSA-23 Define Resource Quota policies' evidence-based view. It includes sections for Evidence, Rationale, Command, and Remediation. The Evidence section shows 'No ResourceQuota policies defined'. The Rationale section explains that ResourceQuotas are restrictions placed on the aggregate resource usage for an entire namespace. The Command section shows a YAML snippet for defining a ResourceQuota.

NSA-23 Define Resource Quota policies

Evidence for rule: NSA-23 Define Resource Quota policies on 27 Aug, 2023 12:00:00 AM

Path
No data.

Reason
No ResourceQuota policies defined

Description
Use ResourceQuota policies to limit resources on a namespace or Pod level

Rationale
ResourceQuotas are restrictions placed on the aggregate resource usage for an entire namespace, such as limits placed on total CPU and memory usage

Evidence
No ResourceQuota policies defined

Command
-

Remediation
ResourceQuota objects to limit aggregate resource usage within a namespace are created by applying a YAML file to a namespace or specifying requirements in the configuration file of Pods. Example:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: example-cpu-mem-resourcequota
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Figure 3-4: Evidence based view

Another example below shows the importance of implementing a default network policy to prevent lateral movement inside the cluster. This is especially important because, by default, every pod can talk to one another inside a cluster, increasing your lateral surface attack in case one pod with vulnerabilities is compromised.

NSA-23 Define Resource Quota policies

Evidence for rule: NSA-23 Define Resource Quota policies on 27 Aug, 2023 12:00:00 AM

Path

No data.

Reason

No ResourceQuota policies defined

Description

Use ResourceQuota policies to limit resources on a namespace or Pod level

Rationale

ResourceQuotas are restrictions placed on the aggregate resource usage for an entire namespace, such as limits placed on total CPU and memory usage

Evidence

No ResourceQuota policies defined

Command

--

Remediation

ResourceQuota objects to limit aggregate resource usage within a namespace are created by applying a YAML file to a namespace or specifying requirements in the configuration file of Pods. Example:

apiVersion: v1
kind: ResourceQuota
metadata:
 name: example-cpu-mem-resourcequota
spec:
 hard:
 requests.cpu: "1"
 requests.memory: 1Gi
 limits.cpu: "2"
 limits.memory: 2Gi

Figure 3-5: Default network policy to prevent lateral movement inside the cluster.

Conclusion:

Uptycs provides a 3 step process of visualizing, triaging, and generating evidence to remediate that allows CISOs and security admins to be continuously compliant. This process applies to NSA Hardening and any standard such as CIS Kubernetes Benchmarks out of the box.

If you're looking to harden your infrastructure using the NSA Hardening rules, try the Uptycs platform and see how it enables you to take the great guidance the NSA has provided and make it operational and actionable for your organization.

Vulnerability Management With Shift Left Controls



When you run cloud-native workloads, vulnerabilities are always found, whether a certain vulnerable package or a known exploit. You could have thousands or even hundreds of thousands of vulnerabilities at any time. Multiply that by the number of container assets and images, and you have a big challenge of figuring out where to start.

In the previous chapter, we dove deep into how one could operationalize valuable guides such as the Kubernetes NSA Hardening Guide by leveraging the Uptycs platform for visibility across your cluster fleet, continuous monitoring and scanning, and detailed evidence and remediation steps. Now, we'll dive into vulnerability management and how to implement shift left controls to prevent any vulnerabilities, malware, or secrets from entering your SDLC and supply chain.

Top vulnerability management challenges for Kubernetes and containers

Here are some of the challenges we see when it comes to vulnerability management for Kubernetes and containers today:

- **Vulnerability prioritization and key performance indicator (KPI) based measurement:** With all the noise and scale of ephemeral workloads such as containers and pods, SecOps admins many times have to go through a lot of noise to get at the heart of the issues, such as which vulnerabilities are most critical or which ones need the most fixing based on their specific environments. Moreover, a lack of

KPIs means that vulnerability management solutions today are good with information but not good with providing actionable insights to operationalize vulnerability fixing at scale.

- **Insecure/fragmented security across the software development life cycle (SDLC):** Without a secure DevOps pipeline, customers must be more reactive in remediating vulnerabilities. No images with known vulnerabilities leaking credentials or containing malware should ever be promoted into a position where they could easily or accidentally be deployed to production systems. In addition, SecOps admins also have to use disparate tooling to catch vulnerabilities in their CI versus their registry, which can lead to blind spots and more time spent evaluating results versus taking action!
- **Lack of flexible controls to fit your developer needs:** While critical vulnerabilities may be important to be aware of, there may not be a fix available. It's also beneficial to differentiate between dev-test and production environments. While always enforcing rules in production, consider only auditing in dev-test, ensuring CI image builds aren't halted, and keeping developers unhindered.

Let's see how Uptycs unified CNAPP and XDR security platform enables you to solve these challenges while providing a single solution for all your Kubernetes/containers vulnerability management from code to cloud.

Step 1: Vulnerability prioritization and KPI measurement

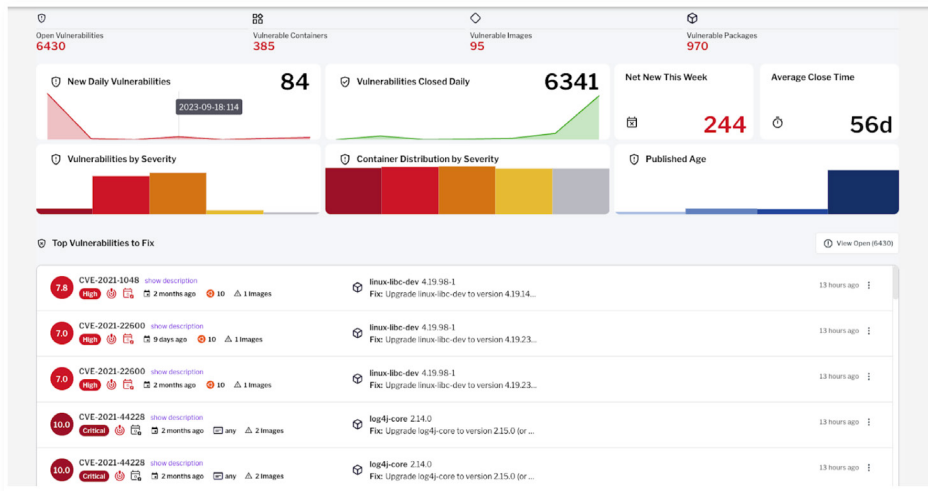


Figure 4-1: Container vulnerabilities dashboard in Uptycs platform

The Uptycs platform gives you a single place to start for anything and everything on your Kubernetes and containers vulnerability management! Specifically, the dashboard empowers SecOps admins to do the following:

- **Prioritize what is important:** Uptycs does intelligent prioritization for you by leveraging key, actionable metrics such as:
 - » Whether a fix is available or not
 - » Whether the vulnerability is exploitable or not
 - » CVSS Score
 - » Number of assets affected

By starting here you can instantly cut the noise down and start with what is most important!

- **Operationalize daily/weekly triages:** Through metrics such as what are your new daily vulnerabilities or net new this week - SecOps admins now

have an easy way to see what is most relevant now and assign a JIRA ticket or add to an exception list with one single click.

- **Continuously measure and improve:** Using metrics such as average close time, CISOs and SecOps admins can continuously track and measure their overall velocity when fixing vulnerabilities. The Uptycs platform gives you a resolved vulnerability view to measure and audit your vulnerability fixes, including specific package updates.

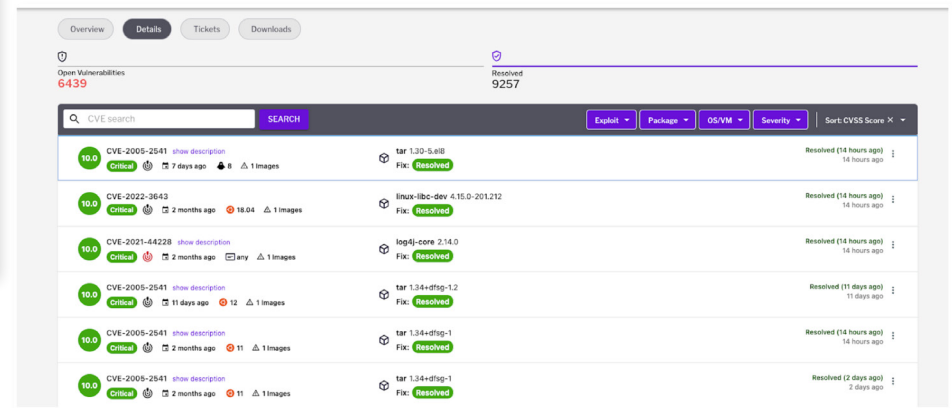


Figure 4-2: Resolved vulnerabilities shown in Uptycs platform

Step 2: Implementing shift left controls across your SDLC

Next, in order to be more proactive in fixing your vulnerabilities earlier in the SDLC, Uptycs offers CI and Registry scanning capabilities that are extremely easy to onboard, allowing you to secure your innovation pipeline from code to cloud.

Shift Left - Securing Your Innovation Pipeline

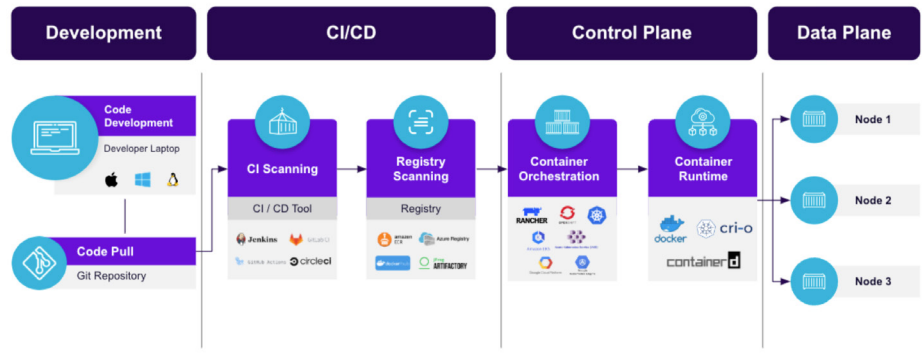


Figure 4-3: Secure your innovation pipeline from code to cloud

Capability	Supported Platforms
CI Scanning	Jenkins, GitHub, GitLab, Travis CI, AWS CodeBuild
Registry Scanning	JFrog Artifactory, Private Docker, Amazon ECR, Azure ACR, Google GCR

What sets Uptycs apart?

We can uniquely scan images across CI and Registry for vulnerabilities, malware, and secrets - ensuring your code is always at its safest.

The screenshot shows the Uptycs platform interface for CI Image scanning. It displays a table of image scans with columns for Image Name & Tags, Build Status, Scan Time, Vulnerabilities, CI Runner, Commit, and Actions.

Image Name & Tags	Build Status	Scan Time	Vulnerabilities	CI Runner	Commit	Actions
alpine alpine:3.15.0 alpine_test_image:latest	Failed	09/20/2023 15:53:51	1 11 10 0 0 0	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
redhat_os redhat_os:latest redhatubi8-minimal:latest	Failed	09/20/2023 11:56:33	2 18 40 3 1 1	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
oraclelinux oraclelinux:7 oraclelinux_7:latest	Failed	09/20/2023 11:55:39	0 1 1 0 0 0	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
amazonlinux amazonlinux:latest amazonlinux_latest:latest	Failed	09/20/2023 11:54:16	0 1 4 0 0 0	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
helloworld_image helloworld_image:latest gcr.io/google	Failed	09/20/2023 11:36:29	0 3 6 0 0 0	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
nginx_dockerfile nginx_dockerfile:latest	Failed	09/20/2023 11:35:44	6 120 138 7 11 11	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL
alpine313 alpine313:latest alpine:3.13	Failed	09/20/2023 11:28:49	0 0 1 0 0 0	Jenkins	http://192.168.133.151:8080/job/prod...	VIEW ALL

Figure 4-4: CI Image scanning in Uptycs platform

When you start with CI Scanning, the Uptycs platform shows you the security results and passed and failed image builds so that you can correlate your security findings to your SDLC.

You also get the same view with registry scanning, allowing admins to filter by a given CVE vulnerability and check on what parts of their SDLC, CI Registry, and Runtime that vulnerability is present.

The screenshot shows the Uptycs platform interface for Registry scanning. It displays a table of image scans with columns for Image Name, Image Tags, Scan Time, Image Digest, Registry Names, Vulnerabilities, and Actions.

Image Name	Image Tags	Scan Time	Image Digest	Registry Names	Vulnerabilities	Actions
aws/secret	1	09/20/2023 03:27:37	43b79f043c172c5b36a030568	jfrog-registry-protect, ecr-req	0 0 0 0 0 0	VIEW ALL
aws/upqa007	1	09/20/2023 03:27:38	0cf81bb1c9a9b5b7b7be5ac5b14	jfrog-registry-protect, ecr-req	5 13 6 0 0 8	VIEW ALL
testingqa/jetty	2	09/20/2023 03:27:38	727180af101498f1240436772501	jfrog-registry-protect, azure-i	3 65 24 0 5 5	VIEW ALL
mongo-express	1	09/20/2023 03:27:45	d5cee110909938c9c68800b93	jfrog-registry-protect, azure-i	6 26 4 0 0 1	VIEW ALL
testingqa/oraclelinux	1	09/20/2023 03:27:45	3ff548efbc53f697335d4c86373	jfrog-registry-protect, azure-i	4 0 4 0 0 0	VIEW ALL
fedora	2	09/20/2023 03:27:43	d096626ba85fe493c641a0307	jfrog-registry-protect, azure-i	0 0 0 0 0 0	VIEW ALL
testingqa/erlang	1	09/20/2023 03:27:36	2c3a44400f98f93f9d795976	jfrog-registry-protect, azure-i	47 408 792 80 58 58	VIEW ALL
kibana	1	09/20/2023 03:27:44	290e6536e48875edf13cd04758	jfrog-registry-protect, azure-i	0 0 0 0 0 0	VIEW ALL
testingqa/plone	2	09/20/2023 03:27:37	5f2195c8f779b382060147cf381	jfrog-registry-protect, azure-i	26 228 299 27 37 37	VIEW ALL
testingqa/haproxy	1	09/20/2023 03:27:47	349d1fca285f9b2721b870e398	jfrog-registry-protect, azure-i	0 2 4 0 0 0	VIEW ALL
django	1	09/20/2023 03:27:37	14a82eb5d9585b3c8000c78	jfrog-registry-protect, azure-i	190 106 146 107 94 94	VIEW ALL
testingqa/sonarqube	2	09/20/2023 03:27:43	79d73ad63b425737c1e1e96371	jfrog-registry-protect, azure-i	14 100 77 2 85 85	VIEW ALL

Figure 4-5: Registry scanning in Uptycs platform

As you start to prioritize fixes of images across your SDLC or look at your most vulnerable images, you get a single pane of glass view of your image security across the SDLC to answer key questions such as:

- Did the developer image go through the security controls across the SDLC? If not, what was missed?
- Did the image get deployed from a trusted registry? Images should only be pulled from trusted registries that are ideally being scanned for vulnerabilities.
- Did the developer bypass/not use the enterprise registry and instead deploy from registries such as DockerHub?

These insights can be key to ensuring your development teams and business units follow the right process while ensuring you have end-to-end security controls across your developer pipeline.

Image Name	Image Tags	Scan Time	Image Digest	Registry Names	Vulnerabilities	Actions
aws/secrets	1	09/20/2023 03:27:37	43b79f0d43c172c5cb36a30956b	ifrog-registry-protectu, ecr-reg	0 0 0 0 0	VIEW ALL
aws/uptys007	9	09/20/2023 03:27:38	0cd81b0c9a9b0b7b7ee5ac5b14	ifrog-registry-protectu, ecr-reg	5 13 6 0 8	VIEW ALL
testingqa/jetty	2	09/20/2023 03:27:38	727180af101498f124043a772501	ifrog-registry-protectu, azure-i	3 65 24 0 5	VIEW ALL
mongo-express	1	09/20/2023 03:27:45	d5cee11090938c39c78800b93	ifrog-registry-protectu, azure-i	6 26 4 0 1	VIEW ALL
testingqa/oraclelinux	6	09/20/2023 03:27:45	3f548ebc53f697335d4c86373	ifrog-registry-protectu, azure-i	4 0 4 0 0	VIEW ALL
fedora	2	09/20/2023 03:27:43	d9965268a95fe498c641a8307	ifrog-registry-protectu, azure-i	0 0 0 0 0	VIEW ALL
testingqa/erlang	4	09/20/2023 03:27:36	2c3a144401098f93f9d7909766	ifrog-registry-protectu, azure-i	47 406 792 89 58	VIEW ALL
kibana	1	09/20/2023 03:27:44	290eb536e45875edf13dc4758	ifrog-registry-protectu, azure-i	0 0 0 0 0	VIEW ALL
testingqa/plone	2	09/20/2023 03:27:37	6f2195c8f779ba382060147c386	ifrog-registry-protectu, azure-i	26 228 299 27 37	VIEW ALL
testingqa/haproxy		09/20/2023 03:27:47	349d1fac285f8b2721b470ed98	ifrog-registry-protectu, azure-i	0 2 4 0 0	VIEW ALL
django		09/20/2023 03:27:37	14a622ebd59585bc3cb800bc78	ifrog-registry-protectu, azure-i	190 106 146 107 94	VIEW ALL
testingqa/sonarube	2	09/20/2023 03:27:43	79d73a6d3b425737ca1e696371	ifrog-registry-protectu, azure-i	14 100 77 2 55	VIEW ALL

Figure 4-6: Image Traceability Through the SDLC

Step 3: Enabling developer-friendly guardrails

While you may want to enable guardrails in runtime, a delicate balance must be achieved in the SDLC to ensure you have security controls in place while not hindering developer velocity.

In order to facilitate this, Uptycs offers key capabilities as part of the CI Scanning capabilities, such as the following:

Grace Periods: Users may configure a vulnerability “Grace Period,” which allows a vulnerability, once first detected, to remain within an image for a set amount of time. If the vulnerability remains within an image once the Grace Period has elapsed, then builds will fail until the vulnerability is no longer detected. This grants developers time to remediate the newly detected vulnerabilities without immediately interrupting their task when the vulnerability was first discovered.

Policy Configurations: What if you wanted to ignore vulnerabilities with no fixes as part of your CI Scanning? Or only fail based on a set of critical CVEs or even ignore certain vulnerabilities? As part of your Uptycs CI Scanning configuration, you can enable certain policy configurations and flags to tailor your vulnerability management to your business needs.

Configure vulnerability scanning.

vulnerabilities:

`fatal_cvss_score: 7` # Configure the maximum allowable CVSS score.

`ignore_no_fix: true` # Ignore any detected vulnerabilities for which there are no known fixes.

Image Admission Controller: With the image admission controller (Fig. 6), you can implement secure guardrails by blocking images coming from registries scanning in Uptycs and with critical or high vulnerabilities from being used in runtime. This gives SecOps admins confidence that critical or high vulnerabilities will never make it into their container and Kubernetes runtime environments while allowing developers the necessary time to fix the image vulnerabilities earlier in the SDLC.

protectum-kube-vm2 Last updated a few seconds ago

Image Failures Admission Failures Audit Failures

Image Search Sort

zookeeper	cdbd2b8de9c398e5875e3c6244579eebc11c4bb95929f5507916d98616e8a3	40 Failures	Sep 21, 2023 3:15 AM	40 Vulnerabilities	0 Secrets	1
percona	68280e1b247ffe97f98e5d10b354fd7b0843c5f547c914a6d65e78e4b9e754	2 Failures	Sep 21, 2023 3:15 AM	2 Vulnerabilities	0 Secrets	1
zookeeper	cdbd2b8de9c398e5875e3c6244579eebc11c4bb95929f5507916d98616e8a3	40 Failures	Sep 20, 2023 3:15 AM	40 Vulnerabilities	0 Secrets	1
percona	68280e1b247ffe97f98e5d10b354fd7b0843c5f547c914a6d65e78e4b9e754	2 Failures	Sep 20, 2023 3:15 AM	2 Vulnerabilities	0 Secrets	1

Figure 4-7: Image admission controller in Uptycs platform

Conclusion

Navigating vulnerabilities in Kubernetes and container environments is complex. As these vulnerabilities mount, challenges arise in prioritization, securing the SDLC, and ensuring developer flexibility. The Uptycs platform is designed to meet these challenges head-on. By offering intelligent vulnerability prioritization, seamless integration with the SDLC through CI and Registry scanning, and developer-centric guardrails, Uptycs ensures businesses can effectively manage risks while fostering innovation.



Runtime Admission Controls



Runtime Admission Controls (RACs) add a layer of protection that ensures compliance and verification after the admission controller has approved an object. By implementing appropriate rules and processes with runtime admission controls, your organization can ensure protection from any malicious or improper code execution.

Benefits of admission control

SecOps Teams want to ensure that insecure resources, be it a pod, container, or even an ingress controller, are deployed with secure defaults. Insecure and non-compliant resources or resources with insecure defaults should be blocked from deployment. At the same time, developers want to move fast and not be overly restricted when it comes to deploying simple applications, especially in dev-test environments.

The concept of **admission control** was created to enforce runtime policies. Admission controllers in a cluster provide the ability to set policies that can enable secure guardrails for deployments of Kubernetes objects, be it a cluster role binding, workload, namespace, and more. At the time of resource deployment, the Kubernetes API intercepts the request and validates it against a set of policies deployed on the cluster. If the policy is not passed, the object deployment will fail. These policies can be tailored to meet business needs and set secure defaults. For example, you can write policies around:

- **Privileged pods** - Kubernetes provides Pod Security Admission policies to be able to prevent insecure workloads such as privileged pods from being deployed. Privileged pods, if exploited, can lead to container breakout, which can lead to the exploitation of an entire host.
- **Namespace resource limits** - Each developer or business unit typically gets its own namespace in shared cluster setups. However, when these namespaces lack predefined resource limits, it often leads to competition for resources, causing issues for customers. This can lead to a security issue as an attacker can deploy and inflate a namespace to a point with a large application that can kick out other mission-critical applications on a cluster. Admission policies can check for default resource limits defined on a namespace manifest and fail namespace deployments that don't have limits and/or quotas defined.
- **Insecure ingress controller** - Network security is a key challenge in Kubernetes from a dev/test point of view. To test if things actually work, developers deploy an nginx or kong ingress controller that allows any traffic specified using a wildcard hostname. This is especially seen given the number of IaC examples that exist today. However, this can lead to public internet exposure in production environments, a crucial security risk as any malicious unauthorized user can access the cluster. Admission policies can

be written to check for specific hostname strings such as */wildcard and block appropriately.

Several open-source solutions are available to implement admission control, including OPA Gatekeeper (which employs Rego for policy definition) and Kyverno. Additionally, Kubernetes offers built-in features like Pod Security Admission and Pod Security Standards to address critical concerns, such as preventing privilege escalations. However, some key challenges occur today when using admission controls. Let's take a look at those challenges.

Key challenges with admission control

- **Breaking production:** While there are many ways to do admission control today, DevOps teams in many situations are hamstrung in enforcing admission control in production environments because it can break production applications if an incorrect policy prevents an object from being blocked. In those situations, developers may not be well equipped to understand the consequences of violating those policies, especially if those policies are primarily configured only by SecOps teams. Solutions, such as OPA Gatekeeper, can do dry runs or audits rather than enforcement to understand what policies would break without preventing actual deployment.
- **Lack of integration with DevSecOps tooling:** While customers may have tooling for CI or registry scanning for vulnerabilities, malware, and secrets, most admission controllers today don't look at the data that comes from the scanning results to make an informed decision on whether to allow or prevent a container image from being used for container deployment. The siloed decision-making process results in additional time spent correlating data and cautiously managing basic runtime deployments.
- **Increased cognitive load for developers:** Policy controls, much like vulnerability management, are essential. However, developers need to easily identify the most relevant policy failures. Simple developer-friendly tooling that allows engineers to triage what audit failures are most relevant to their specific namespaces and pods in a single UI is key to allowing developers to remediate issues faster.

- **Inconsistent policy enforcement across cluster fleet:** - Management of policies at cluster fleet-level scale can be an extreme challenge, especially when there is a lack of visibility in which policies are applied where and duplication. Customers need single-pane-of-glass visibility to see which policies are enforced and to categorize the most important issues across different types of Kubernetes resources so the appropriate engineers get visibility into what is most relevant for their work.
- **Aligning/deciding criteria for admission:** Deciding the right balance of controls between SecOps and engineering can be a big challenge and requires consistent communication as new requirements come in. At the same time, how do you decide what to enforce, especially to strive for consistency? The indicators of compromise or default secure configurations must be broad enough to support all your different environments.

Enforcing admission controls using Uptycs

Uptycs solves the key challenges outlined above by offering native admission control for the following:

Image deployments: Uptycs can integrate directly with registry scanning to block images with critical or high vulnerabilities found in a registry scan from being used for container deployments on a cluster. This allows SecOps teams to be confident to shift left and catch security issues earlier in the software development lifecycle (SDLC) while enabling those same guardrails using valuable information from the registry scan result for runtime deployments.

In the example below, we see image failures for the Zookeeper image because 42 critical and high vulnerabilities were found.

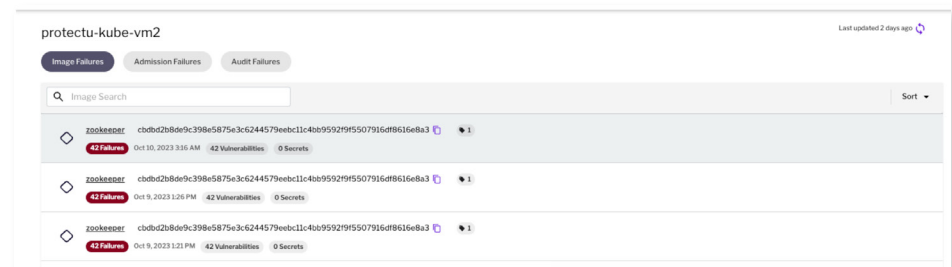


Figure 5-1: Image admission failures

Runtime deployments: Uptycs integrates with OPA Gatekeeper and acts as an actual admission controller for OPA Gatekeeper so that you don't have to deploy it separately. Customers can create their own OPA Gatekeeper policies based on business needs or use some of the turnkey policies that Uptycs provides. With these policies stored in Git, customers can use GitOps to push the policies to their cluster fleet and gain single-pane-of-glass visibility at both the policy and individual asset levels.

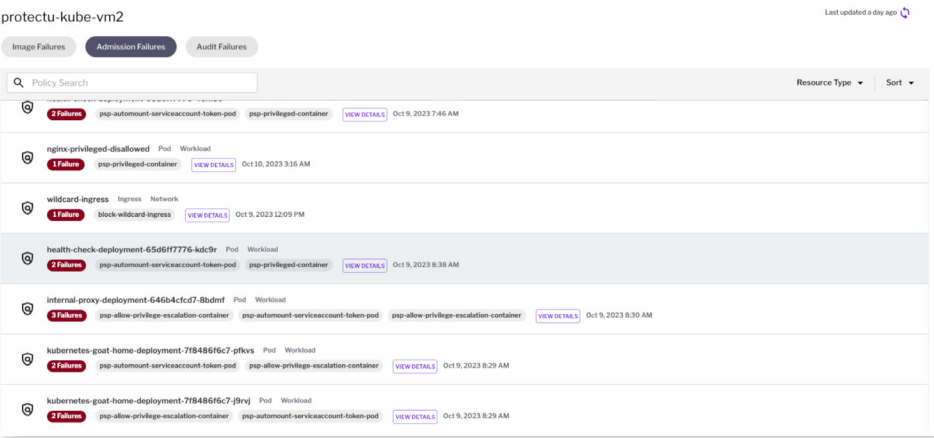


Figure 5-2: Runtime Policy admission failures

As seen above, at the policy level, SecOps teams get a single-pane-of-glass view of their admission failures (i.e., which resources failed deployment and against which policies), as well as audits (i.e., which policies are being violated the most for existing resources that are already deployed).

In the example below, we see a wildcard ingress controller that has failed deployment because it is using a wildcard host, which can be used to intercept traffic from other applications and allow any malicious attacker on the Internet to enter the cluster.



Figure 5-3: Example admission failure for ingress controller

Similarly, from a DevSecOps point of view, developers can look at audit failures for the specific namespaces they have access to rather than having to cut through all the noise across all the policies. This allows them to easily prioritize and reduce the cognitive load in understanding which policies they need to fix for their specific applications.

The example below shows that the nginxns namespace has one audit failure. The DevOps or developer engineer responsible for that namespace can click the audit failure to see what failed.

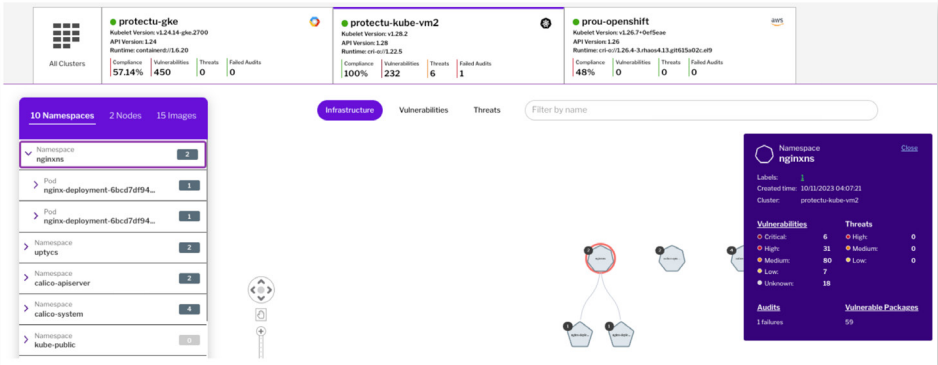


Figure 5-4: Audit visibility at the namespace level

These controls can easily be enabled as part of the Uptycs protect capabilities during onboarding or via a simple helm update!

Conclusion

As Kubernetes ecosystems continue to grow and evolve, ensuring secure defaults and robust hardening measures become paramount. Admission controls play a pivotal role in this security landscape, balancing developers' agility and the imperative for stringent security protocols.

While challenges exist, solutions like Uptycs offer a holistic approach to seamlessly integrate security throughout the development lifecycle. By harnessing the power of admission controls and integrating them with other security facets, organizations can fortify their Kubernetes deployments against potential misconfigurations and threats. As we journey ahead, embracing these tools and strategies will be key to building a resilient and efficient Kubernetes environment.



Authorization, Access, and Secrets



Kubernetes (K8s) stands as a beacon of efficiency. However, the ease of managing and deploying applications comes with security challenges, particularly in access control. Role-Based Access Control (RBAC) has been the traditional gatekeeper, defining who can access what in a K8s cluster. Yet, as cybersecurity threats evolve, the limitations of RBAC become apparent, especially when managing sensitive data or “secrets.”

RBAC operates within a predefined scope of rules which, while useful, might not suffice in the face of evolving security threats and complex, dynamic Kubernetes environments. Moreover, the management of sensitive information, known as secrets, intertwines with RBAC, adding another layer of complexity to the security narrative.

Let’s unravel why RBAC might not be enough to solve your Kubernetes access control challenges, and what you should be doing instead.

RBAC in Kubernetes

Role-Based Access Control (RBAC) is a standard mechanism for managing who has access to the system’s resources and what actions they can perform. In Kubernetes, RBAC allows administrators to define roles with specific permissions like viewing, creating, or deleting resources, and assign these roles to users or groups.

Typical configurations involve setting up roles and role bindings that tie users/groups to roles. However, common misconfigurations can occur, such as overly permissive roles, incorrect role bindings, or stale roles that remain in the system even after they are no longer needed. These misconfigurations can open up avenues for unauthorized access or actions within the Kubernetes environment, posing significant security risks.

Secrets Management in Kubernetes

Secrets in Kubernetes serve as a means to store and manage sensitive information like passwords, tokens, and keys. Managing these secrets securely is paramount to ensuring the integrity and confidentiality of the system.

RBAC plays a role in secrets management by controlling who can access or manage secrets. However, common misconfigurations like overly permissive access to secrets, or improperly secured secrets (e.g., storing secrets in plain text or version control systems), can lead to exposure of sensitive data.

Furthermore, the intertwining of RBAC with secrets management can sometimes obscure the security landscape, making it challenging to ensure both robust access control and secure secrets management.

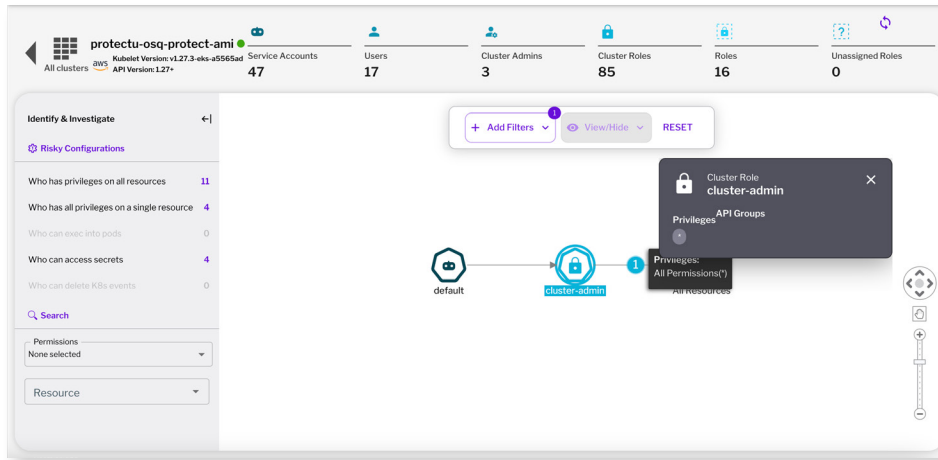


Figure 6-1: Uptycs helps identify common misconfigurations like overly permissive access to secrets, or improperly secured secrets.

As we delve deeper, we'll unravel the intricacies of this interaction and how a more holistic approach can provide a robust solution to these challenges.

Managing Authorization, Access, and Secrets in Kubernetes

Kubernetes provides a robust platform for orchestrating containerized applications, however, managing authorization, access, and secrets within K8s poses its own set of challenges. Here are three best practices to ensure a secure K8s environment.

1. Use Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a key pillar to Kubernetes security. RBAC allows you to assign roles and permissions to users, ensuring that only those with the correct access can view or modify data. By using RBAC, you can ensure that only authorized users are able to access sensitive data and resources, helping to protect your system from malicious actors.

2. Monitor Permissions and Access

This will allow you to identify any potential risks in your system and take steps to secure them before they are exploited by malicious actors. Additionally, monitoring permissions on an ongoing basis will help

you quickly identify any changes in user access that could indicate a potential breach or other security issue. This is especially important when onboarding new applications and monitoring default service accounts that may have access to all permissions by default, even if assigned a specific role binding.

Advanced security solutions like Uptycs can also be used to map real-time threats to risky RBAC configurations, giving you end-to-end visibility of your Kubernetes environment and helping you remove any blind spots in your security posture. With Uptycs, you can gain insights into which users have access to what resources, as well as detect suspicious activity across all of your Kubernetes clusters in real time. In addition you can map real-time data plane threats with RBAC control plane misconfigurations. Let's walk through an example:

One common threat profile is performing a `kubectl` command execution from inside the cluster. Typically this is insecure because the `kubeconfig` file should be access securely by a user or service account outside the cluster. If the pod is compromised in this situation, that means a malicious attacker can get access to the `kubeconfig` which can potentially give access to the rest of the cluster including other nodes and namespaces if that `kubeconfig` is not properly scoped, another common misconfiguration.

When we go into Detections in the Uptycs platform, we can see the threat highlighting the insecure Kubernetes `kubectl` access from the pod. We can see the mapping to the MITRE ATT&CK Framework as well as specific metadata for the alert that raised the detection including:

- The default service account that was used to access the pod
- Details of the pod such as what namespace and cluster it is in
- The source IP of the detection - this can be extremely useful in blocking certain malicious IPs especially when Internet access is required for some specific applications

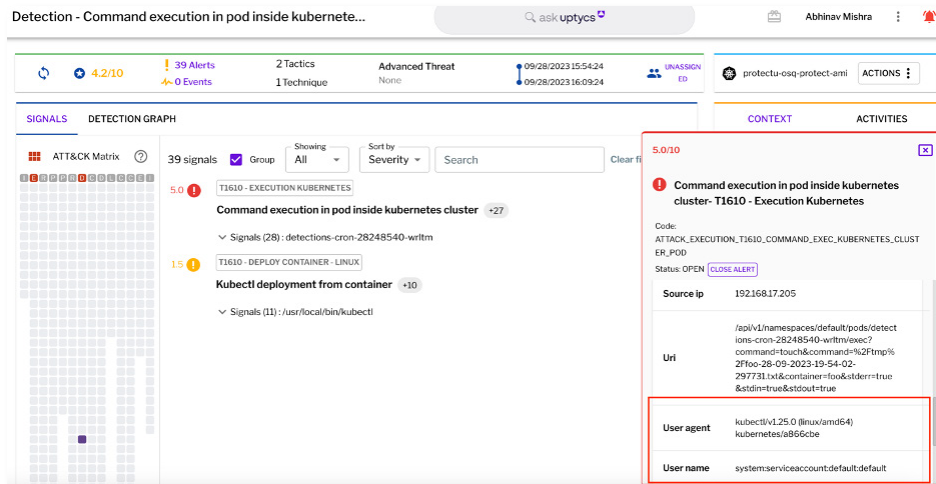


Figure 6-2: Detections in Uptycs Platform

Navigating to the detection graph we can see that the service account tried to apply a deployment. That could lead to a massive risk especially if the deployment is some insecure application!

How do we make sure this kind of access to the cluster is restricted?

In order to do that, let's take a look at the user. It seems that it was a default service account that was compromised. Default service accounts are constantly a problem in Kubernetes and are many times not addressed proactively to remove the default in case it is compromised.

Let's take a look at this default service account to see if it has excess permissions. To do that we can go to the RBAC Access Monitor for this cluster.

The access monitor gives the ability to ask questions such as who can access secrets or who can access all resources. In addition we can filter by users or service accounts. Let's filter by the default (default) service account we saw earlier.

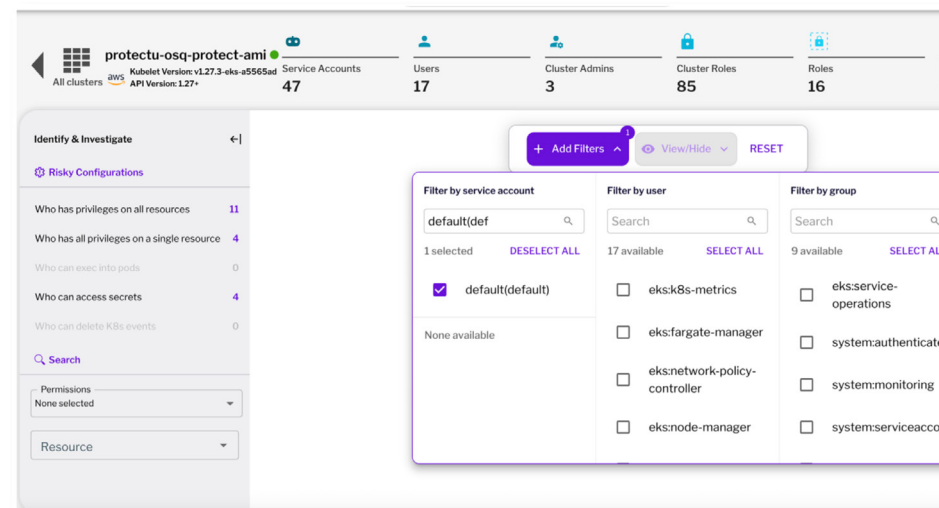


Figure 6-3: Default service account.

We notice that this user has the ability to execute anything and has all privileges across the cluster which is dangerous. That means if a malicious attacker gets access to the service account, they could delete audit logs or remove secure configurations, removing any evidence trail of the attack.

We also see that the default service account has access to pods with critical and high vulnerabilities. If any one of those pods are access with that service account, it can move laterally across the rest of the cluster.

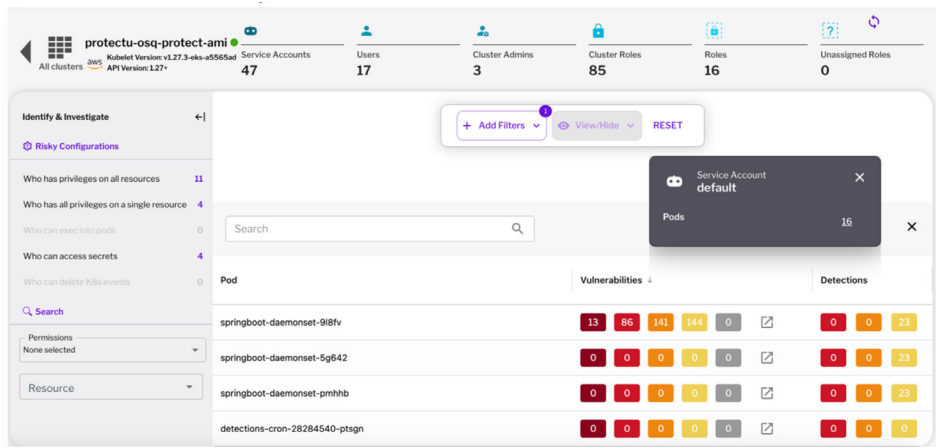


Figure 6-4: The default service account has access to pods with critical and high vulnerabilities.

Tools like Uptycs further simplify this journey by offering a unified platform equipped with the necessary capabilities to address the challenges head-on, ensuring a secure, compliant, and efficient K8s environment. As the landscape of cybersecurity continues to evolve, embracing a holistic approach to K8s security that extends beyond RBAC, and aligning with robust security solutions like Uptycs, will prove instrumental in staying ahead of threats and ensuring a resilient K8s ecosystem.

The Uptycs superpower of correlating real-time threat detections from the data plane to risky control misconfigurations in RBAC and more is key to mastering your Kubernetes security.

3. Leverage Advanced Security Solutions

Advanced security solutions like Uptycs can also be used to map real-time threats to risky RBAC configurations, giving you end-to-end visibility of your Kubernetes environment and helping you remove any blind spots in your security posture. With Uptycs, you can gain insights into which users have access to what resources, as well as detect suspicious activity across all of your Kubernetes clusters in real time.

Conclusion

Navigating authorization, access, and secrets management in Kubernetes environments can often feel like a daunting task. RBAC provides a foundational layer of control, however, the dynamic and complex nature of K8s environments calls for a more comprehensive and proactive approach to security.

Real-Time Detections and Auditing



In today's rapidly evolving digital landscape, the ability to detect cyber threats in real-time is a critical aspect of securing any IT environment. With the rise of containerization, Kubernetes has become a popular choice for managing and deploying applications across distributed infrastructure.

However, with this increased adoption of Kubernetes comes new security concerns that must be addressed. Real-time detection and auditing capabilities are essential for identifying security threats in a timely manner, effectively reducing the risk of potential security breaches in Kubernetes environments.

Traditional IT infrastructure security strategies were not designed for the dynamic, distributed nature of Kubernetes environments, making real-time detection and monitoring even more critical. Real-time detection tools provide constant monitoring and analysis of log data, network traffic, and other event data. These tools can quickly identify abnormal activity or cyber-attacks, enabling rapid mitigation of potential security incidents.

Having a single-pane-of-glass view is crucial for simplifying the management and monitoring process of Kubernetes environments. A centralized, comprehensive view of the infrastructure provides an easy way to triage and generate evidence for compliance issues. Having this visibility makes monitoring multiple instances more manageable and quickly helps identify the source of issues.

Continuous posture checks help identify vulnerabilities and misconfigurations in Kubernetes infrastructure. These checks provide insights into both existing and new infrastructure, helping teams identify and address potential security issues before they become a significant risk. Regularity in monitoring infrastructure and applications, can save any IT environment from breaches, loopholes, and errors.

The ability to provide remediation information in real-time enables prompt action to address any identified issues, minimizing the impact of security incidents on applications and infrastructure. A strong remediation process can keep an environment safe and secure before the issue escalates. Real-time information is critical to ensuring prompt action in response to cybersecurity incidents.

Real-time evidence is essential for demonstrating compliance with regulatory requirements and maintaining a strong security posture. In heavily regulated industries, compliance standards must be consistently maintained to avoid consequences. With the help of real-time evidence, it is easier to provide the regulatory authorities with the necessary logs to fulfill regulatory demands.

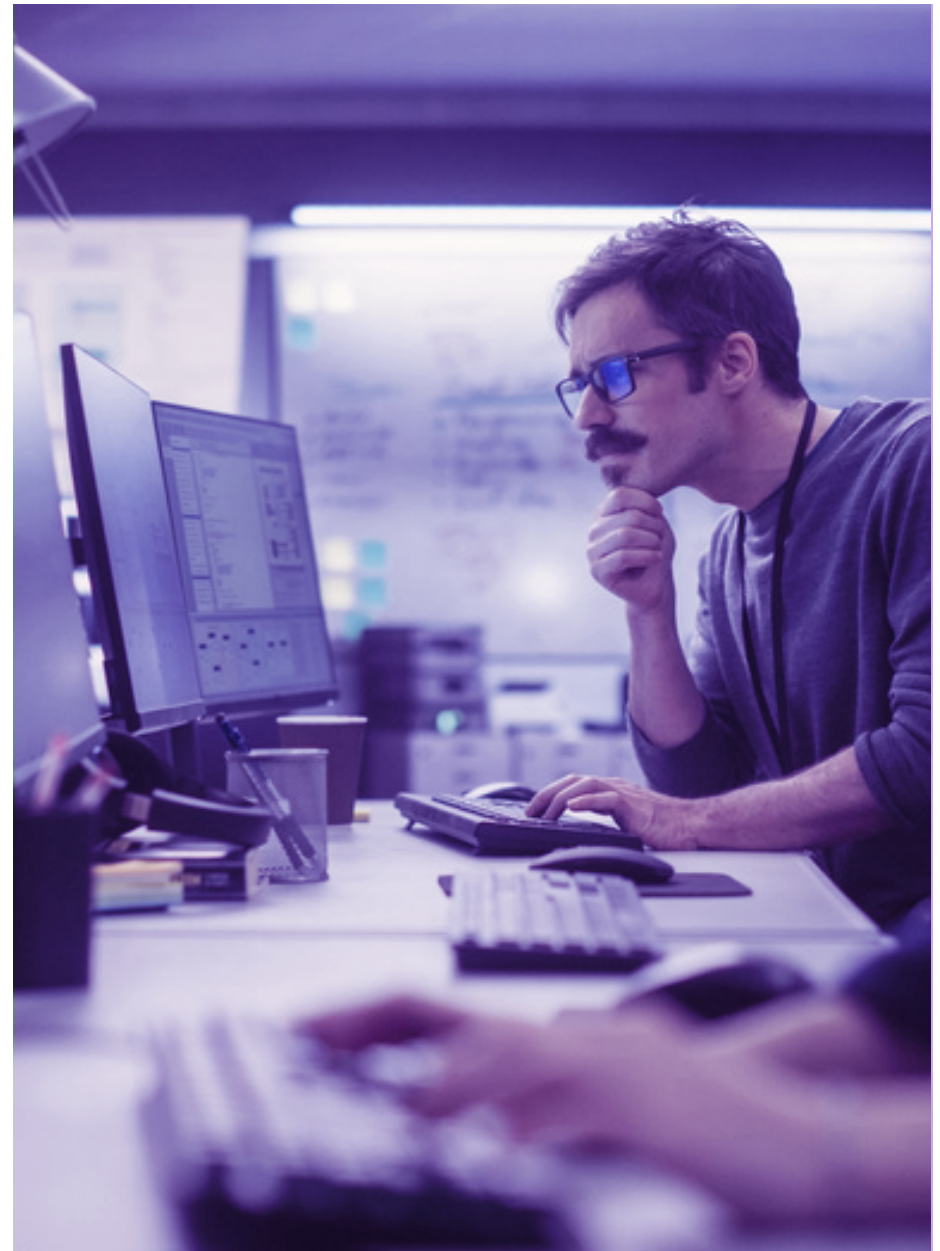
Kubernetes environments bring about new security concerns that should be taken seriously by any organization that utilizes this technology. Deployments for apps and websites should have sufficient security protocols in place. Real-time detection and auditing capabilities provide a single-pane view, continuous posture check, remediation

information, and evidence to maintain a high-security posture in Kubernetes infrastructure.

Investing in the right security tools can benefit organizations with peace in mind and assure security for their apps and infrastructure. The Uptycs' platform offers a Shift Up approach to cybersecurity, with a focus on real-time detection and auditing of Kubernetes environments. The platform provides a unified CNAPP and XDR platform, covering Kubernetes, cloud providers, cloud workloads, container runtime, SaaS and identity, endpoints, and host OS.

Uptycs' structured and normalized telemetry, data lake, and threat correlation features can help organizations perform continuous posture checks, provide remediation information, and generate real-time evidence for compliance issues.

Uptycs' Shift Up outcomes include faster and more effective threat detection and response, protection from misconfigurations and policy violations, reduction in operating costs and deployment times, reduction in the mean time to detect, respond, remediate, block, and unblock, increased collaboration across teams by removing silos, access to data and answers in minutes, not hours, and a more complete and cohesive security and compliance posture. Uptycs' platform also offers features for discovering assets, auditing and identifying issues, and responding to suspicious behavior. Users can search across live infrastructure or investigate historical states, customize threat analysis, and enable composable security controls.



Securing Your Kubernetes Environments via Network Policies



Kubernetes offers numerous benefits, and one of the most significant is its ability to rapidly deploy and connect applications across namespaces and cloud environments. However, it's crucial to address the security implications that come with this freedom of communication.

To safeguard your applications in production, the Cybersecurity and Infrastructure Security Agency (CISA) has released the Kubernetes Security Hardening Guide. This guide recommends implementing network policies to control ingress/egress traffic and enforce segmentation in your Kubernetes clusters.

But what exactly is a network policy?

In Kubernetes environments, network policies serve as a crucial tool for network isolation and traffic flow management. Think of them as firewall rules that dictate who your application can communicate with, whether it's different entities within your cluster or the internet. Additionally, network policies help define whether these rules should apply to inbound traffic (ingress) or outbound traffic (egress), and specify the scope of the policy (i.e., which pods or applications it applies to).

Enforcing network policies allows you to have granular control over the communication between pods and services, ensuring proper isolation and minimizing the attack surface. In the event of a breach, this mitigates the impact by containing it to the smallest possible extent.

Consider this example: By default, the default namespace is protected from the internet, while the test namespace lacks protection. If

the test namespace is compromised, it can laterally infiltrate even the default namespace. To prevent this, creating a network policy specifically for east-west traffic would protect app 2 and prevent unauthorized access.

In shared cluster scenarios, network policies can also enforce isolation. By assigning each team or application its own namespace, you can prevent pods and services in one namespace from communicating with those in others.

By implementing network policies, you strengthen the security posture of your Kubernetes environment while facilitating efficient application deployment.

How to Use Network Policies with Your Existing Network Infrastructure

Many Container Network Interface (CNI) providers now support network policy enforcement, including popular open source options such as Calico and Cilium. If your CNI provider doesn't support network policies, you can use CNI chaining to enforce them. In this model, the primary CNI provider handles the basic network connectivity and IP address management, while the chained CNI takes care of network policy enforcement.

Challenges and Future Outlook

Scaling Network Policies: Defining and enforcing network policies manually can be manageable for a cluster or two, but it becomes a significant task when dealing with a large fleet of clusters that require consistent policies.

Controlling Network Access: Currently, enabling role-based access controls for network policy creation and visibility is challenging. For instance, namespace owners may only have permissions to create network policies and view traffic flows within their specific namespaces.

Enforcing a Standard Zero-Trust Operating Model: Administrators need the ability to enforce a consistent security approach, such as starting with a default deny for all egress/ingress and selectively allowing traffic flow patterns across different projects.

Enabling Data Retention: Admins and developers often require historical data for debugging and validation purposes, but implementing this functionality can be cumbersome in existing setups.

How Uptycs Can help

Uptycs is a powerful platform that can provide critical visibility into your Kubernetes environment's network traffic. With its ability to monitor and analyze pod-to-pod communication, you can easily detect any unauthorized network connections or suspicious traffic patterns that could indicate security threats.

One of the major benefits of Uptycs is its integration with popular open-source solutions like Calico, Cilium, and Istio. These tools offer robust network policy management capabilities for securing Kubernetes environments, and when combined with Uptycs' monitoring and analysis features, they provide a comprehensive solution for enhancing your environment's security posture.

With Uptycs, you can easily implement and manage network policies in your Kubernetes clusters using familiar tools and workflows. This means that security teams can easily leverage their existing knowledge and skills to secure Kubernetes environments through network policies, without the need for additional training or specialized tools.

In addition to providing visibility and control over pod-to-pod communication, Uptycs also offers real-time threat detection and incident response capabilities. This allows you to not only prevent unauthorized network connections but also quickly respond to any security threats identified in your Kubernetes environment.

Overall, with Uptycs, you can improve your Kubernetes environment's overall security posture by implementing robust network policies and leveraging advanced monitoring and analysis capabilities. By taking a proactive approach to securing your Kubernetes clusters, you can minimize the risk of successful cyberattacks and ensure the integrity of your critical applications and data.



uptycs 

