sysdig

Securing Kubernetes Checklist

Kubernetes has become the de facto operating system of the cloud. It enables developers to easily package their applications into portable microservices. However, Kubernetes can be challenging to operate. DevOps teams often put off addressing security processes until they are ready to deploy code into production.

Kubernetes requires a new approach to security. Traditional tools and processes fall short of meeting cloud-native requirements by failing to provide visibility into dynamic container environments.

Fifty-four percent of containers live for five minutes or less[1], which makes investigating anomalous behavior and breaches extremely challenging.

One of the key points of cloud-native security is addressing container security risks as soon as possible. Doing it later in the development life cycle slows down the pace of cloud adoption, while raising security and compliance risks.

The **Cloud/DevOps/DevSecOps** teams are typically responsible for security and compliance as critical cloud applications move to production. This adds to their already busy schedule as they keep the cloud infrastructure and application health in good shape.

We've compiled this checklist to provide guidance on choosing your approach to security as you ramp up the use of containers and Kubernetes.

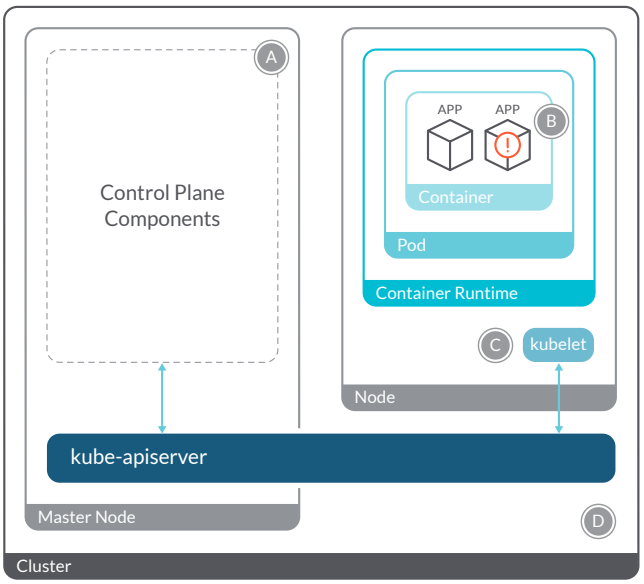
[1] https://www.zdnet.com/article/technology-containers-short-lifespans-are-getting-even-shorter/

Breaking down Kubernetes security risk

Let's first take a glance at a Kubernetes cluster to understand which elements you need to protect.

The first area to protect is your **applications and libraries**. Vulnerabilities in your base OS images for your applications can be exploited to steal data, crash your servers or scale privileges. Another component you need to secure are third-party libraries. Often, attackers won't bother to search for vulnerabilities in your code because it's easier to use known exploits in your applications libraries.

The next area is the **Kubernetes control plane** - your cluster brain. Programs like the controller manager, etcd, or kubelet can be accessed via the **Kubernetes API**. An attacker with access to the API could completely stop your server, deploy malicious containers, or delete your entire cluster.



A Access via Kubernetes API Proxy etcd API
B Exploit vulnerability in apps or 3rd party libraries
C Access via API
D Access to the servers or virtual machines

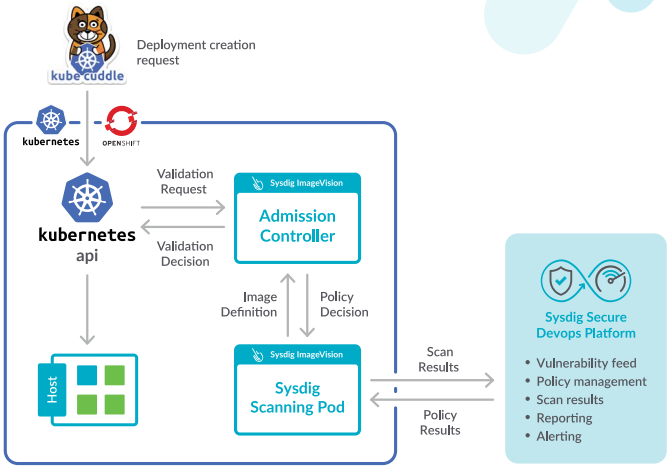
Additionally, your cluster runs on servers, so **access** to them needs to be protected. Undesired access to these servers, or the virtual machines where the nodes run, will enable an attacker to have access to all of your resources and the ability to create serious security exposures.

Now that we know what to secure, let's get into the details and review the framework for approaching Kubernetes Security:

1

SECURING KUBERNETES CHECKLIST

Threat Prevention with Admission Controllers



What is it: **Kubernetes admission controllers** are pieces of code that intercept Kubernetes API calls before the objects are created. They can be seen as a gatekeeper that intercepts API requests and enforces what can run on the cluster.

CI/CD image scanning is a critical requirement when implementing container and Kubernetes security. But developers might sometimes bypass the CI/CD pipeline and deploy an image directly to the cluster. You can integrate an **admission controller** with a scanning engine to prevent risky images from being deployed if they don't meet your security and compliance requirements.

Benefits: When **image scanning** is used with the **admission controller**, you can block threats before they reach production. Immediately trigger a scan for every image that is trying to be deployed in the cluster. You can also use additional environment context when defining admission criteria, such as namespace, pod metadata, etc. By triggering image scanning via the admission controller you can:

- Check your application, its libraries and other files for well-known vulnerabilities.
- Analyze the metadata to detect misconfigurations like exposed insecure ports, running as privileged (root) user, or exposed credentials.
- Define custom checks, like package blacklisting or detecting wrong file permissions.

If these security policies are not met, you can block the image from reaching production and notify your developers to fix the issues.

Approach: Enable the **Kubernetes admission controller** and integrate with a scanning engine to prevent risky or unscanned images from being deployed.

What is it: **PodSecurityPolicy (PSP)** is a cluster-level resource that controls the actions a pod can do or what resources it can access, and can be used to implement least privilege access for pods.

Benefits: **PSP** can prevent threats without impacting performance at runtime by enforcing least privilege access for pods in your clusters. You can enforce preventative controls such as disallowing running privileged containers, restricting resources, or limiting access to volumes at this level.

Approach: **PodSecurityPolicy** is implemented as an optional (but recommended) admission controller.

2

SECURING KUBERNETES CHECKLIST

Securing the Kubernetes Control Plane

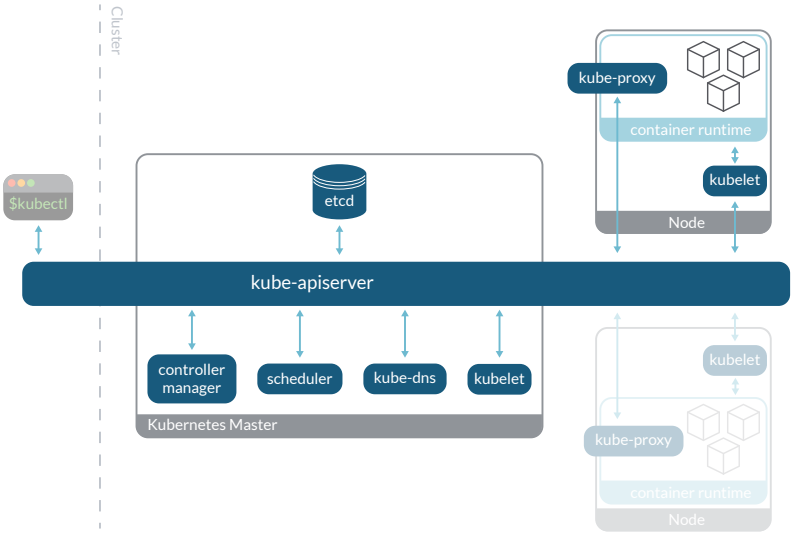
What is it: The Kubernetes control plane is the brain of your Kubernetes cluster. It manages all of your cluster resources, can schedule new pods, and can read all of the secrets stored in the cluster.

Benefits: The control plane controls your cluster; securing it will prevent a malicious user from extracting information, crashing your infrastructure or scheduling pods with access to the parent node.

Approach: Isolate the cluster network, secure the API and audit kubectl commands.

Control plane components communicate via the Kubernetes API, and kubectl instructions also translate into API calls. To secure it:

- Check the **kubelet** config: Disable `anonymous-auth`, set a `client-ca-file`, ensure `authorization-mode` delegates to the API server, and disable the `read-only-port`.
- Enable `NodeRestriction` in your API so kubelets are only allowed to perform modifications in their own node.
- Enable authorization via RBAC.



SECURING KUBERNETES CHECKLIST

Securing Workloads at Runtime

What is it: Managing security risk at runtime in containers and Kubernetes environments. **Runtime security** detects abnormal behavior that could indicate a container has been compromised.

Benefits: Flag owners and respond quickly to newly discovered **vulnerabilities** before they are exploited. Detect and remediate **attacks** when they happen, before they cause major damage. Protect from software **bugs or misconfigurations** that cause erratic behavior and resource leaking.

Approach: Scan continuously so you can detect issues as soon as possible. Also, place automatic incident responses so action can occur right away. Finally, capture forensic data when an incident happens so you can investigate the root cause and prevent it from happening again. Let's expand a bit on each of those strategies.

Runtime vulnerability reporting: After an image is initially scanned, new vulnerabilities may be found or your policies may change. You need to keep scanning your images to ensure that they're secure over time. Some image scanners would require you to do a full re-scan each time, others will save the metadata and will be able to warn you of new issues without a new scan. You need to be able to map critical vulnerabilities (e.g., CVE's with a fix available in images that are running longer than 30 days) to specific applications and identify teams responsible to fix them. This requires mapping CVE's back to the Kubernetes asset landscape (specific namespaces, deployments, clusters, pods, etc.).

Abnormal behavior detection: Is your container doing what it's supposed to do? Is it accessing files it shouldn't? Does it have strange network connections? Did anyone spawn a terminal shell? By monitoring your container's activity, you can detect abnormal behavior.

You'll need **instrumentation** to detect these issues. Does your instrumentation cover just your apps, or also the system calls? The more data you have, the more behaviors you'll be able to detect. How many **resources** does your instrumentation need? Some solutions will need a lot of memory, while others will tax your CPU.

Falco is the de facto Kubernetes threat detection engine; it detects unexpected application behavior and alerts on threats at runtime. Falco captures system calls using eBFP (among other sources), which provides visibility into runtime system activity with Kubernetes application context, and also makes it ready for high performance production environments.

Creating rules for all of your pods can be a time consuming task. Having a wide library of **out of the box rules** available can make a difference here. With so many images. it's easy to miss something, so being able to use **machine learning** to profile expected behaviours is a nice safety net.

Threat detection via operational security: Log services like AWS CloudTrail can enable governance, compliance, operational auditing, and risk auditing for your cloud account. With it, you can log, monitor, and retain account activity related to actions (configuration changes, events created/deleted/modified) across your entire cloud infrastructure. The out-of-the-box set of Falco rules for CloudTrail, a source of truth for operational audit, can minimize the setup effort, response time, and resources needed for runtime visibility you need to implement AWS threat detection and investigating security events.

4

SECURING KUBERNETES CHECKLIST

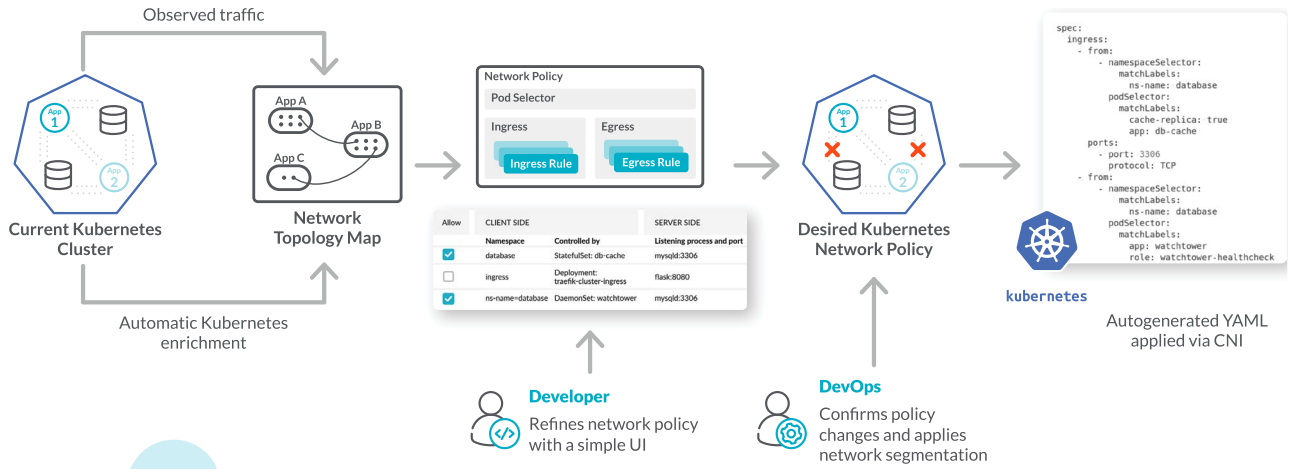
Kubernetes Native Network Segmentation

What is it: By default, Kubernetes pods are non-isolated, meaning they accept traffic from any source. Kubernetes network policies are a native security control of the platform that can be used to define how apps and services explicitly communicate with one another.

Benefits: Protect against threats like lateral movement across containers, privilege escalation, and data exfiltration. Your teams can also meet compliance requirements (NIST, PCI, etc.) that require network segmentation.

Approach:

- Monitor all network connections made between apps and services running in Kubernetes.
- Use application context and Kubernetes metadata. Use native controls of Kubernetes network policies to implement least privilege policies.
- Configure Kubernetes native Network Policies to segment and restrict traffic between, to, and from pods.
- Simplify network policy management and integrate in your policy as code framework.



5

SECURING KUBERNETES CHECKLIST

Incident Response and Forensics

What is it: When an unauthorized event occurs, you'll need a full audit trail that describes what and when it changed, and by who. You will also need to know how sensitive files were modified and who made those changes.

Benefits: You can become operationally efficient and audit-ready with your container environment. It will also help you resolve issues quickly and validate compliance requirements for PCI, NIST, SOC2 etc.

Automatic incident response: React to incidents right away before they become a bigger issue. Nothing is faster than an automatic response. Critical incidents will require you to stop the affected pods, but for other incidents, a notification is enough. Being able to notify the relevant people for further investigation through the appropriate channels is crucial.

Auditing and forensic tools: You need to capture all of the information you possibly can around an incident since, by the time you're going to investigate it, the containers may already be gone. Besides the captures, you'll need a way to browse the data so you can correlate events and find the source of the issue faster. For example, you should be able to identify unusual network activity, correlate it to shell commands executed around that time, and see what files changed.

Automated response

- Notify incidents
- Pause / Kill containers

Forensics data

- What changed?
- When?
- By Who?

Audit Tap: Track every network connection to or from a specific process, even if the connection is not successful. Capture a record of all accepted / failed net connections to identify suspicious or unusual processes.

File integrity monitoring: Gives you visibility into all of your sensitive file related activity. It's used to detect tampering of critical system files, directories, and unauthorized changes, regardless of whether the activity is a malicious attack or an unplanned operational activity.

- Bake FIM checks into your image scanning policy.
- Create Runtime Policies to monitor for Filesystem Changes.
- Implement an automated response mechanism.
- Ensure you have comprehensive forensics data.

SECURING KUBERNETES CHECKLIST

Kubernetes Security Options: DIY v/s Turnkey

Steps to Securing Kubernetes	Opensource (DIY)	Sysdig Secure (Turnkey)
Threat prevention with admission controllers	Kubernetes admission controller can integrate with a scanning engine to validate if images are vulnerability free.	Sysdig Secure embeds scanning into the CI/CD pipeline. It provides out of the box policies covering best security practices and compliance standards.
	kube-psp-advisor is a tool that makes it easier to create K8s Pod Security Policies (PSPs) from either a live K8s environment or from a single .yaml file containing a pod specification.	Prevents risky images from ever being deployed (via Kubernetes admission control). You can scan directly in the pipeline and prevent risky images from going into the registry. You get out of the box integrations and alerts with tools like Slack, SNS, PagerDuty, etc.
Securing Kubernetes control plane	Validate cluster configuration is compliant based on CIS Benchmarks for Kubernetes (kube-bench). K8-security-configwatch can review the changes in your Kubernetes config files, and highlight those that can affect the security of the cluster. Use Falco to detect unexpected Kubernetes control plane activity.	Gain deep visibility across hundreds of thousands of nodes with out-of-the-box dashboards to monitor Kubernetes control plane activity. Detect anomalous activity faster with curated Falco rules based on Kubernetes audit logs, with automatic remediation, alerting and notification integrations. Schedule continuous compliance assessments and generate reports based on CIS benchmarks for Kubernetes.

Steps to Securing Kubernetes	Opensource (DIY)	Sysdig Secure (Turnkey)
Securing workloads at runtime	Falco, the open source cloud-native runtime security project, is the de facto Kubernetes threat detection engine. Falco detects unexpected application behavior and alerts on threats at runtime.	Detect new vulnerabilities at runtime and tie the risky image to a specific namespace, cluster, deployment, pod, etc. Save time detecting anomalous activity by extending Falco with curated out of the box rules. Improve DevOps productivity by using ML-based image profiling. Gain deeper visibility into all network traffic across containers running on hybrid/multi-cloud environments
Kubernetes native network segmentation	Kubernetes Network Policies are a native resource that allow you to specify how a pod is allowed to communicate with various "entities" over the network. Network policies are implemented by a network plugin like Calico.	Reduce risk with network visibility that enables microsegmentation in minutes. Deep visibility provides guardrails for teams without Kubernetes security expertise. Implement the right policies with a unified view and shared context across teams. Simplify network policy management by automating K8s network policies.
Incident response and forensics	Sysdig is an open source Linux system exploration and troubleshooting tool for containers.	Speed up incident response with comprehensive audit trails and deep forensics data. Respond faster via auto-remediation and alerting. Validate runtime compliance with policies mapped to various compliance standards (NIST, PCI, SOC2).

Dig deeper into how Sysdig provides Kubernetes Security.

To learn more about securing Kubernetes download Kubernetes Security Guide.