sysdig

Securing Kubernetes Checklist

Kubernetes has become the de facto operating system of the cloud. This rapid success is understandable, as Kubernetes makes it easy for developers to package their applications into portable microservices. However, Kubernetes can be challenging to operate. Teams often put off addressing security processes until they are ready to deploy code into production.

Kubernetes requires a new approach to security. After all, legacy 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 to 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/

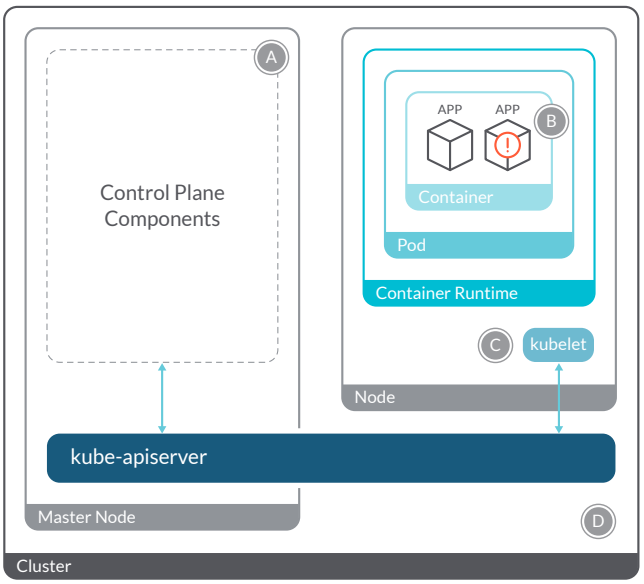
SECURING KUBERNETES CHECKLIST

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 vector 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

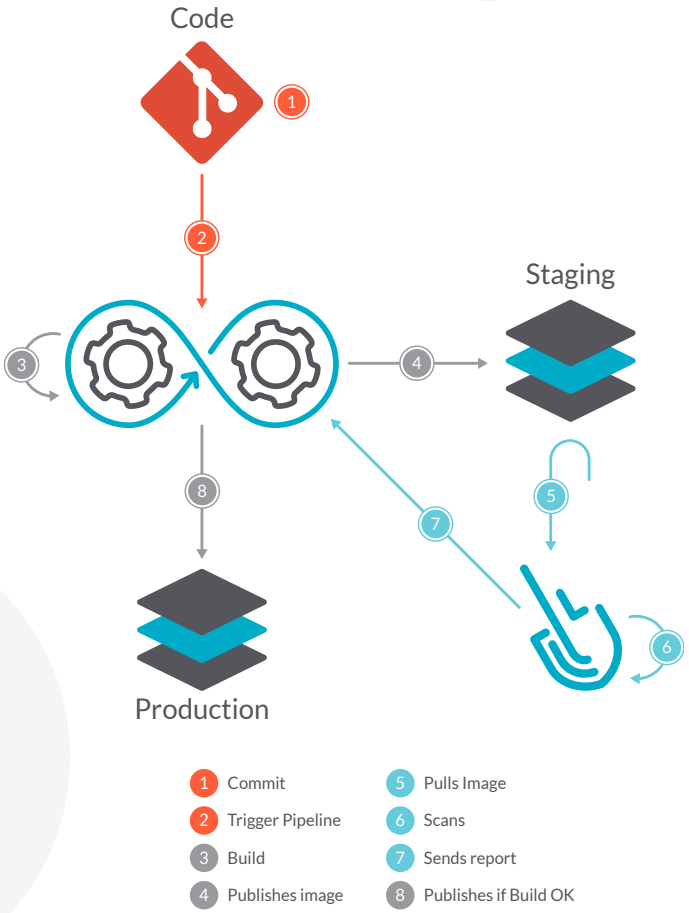
Additionally, your cluster runs on servers, so **access** to them needs to be protected. Undesired access to these servers, or the virtual machines where the nodes run, will enable an attacker to have access to all of your resources and the ability to create serious security exposures.

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

Securing container images in a CI/CD pipeline



What is it: A CI/CD automates building, testing and publication of your code. An image scanner checks the contents of your container images.

Benefits: By scanning your images in a CI/CD pipeline, you can block threats before they reach your production environment. At this level, 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.

Then, you can notify your developers to fix the issues and even block the image from reaching production.

Approach: Add an extra step in your pipeline for the image scanner after building the image, and before publishing it to the registry. Fail the pipeline if the image scanner fails, thus blocking the image from being published.

Inline image scanning is a variant, where the actual scanning is done in your pipeline. Only some metadata is sent to the backend for validation against your policies. This removes the necessity of an intermediate staging repository, saving you from accidentally leaking credentials and protecting your privacy.

Extra checks can be placed during the image life cycle. For example, enable **Kubernetes admission controller** to prevent risky or unscanned images from being deployed. Going one step further, you could perform image scanning with the admission controller as well.

To learn more about how Sysdig Secure provides image scanning in CI/CD pipelines and registries, read more here: https://sysdig.com/products/kubernetes-security/image-scanning/.

2

SECURING KUBERNETES CHECKLIST

Securing the Kubernetes Control Plane

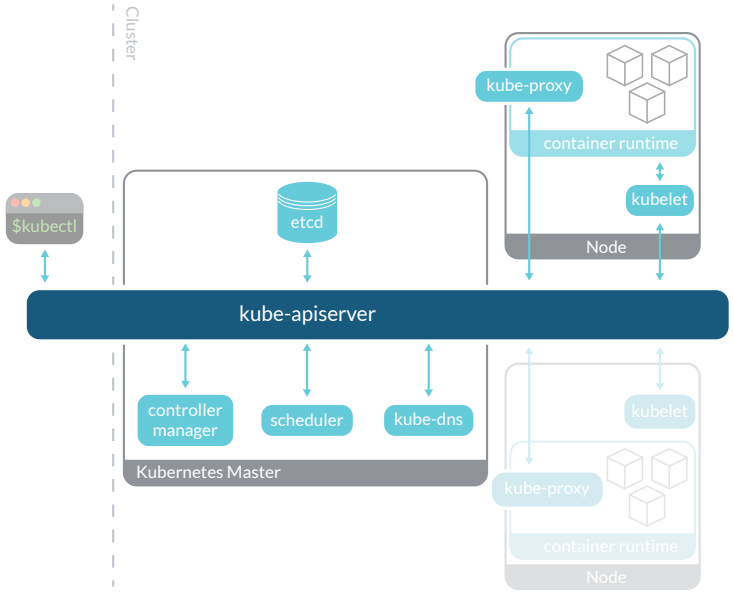
What is it: Kubernetes control plane is the brain of your Kubernetes cluster. It manages all of your cluster resources, can schedule new pods and can read all of the secrets stored in the cluster.

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.



3

SECURING KUBERNETES CHECKLIST

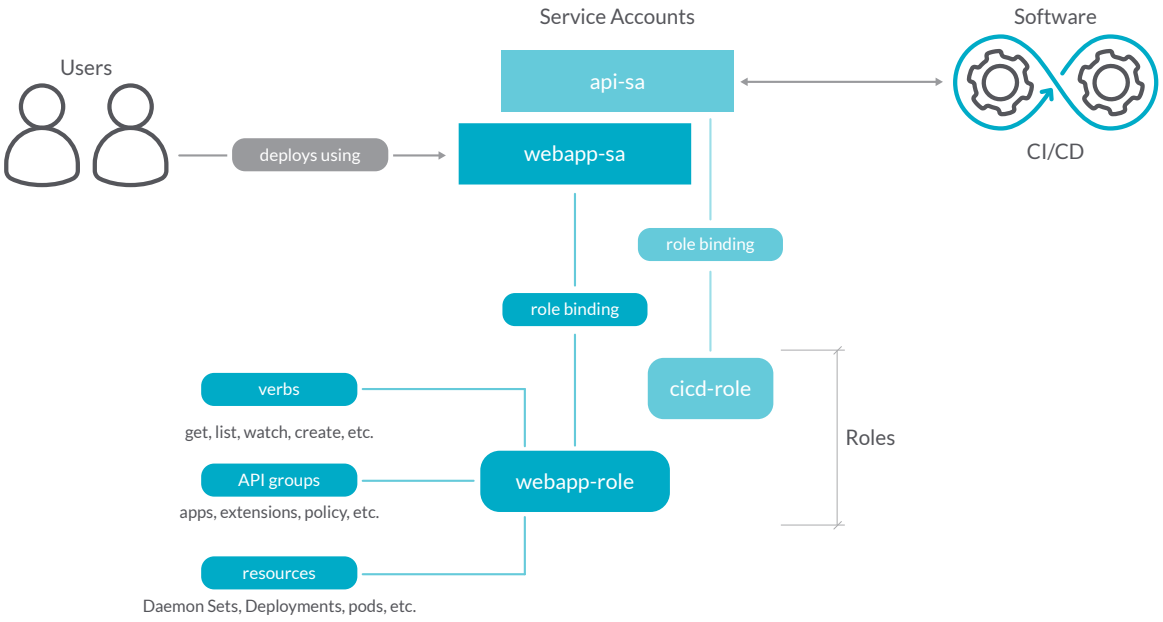
Reduce risk with Role-Based Access Control

What is it: A way to link Kubernetes subjects (e.g., human users, software, kubelets) to the actions (get, watch, list, etc.) they are allowed to perform over Kubernetes entities (e.g., pods, secrets, nodes).

Benefits: By limiting access to resources, a compromised account won't expose the whole cluster, but rather a small part.

Approach:

- Create users, accounts, roles and binding.
- Enable RBAC.
- Follow a least privileged strategy, restricting access to only the resources and actions strictly required for each role.



4

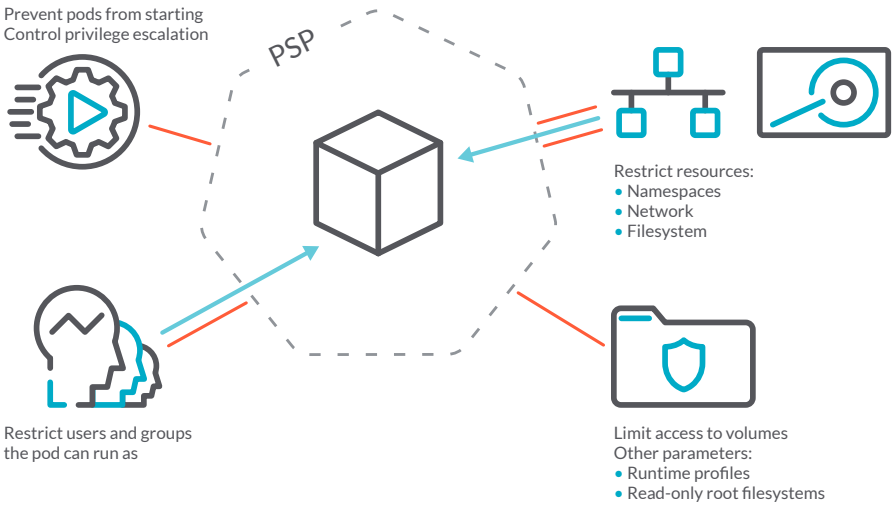
SECURING KUBERNETES CHECKLIST

Securing Pods with Pod Security Policies

What is it: A cluster-level resource that controls the actions a pod can do or what resources it can access. Policies are checked before a pod is scheduled by the PodSecurityPolicy (PSP) admission controller.

Benefits: Prevent threats without impacting performance at runtime by enforcing least privilege access for pods in your clusters. Preventative controls such as disallowing running privileged containers, restricting resources, or limiting access to volumes can be enforced at this level.

Approach: Deploy PSPs and follow a least privileged strategy.



Deploying PSPs usually involves the following:

- Apply your PSPs **before** enabling them.
- Enable the PodSecurityPolicy Admission Controller in the kubeAPI configuration.
- Recycle your Pods that are under the control of the PSPs.
- Monitor closely and fix/add PSPs as needed.

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 on it, or your policies may change. You need to keep scanning them 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 on 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.

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.

SECURING KUBERNETES CHECKLIST

Kubernetes Security Options: DIY v/s Turnkey

Steps to Securing Kubernetes	Opensource (DIY)	Sysdig Secure (Turnkey)
Securing container images in CI/CD pipelines	Open source tools like Clair, Anchore, provide image scanning and can be integrated into your CI/CD pipeline.	Sysdig Secure embeds scanning into the CI/CD pipeline. It provides out of the box policies covering best security practices and compliance standards. 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). K8s-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.
Reduce risk with role-based access control (RBAC)	Open Policy Agent (OPA) gives you a high-level declarative language to author and enforce policies across your stack. Leverage RBAC manager to simplify authorization in Kubernetes.	Sysdig Secure helps you establish federated access control across different teams within your organization. Administrators can limit the exposure of data to those who actually need it.

Steps to Securing Kubernetes	Opensource (DIY)	Sysdig Secure (Turnkey)
Securing pods with security policies	Gatekeeper is a validating webhook that enforces CRD-based policies executed by Open Policy Agent. 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.	Save time blocking runtime threats automatically generating and validating the least-privilege pod admission policy for your workloads.
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 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. Validate runtime compliance with policies mapped to various compliance standards (NIST, PCI, etc). Respond faster via auto-remediation and alerting. Speed up incident response with comprehensive audit trails and deep forensics data.

To learn more about Sysdig Secure, visit the product page.

To learn more about securing Kubernetes download Kubernetes Security Guide.