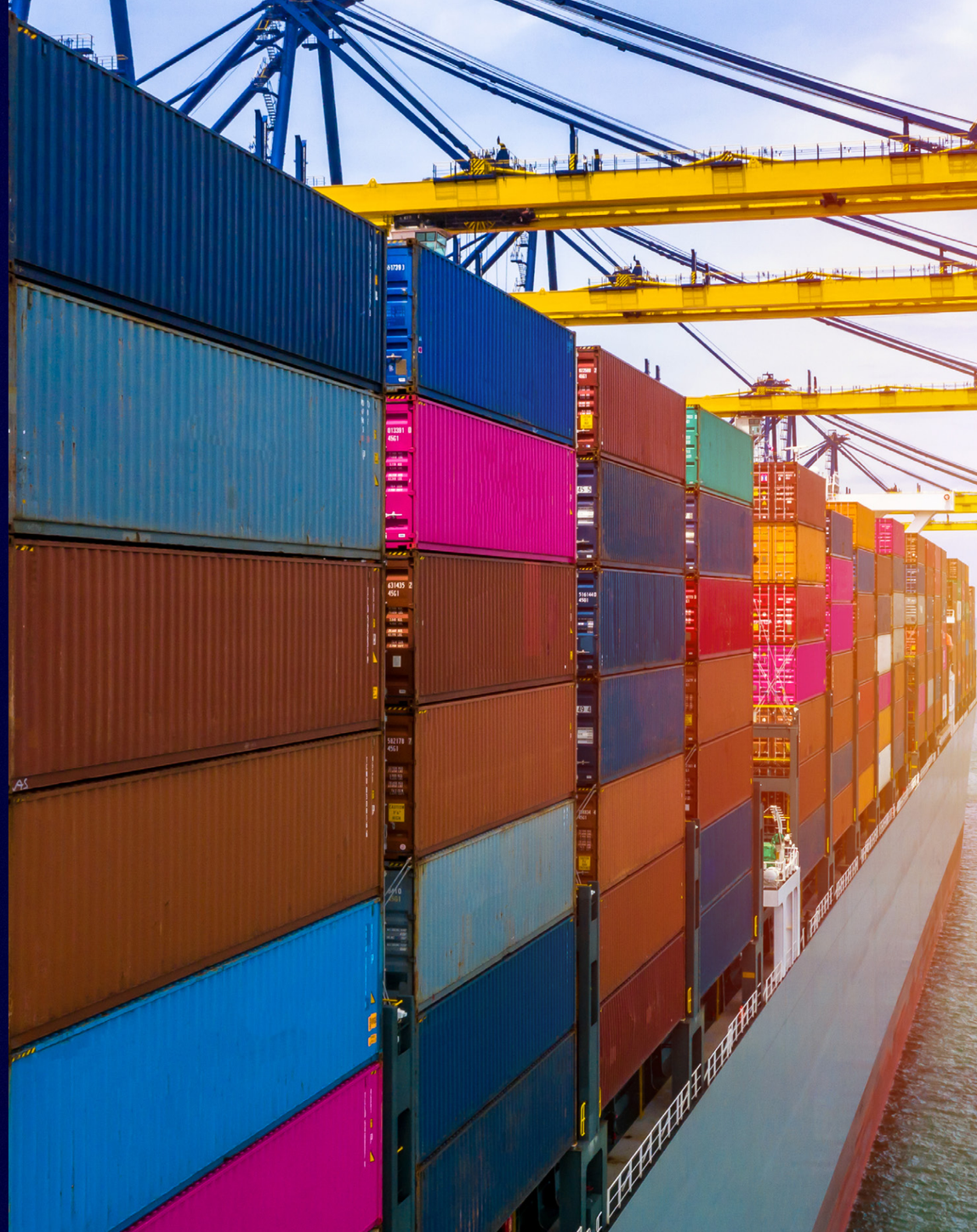
10 security best practices for Kubernetes



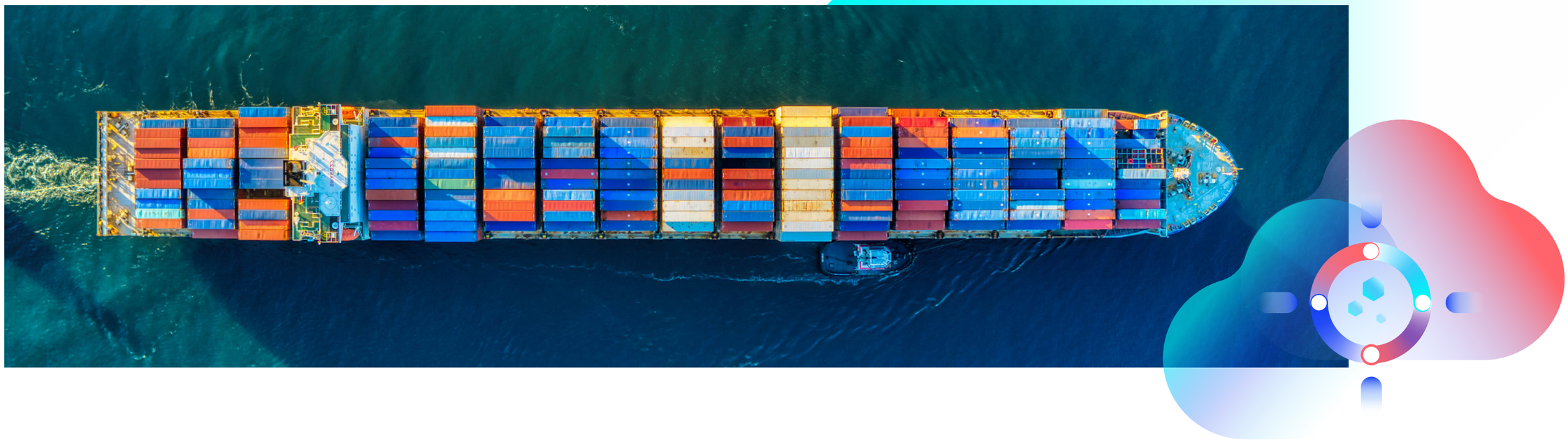
LACEWORK

Securing Kubernetes

Kubernetes, or K8s, is a critical infrastructure layer in most cloud deployments as it is the leading container orchestration and management technology. While Kubernetes and containers offer substantial operational benefits, they also need to be properly secured to prevent an external or internal cyber threat from compromising containers and the workloads within them. Protection includes being able to quickly detect exploits of vulnerabilities and misconfigurations at all layers of the K8s and container environment. The K8s and container environment also needs to be monitored and secured to comply with regulations and frameworks.

of respondents admitted to experiencing a security incident in the last 12 months.

State of Kubernetes Security Report



LACEWORK

Need security from build to runtime

It is critical to have layered Kubernetes and container defenses in place that span build to runtime. This ensures full security protection including threat detection, configuration management, and vulnerability checks across your Kubernetes environment. The size, velocity, and temporal nature of K8s environments make it challenging to secure, but implementing K8s security best practices can help. **Check out these ten best practices for securing Kubernetes and containers.**

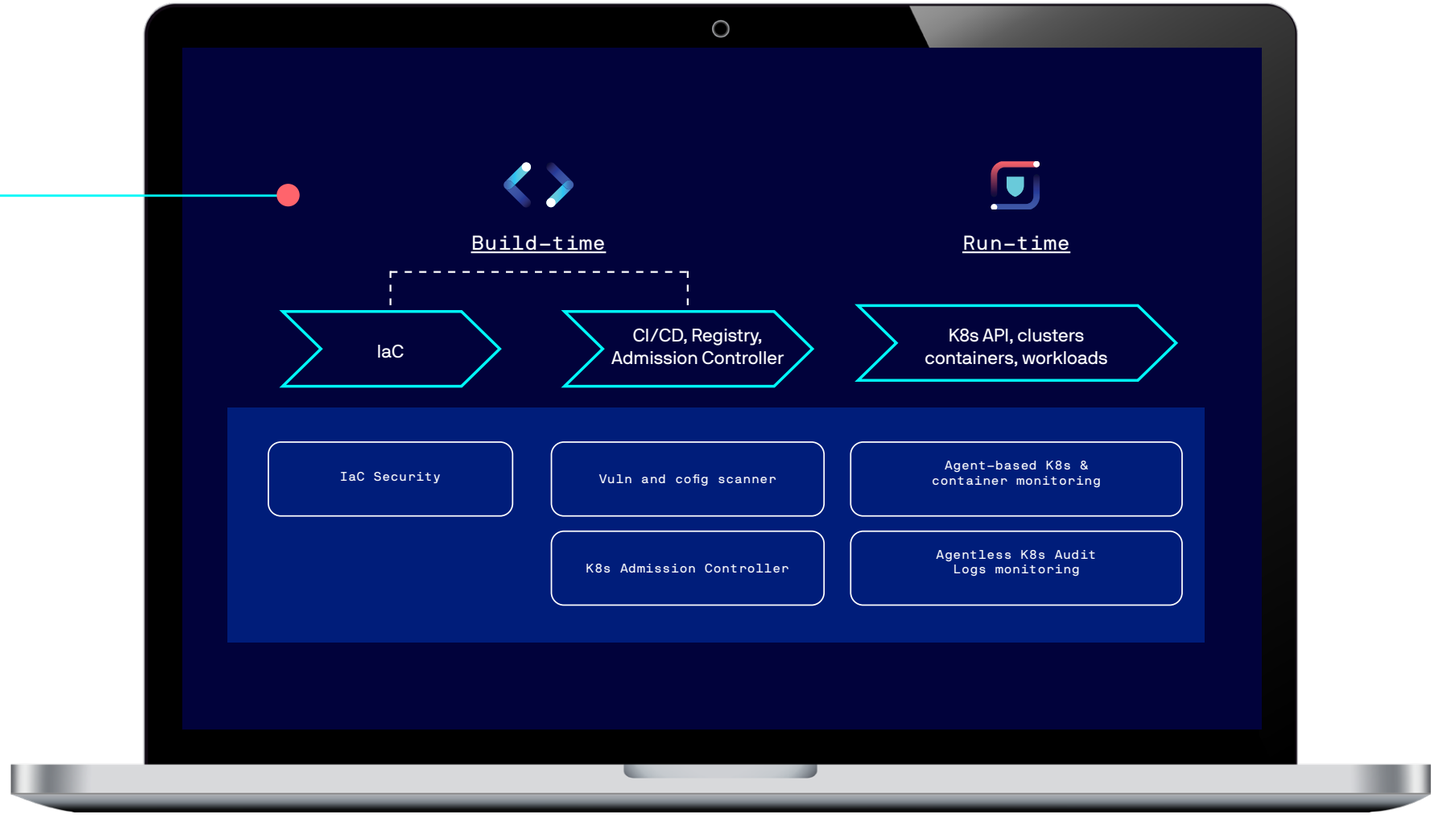


FIGURE 1 – At the top are key technologies that are part of the Kubernetes and container deployment process from build time to runtime. In the blue box are key security solutions and capabilities needed to secure these technologies.

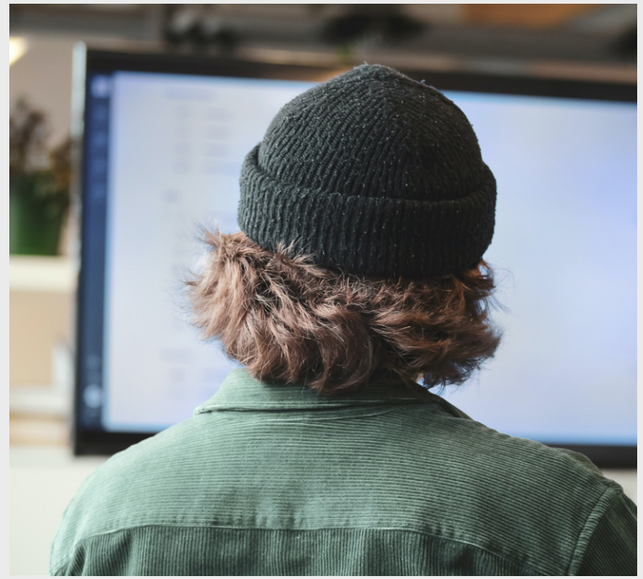
#1. Monitor and fix IaC at check-in

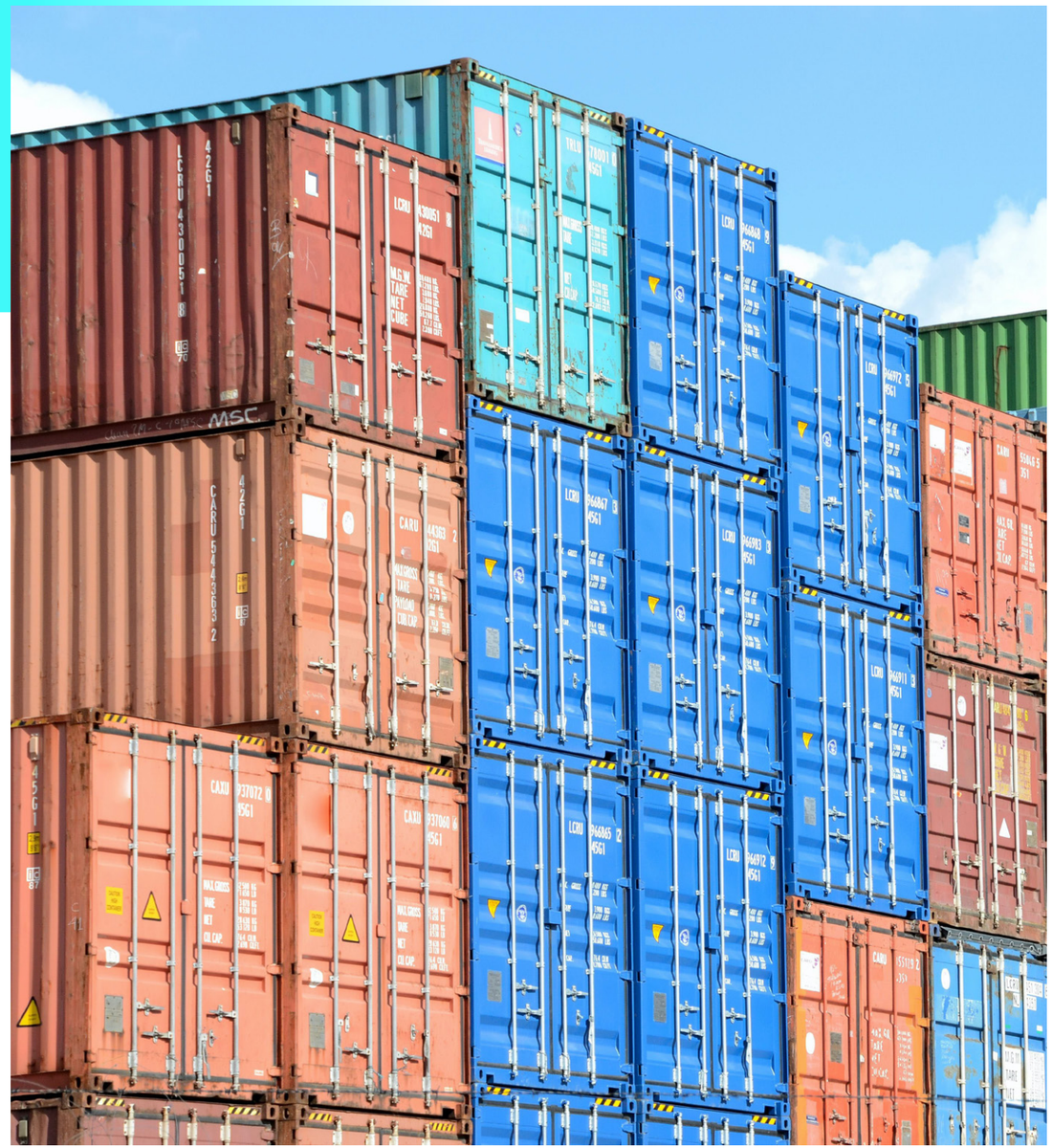
Many organizations use Infrastructure as Code (IaC) such as HashiCorp Terraform, AWS Cloud Formation, or Helm Charts (specific to Kubernetes applications and services) to configure their cloud infrastructure, which ultimately is part of the Kubernetes and container environment. While IaC offers many benefits to accelerate and simplify cloud deployments, it can also introduce security risk if the developers writing IaC are neither infrastructure nor security experts and accidentally introduce misconfigurations, or forget to add key security configurations, to the code. These misconfigurations will end up in production across possibly thousands of systems, where they are much harder to find and more costly to remediate. More importantly, a threat could exploit misconfigurations to breach an organization. Misconfigurations could include things like storage buckets exposed to the public Internet, kubelets using weak cryptographic ciphers, over-privileged Kubernetes service accounts, or a workload with too much access to the underlying node or the Kubernetes API.

What is needed is an automated IaC security solution that can inspect code as it is committed to Git-based source code repositories, scan it with hundreds of pre-built policies to check for misconfigurations or insecure code, and quickly provide violation details directly in Git. Optionally, this solution should also be able to offer one-click automated remediation and block insecure code from being checked in. As a result, developers are empowered to easily fix any issues within the interfaces and workflows they are used to. This leads to a greatly reduced chance a misconfiguration will end up in production where it can lead to a downstream breach or costly remediation.

Mistakes multiply with IaC

Would it save time if you could identify and remediate issues at the source before they were replicated with IaC deployments?





#2. Container configuration and vulnerability checks at build time

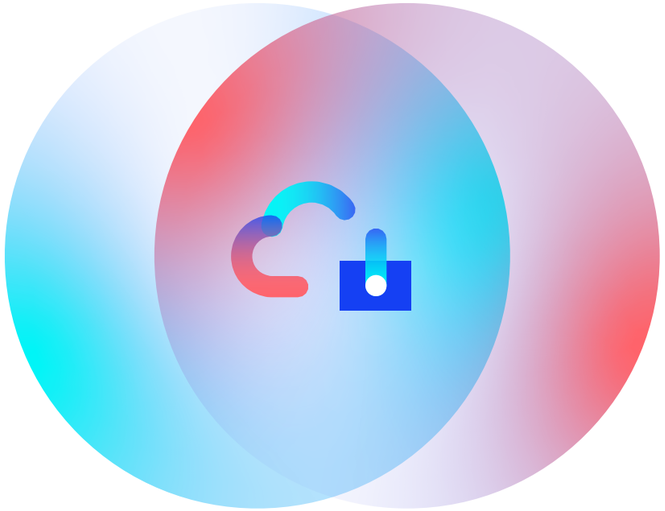
Performing security checks at build time is critical to ensure security issues do not find their way into production. This also applies to the building of containers, which is normally done within CI/CD pipelines prior to containers being checked into registries. Within CI/CD pipelines, automated and integrated security scans are needed to check containers for any misconfigurations or vulnerabilities within the base OS, packages, and libraries used by the container. Any containers failing checks should be blocked from entering a registry.

As an added layer of defense, many users of K8s also use the K8s Admission Controller as a final check before container deployment. Admission controllers evaluate requests after the K8s API server has already authenticated and authorized them. In this way, admission controllers provide an optional secondary layer of defense against requests that should not be allowed. The same security checks mentioned above in the CI/CD pipeline should also be applied here in the K8s Admissions Controller, especially because not all containers, including public containers, may go through internal CI/CD build pipelines.

#3. Strict roles-based access control

Kubernetes offers an extensive, built-in Roles-Based Access Control (RBAC) framework that lets the customer control access and permissions for the Kubernetes API. Use RBAC to limit users and groups to just the actions and tasks they may need. Always follow the principle of least privilege to ensure that users, applications, and Kubernetes service accounts have the minimal set of privileges required. Avoid default service accounts and clusterwide permissions and use namespace-specific permissions instead. Use Roles instead of ClusterRoles, and do not give anyone cluster admin privileges unless absolutely necessary. Limit access to the Kubernetes API to an RBAC Role or ClusterRole with the appropriate API request verb and desired resource on which the action can be applied, avoiding wildcards. Multi-factor Authentication (MFA) is another good option to enhance the security of authenticating to the Kubernetes API.

It's important to update RBAC policies continuously because as your policies become outdated, you may end up with users or service accounts that are over-permissioned. So ensure that whenever you create, remove, or update the permissions of any user or service account, or when you create a namespace or pods, you also modify or delete all RBAC policies associated with that entity.



Misconfigured clouds were a leading cause of breaches and can cost an average of $3.86M.

*IBM Cost of a data breach

#4. Harden and isolate nodes and OSes

A Kubernetes node is a worker node that can be a VM or physical machine that typically runs on a Linux operating system (OS). Services running on a node include the container runtime, kubelet, and kube-proxy. To secure Kubernetes nodes, you should adopt the same security strategies that you would use to reduce the attack surface and exposure to vulnerabilities for any server. Part of this is OS hardening, which includes scanning and patching the OS kernel/packages, removing extraneous applications, libraries, and other components, eliminating unnecessary user accounts, and ensuring that nothing runs as root unless strictly necessary.

Provisioning nodes with minimalist Linux distributions is a best practice and public cloud platforms usually offer pre-hardened, lightweight OSes specifically designed for containers and Kubernetes that should be considered for use. Because the OS, unlike a container, is not immutable, you should also be continuously performing vulnerability scans on them at runtime, and not just build time, to identify vulnerabilities that need patching.

There also are networking-related best practices for nodes and host OSes. It is recommended that nodes be isolated on a private network, not be accessible from the internet, and, if possible, have no direct connections between nodes. An ingress controller should be used to limit and validate user access, and only allow connections from the master node on the specified port through the network access control list (ACL). An egress gateway may be configured for access to other services outside the cluster network, to enforce network policies around encryption requirements, and to limit access to cloud services and public IPs. Network port access on nodes should be controlled via network access lists. It is also recommended to limit Secure Shell (SSH) access to the nodes. Kubernetes control and data traffic should also be isolated because otherwise both flow through the same pipe, and open access to the data plane implies open access to the control plane. For additional information on security nodes, see the NSA and CISA Kubernetes Hardening Guide and the CIS Benchmark for Kubernetes.

#5. Monitor network connections

In runtime, it is critical to map and monitor all network connections, both internal (east-west) and external (north-south). This provides an understanding of how Kubernetes workloads and namespaces interact and what external resources (e.g., cloud services, external APIs) are being accessed. This shows the entire footprint of a Kubernetes cluster and can be used for both threat detection and to meet compliance requirements that require the mapping of all internal and external network traffic.

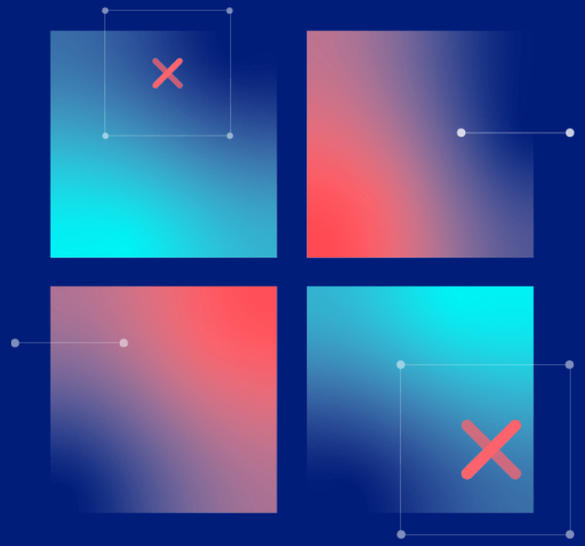
Understanding what normal network traffic looks like enables a runtime monitoring solution to then detect abnormal traffic and events, such as operational issues that lead to an increase of errors in east-west traffic or too many calls to an external API blocked by the provider. Abnormal network traffic can also be caused by a threat in the environment. An example threat might be a rogue container or compromised application performing a network or port scan to discover the environment, or attacking other internal Kubernetes or cloud services.

It is important to monitor access to any known malicious IPs and domains as indications of compromise (IoC). Many attacks toward Kubernetes have resulted in cryptominer software being installed in containers. Network connections would show connections to new IPs and domains, often already known as malicious, to download the cryptominer binary, connect to a crypto pool, and send information back to the attacker.

Kubernetes does not offer any out-of-the-box tools to show and map network connections or detect network-based threats. A best practice is to use a network monitoring security solution that can do this, including alerting on anomalous network behavior and connections to known bad IPs or domains. And for deep visibility and context, the solution should show the connections coming from the nodes and each workload, referring to their full identity (i.e., namespace, workload name, pod name) rather than internal IP address.

Visibility is elusive in complex environments

The ephemeral nature of containers makes it incredibly difficult to know what's happening in your containerized environment.



"Implementing Lacework's platform enabled us to automate our security processes to scale as we grow."

MIKE PARENT, SECURITY ENGINEERING MANAGER, DRIFT

#6. Monitor ingress endpoints

One of the top concerns for many DevOps teams is accidentally exposing an internal service to the internet. It's easy to add an external load balancer or ingress in Kubernetes, and unintentionally expose a service that lacks the proper authentication and authorization required for public endpoints. A large vehicle manufacturer accidentally exposed their Kubernetes dashboard to the internet, letting attackers create and launch new pods through the web interface.

Any addition of an external load balancer, ingress controller, nodeport, etc. Should be logged and validated. First connections from public IPs to a workload should also be raised to ensure that accidental exposure does not occur. The network map could be used to determine if any of the public services have direct access to sensitive resources such as private AWS S3 buckets, internal databases, or vaults.

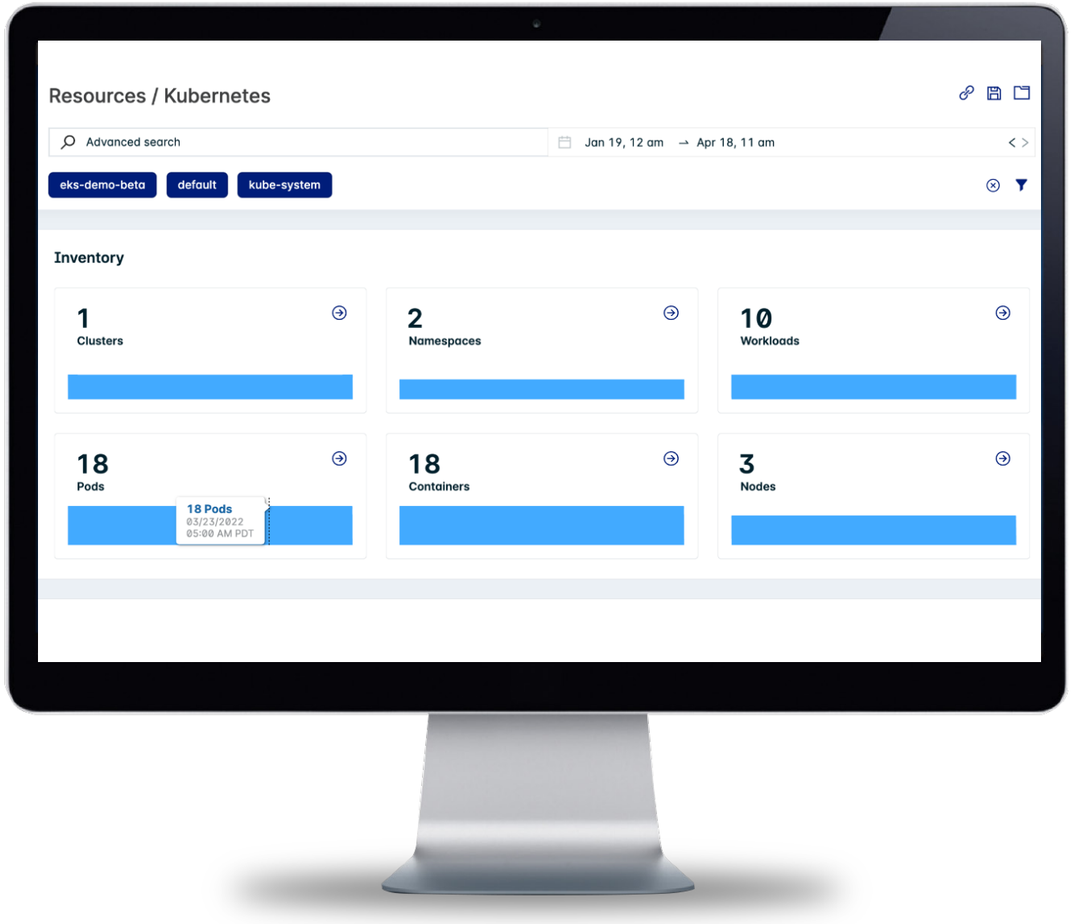
#7. Runtime, agent-based monitoring for host OS and K8s components

To gather full telemetry on your runtime Kubernetes environment, an agent-based solution that offers runtime visibility into a wide range of K8s platforms, host OSes, clusters, nodes, pods, and containers is needed. This solution needs to capture user, network, and process activity across hosts, nodes, and containers to help detect malicious activities such as container escape, data exfiltration, lateral movement, and more. The agent also needs to continuously scan for host OS vulnerabilities, and provide Host-based Intrusion Detection (HIDS) and File Integrity Monitoring (FIM) of critical files in certain directories, and offer malware detection.

All these agent-based capabilities are needed for threat detection and to meet specific compliance requirements. Only an agent can provide this; an agentless approach or periodic scans of containers are not enough as they will miss runtime threat behavior that may appear for only a few minutes before disappearing.

Lastly, the agent should be lightweight and easy to deploy via a range of options including Helm Chart, Terraform, or YAML, or with an installation process optimized for the various managed Kubernetes offerings offered by public cloud providers. The agent should support a wide range of managed Kubernetes services, container network interfaces (CNI), service meshes, and container runtime engines.

Ensure runtime visibility



#8. Monitor Kubernetes audit logs

One of the Kubernetes best practices is to keep the container immutable, and use Infrastructure as Code (IaC) to define your environment prior to deployment. This means manual activities, such as logging to a container (kubectl exec), should be minimized. Forbidden activities should include actions like the manual deletion of resources or changes to Kubernetes RBAC (roles, cluster roles, role bindings, etc.).

It's difficult to detect manual changes by only looking at a Kubernetes cluster configuration, and impossible to tie it back to a user. The solution is to enable the Kubernetes audit logs, also called Kubernetes control plane logs, that record all Kubernetes API calls including user activities and automated workflows. But these logs are very noisy. Millions of logs can be generated daily, mostly for normal, internal activities.

You need a solution that can extract the interesting events out of all this noise, and point out unusual activities such as:

- Container login methods, such as kubectl exec, kubectl attach, and kubectl log
- A network exposure of a container that can bypass any Kubernetes network policy, such as kubectl port-forward
- Any change or addition to roles, role bindings, cluster roles, or cluster role bindings
- Manual changes to a container image, including installation of packages and libraries

Changes to RBAC are difficult to track outside of the audit logs. A typical Kubernetes setup comes up with over 20 default roles and 75 cluster roles. It's nearly impossible to spot changes to these existing roles, or any unintentional or malicious binding. It's much easier to keep an audit trail of these changes in order to validate them later.

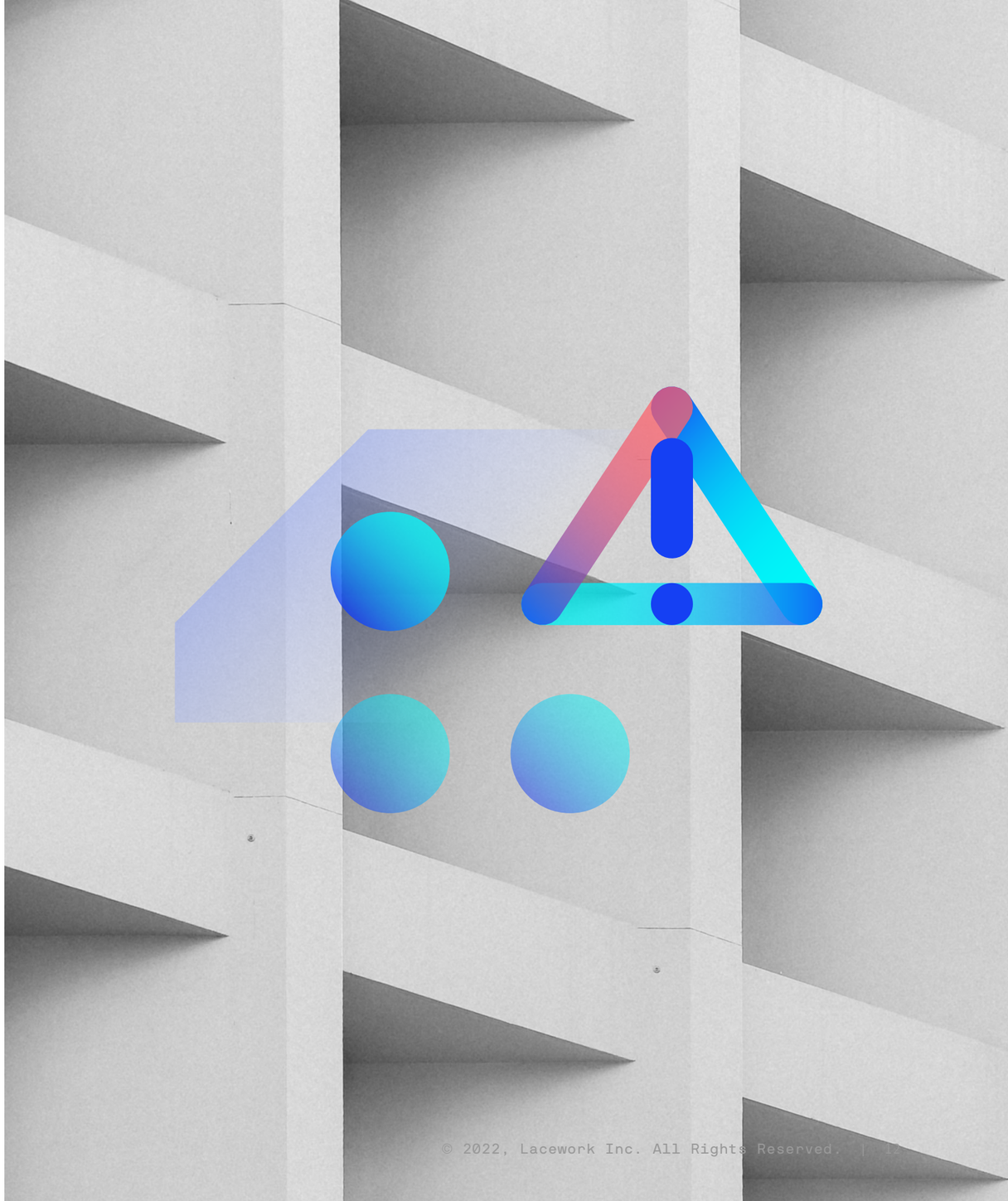
The Kubernetes audit logs also show failed API calls due to lack of authentication, missing permissions, or other types of configurations. This would detect both operational issues and security issues.

#9. Monitor deployment of new components and workloads

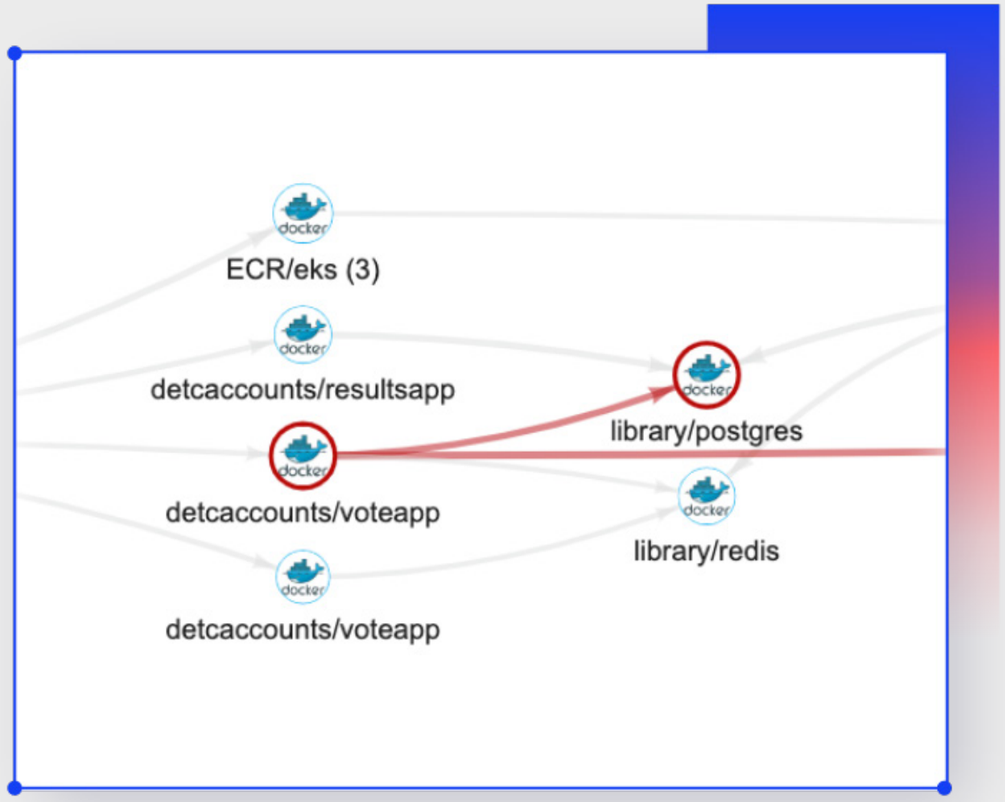
Many of the known Kubernetes breaches resulted in rogue containers, typically cryptominers, being deployed in the breached environment. These attacks were not discovered for months, showing how difficult it is to spot one malicious pod amongst thousands. The attacker can simply use a vaguely familiar name to be undetected.

Most configuration-based security solutions, such as the Pod Security Policy or Open Policy Agent (OPA) Gatekeeper, detect unsafe attributes being used by workloads: privileged, running as root, unsafe capabilities, etc. A rogue container using none of these privileged accesses in Kubernetes will not be detected.

Workload runtime monitoring needs to go deeper. It must be able to detect new registries being used, even new repositories (especially for public registries such Docker Hub and Quay). It should detect workloads created by new users, outside of the continuous deployment (CD) workflow put in place at most companies. This would also detect out-of-band deployments of workloads, containers, or clusters that may not have been validated by the regular continuous integration (CI) workflow, potentially bypassing security checks implemented earlier in the application lifecycle.



Machine learning can automatically learn the activities and behaviors that are unique to your environment and connect events to surface unexpected changes.



#10. Use automation for threat detection

Let's assume the prior best practices have been implemented and, via an agentless and agent-based approach, you are collecting all the events needed to monitor the environment across all layers: K8s, containers, and workloads. The final best practice is to leverage automated, machine-based threat detection across all these events to detect unknown or advanced threats that hide in the sea of events generated at all layers.

This technology first monitors activity in your Kubernetes and container environment to determine and baseline "normal" events like consistent connections to certain IPs over time, health checks, pod restarts, and workload rescheduling. Secondly, anomaly detection identifies deviations off of the baseline that are abnormal and represent unknown threats. Examples of this anomalous activity were covered in the prior sections. Without machine-learning based detection, manual detection rules will miss unknown threats and generate too many false positive alerts that overwhelm security teams.

Another benefit of anomaly detection is that it can detect threat actions resulting from the exploit of a zero-day vulnerability such as the exploitation of the Log4j vulnerability. In the case of successful remote code execution (RCE) exploit through Log4j, monitoring network connections would show connections to new IPs or domains, possibly some already known as malicious, to call back home, exfiltrate data, or fetch malicious content. Another example of a zero-day vulnerability is the recent "cr8escape" vulnerability in the container runtime engine of CRI-O that allows a threat to escape a container, move throughout the cluster, and perform malicious activity. Anomaly detection would alert on this sort of exploit behavior.

Lacework: comprehensive, automated Kubernetes security

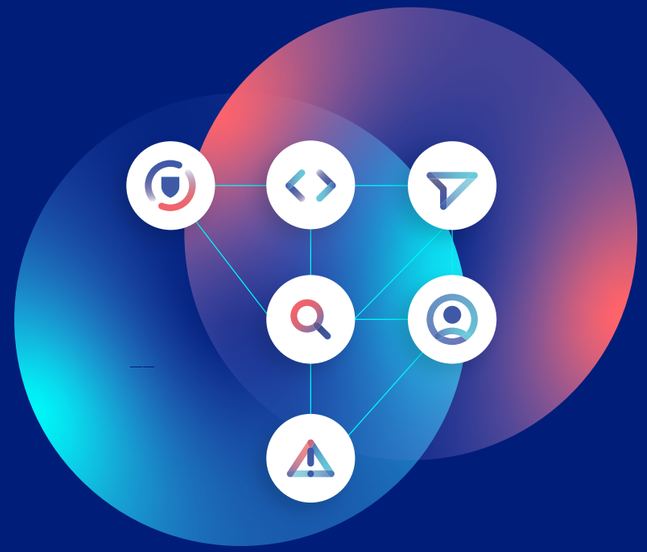
Enter Lacework, the leading cloud security platform that can help you implement these Kubernetes and container security best practices and more.

Why Lacework?

- Comprehensive, integrated end-to-end K8s coverage from a single vendor that spans build to runtime, and the control plane through the data plane
- Broad cloud security platform that protects all major cloud providers, K8s, hosts, containers, and more
- Features including threat detection, vulnerability and configuration checks, compliance reporting, and IaC security
- Includes scanning of IaC, integrations with CI/CD pipelines and K8s Admission Controller to check containers, agentless monitoring of cloud provider logs and K8s audit logs, and agent-based monitoring of hosts, clusters, and containers
- Accurate, machine learning-based threat detection with Polygraph®, and complementary policy-based and signature-based detection

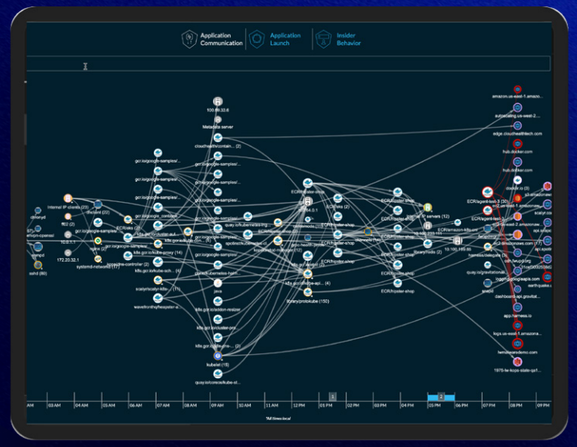
Customer outcomes

- Strong overall K8s security and compliance posture and deep visibility spanning all clouds
- Reduced costs and consolidated technology from several security vendors
- Ensures accurate threat detection, fast remediation, and no alert fatigue
- Speeds time to revenue by “shifting left” K8s security to the development process



Contact Lacework to learn more

Learn more about how Lacework helps with Kubernetes security, and contact Sales to learn more and see a live demo or discuss a trial.



Ready to chat?

Request a demo

Lacework is the data-driven security company for the cloud that delivers end-to-end visibility and automated insight into risk across cloud environments, so you can innovate with speed and safety. The Lacework Polygraph® Data Platform ingests data, analyzes behavior, and detects anomalies across an organization's Amazon Web Services (AWS), Google Cloud, Microsoft Azure, and Kubernetes environments. This patented approach significantly reduces noise and turns millions of data points into prioritized, actionable events. Customers all over the globe depend on Lacework to take software services to market faster and more securely, while consolidating overlapping security solutions into a single platform for better visibility and coverage across a multicloud environment.

Founded in 2015 and headquartered in San Jose, Calif., Lacework is backed by leading investors like Sutter Hill Ventures, Altimeter Capital, D1 Capital Partners, Tiger Global Management, Counterpoint Global (Morgan Stanley), Franklin Templeton, Durable Capital, GV, General Catalyst, XN, Coatue, Dragoneer, Liberty Global Ventures, and Snowflake Ventures, among others.

Get started at www.lacework.com

LACEWORK®