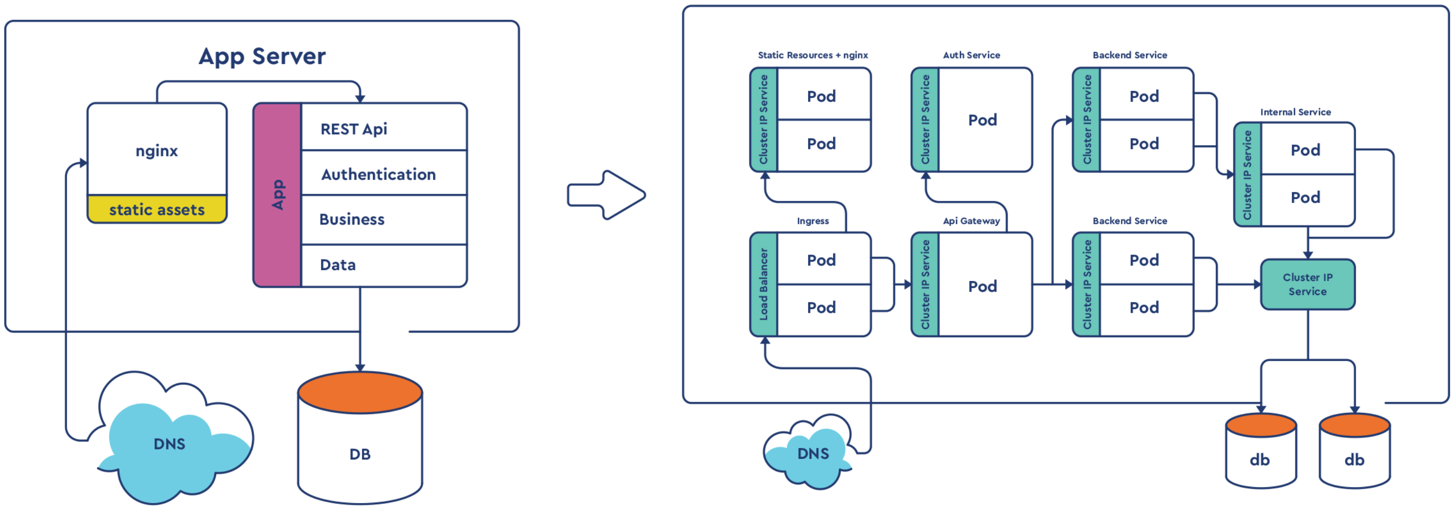
Security for Microservices: Best Practices

Aporeto

Applications are at the core of business transformation. The speed of innovation is driving ever increasing need for new development methodologies, application architecture and cloud native infrastructure. The traditional approaches to developing monolithic applications hamper feature velocity. With monoliths, application logic is wrapped into a single codebase increasing test and deployment scope for the smallest of changes. Microservice-based application development allows the disaggregation of a monolith into a set of loosely coupled and independent functions, enabling developers to work in small teams and iterate on features at a higher velocity. For DevOps teams, adoption of cloud infrastructure and cloud-native capabilities reduce the burden of managing infrastructure through automation and treating infrastructure-as-code. The adoption of cloud-managed services such as managed database and machine learning gives developers the freedom to innovate in the application business logic. Coupling microservices with adoption of cloud infrastructure and cloud managed offerings is a recipe for high-velocity application development. But, what about compliance and security considerations as enterprises undertake this transformation?

Monolith

Cloud & Microservices, Functions



Single Data Center Perimeter Security

Application In the Cloud Distributed Security

Security challenges for microservices

Moving to a microservices framework and adoption of cloud-native services introduces new security considerations for developers/line of business owners and security teams. An application that was once deployed as a monolith in a virtual machine is now deployed as a distributed system on a combination of virtual machines or containers and very likely on infrastructure outside of the traditional enterprise security perimeter. Because APIs are the communications method between microservices, new access control measures and compliance enforcement points are needed.

Two personas play a critical role in successfully enabling microservices:

The Developer

Security and data privacy requirement introduce access control and encryption requirements for developers. A few common challenges developers face to meet these requirements while focusing on building their business logic are:

1 Enabling encryption across all microservice components requires changes to applications and maintaining the associated public key infrastructure (PKI). Any patches associated with encryption libraries means re-deploying these applications.

2 Authorization for APIs needs to be implemented in a distributed manner to ensure all API requests within the Enterprise and access to APIs outside of the Enterprise are sanctioned.

3 Secrets management and key distribution for distributed applications - especially ones governed by strict compliance requirements such as PCI - introduce overhead for the developer.

The Security Practitioner

For the security practitioner maintaining visibility, compliance, and control while embracing this architectural transformation towards microservices is the primary goal. A few common challenges security teams face to achieving this goal are:

1 Dynamic nature of microservices invalidates existing network security approaches. The commonly-used ticket-driven security practices requiring corporate security teams to create new firewall rules with each release greatly hampers application deployment velocity.

2 Losing control over the network and compute infrastructure creates challenges for monitoring the threat landscape at the network and host layer.

3 Assessing compliance for containers and new application architectures driven by APIs.

Core Tenants of a Microservice Security Solution

The adoption of microservices and cloud-native infrastructure requires a comprehensive solution that operates across multiple layers of the security stack and is built on strong identity for policy enforcement. Three core tenants define an effective microservice security solution:

1

Does the solution address the entire microservices stack?

2

Is the solution based on the Zero Trust principles of authentication and authorization for all transactions?

3

Can it be deployed across heterogeneous environments?

Comprehensive

There are three layers in the microservice security stack:

I. Container runtime: interactions of the container with the host
II. Network access: network traffic between microservices
III. Application programming interface (API) layers: The unit of consumption for microservices

Focusing on any one layer results in myopic protection and detection solutions for two key reasons:

I. Detection: The ability to correlate data for anomalies across these three layers greatly improves the signal to noise ratio for detecting attacks. Signals from all three layers give us a better view into the potential kill chain an attacker will execute. For example we are able to correlate a runtime violation with running port scans or brute force attempts to determine open api's on a microservice.

II. Protection: Exercising zero trust and least privilege access control across these 3 layers creates a much more robust solution. A code injection attack followed by remote shell and data exfiltration attempt encompasses multiple layers of the security stack. Enabling access control at each layer greatly reduces the probability of a complete attack execution.

Identity Driven

Service Identity or Application Identity is central to Authentication and enforcing Authorization between users, applications and microservice APIs. The more context available to identify a microservice, the stronger it makes authentication and authorization policies. Considerations for enhancing the identity of a service are as follows:

I. Leveraging vulnerability data from container image scans. This data may change over the life of a container and so does the contextual identity of the container.

II. Containers are most often auto-generated as part of a CI/CD pipeline and carry considerable metadata, such as the type of container (frontend or backend), the type of image it is running (Mongo, Redis) and perhaps some reference identifier back to the code commit that triggered the creation of this container. This metadata can become a multi-attribute identity.

Identity paves the path for scalable encryption across all microservices. Mutual TLS is a popular encryption technique and part of the TLS negotiation process is to authenticate and authorize both ends of a microservice. The identity assigned to a microservice can now become an integral part of authorizing this TLS session. It is important to offload any Public Key Infrastructure logic from the microservice allowing developers to focus on their business logic.

Heterogeneous

Most microservice and cloud-native adoption projects start as small initiatives with specific scopes. There is always a brownfield deployment where microservices have dependencies on monoliths. The services can be hosted in both public and private cloud environments and use a number orchestration capabilities such as Kubernetes, EC2 Container Services or VMWare. A microservice security solution needs to operate in all of these environments.

Best Practices for Microservices Security:

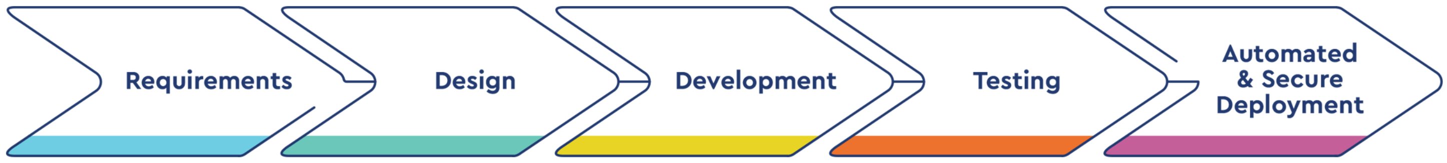
Best practices for security in a microservices environment must include technologies and processes that address all of the core protection and visibility use cases of a distributed application. While network security and container threat and vulnerability management are critical components of comprehensive microservices security, they are just part of the solution for operating securely in Zero Trust environments. APIs and identity in particular are areas of cloud-native applications that are often overlooked and underserved in most security programs. These uses cases must be addressed in a way that does not put additional burden on either the security organization or development.

Network Security/Access Control

Network security equates to enforcing access control policies and deriving network visibility through IP addresses. The adoption of public cloud infrastructure coupled with the dynamic nature of microservices means IP addresses are meaningless for policy enforcement and visibility because they are no longer persistent identifiers for an application.

A better approach is to utilize a persistent and certified application identity-driven from application metadata. Network security and visibility are now decoupled from network infrastructure and become applicable across a heterogeneous environment spanning public and private cloud. This identity-driven security model is crucial for seamless microservice elasticity, load balancing and rescheduling across hosts.

The decoupling of security from network infrastructure and tying of access control policies to application identity also enables security and DevOps teams to automate the security deployment process. The result is application security becomes part of the CI/CD deployment process removing application deployment hurdles.



In the world of microservices updates to applications or load related replication events can happen ten times in a single day - this means new containers are also deployed at that velocity. If we take an example of a security practice involving updates to IP driven policies at every enforcement point - for example, a public cloud security group/ACL or an on-premise firewall rule - the rate of change of policies is very high. Automation of changes would help but troubleshooting policy issues in the event of a failure would be extremely difficult. In cloud environments, relying on common tools such as Netflow/IPFIX for visibility will also be extremely difficult because IP addresses are dynamically assigned and can overlap. The combination of best practices outlined here remediates many of these issues and prevents security teams from being bottlenecks while maintaining velocity and not compromising on compliance or security posture.

Container Runtime: Vulnerability & Threat Detection

Adoption of containers and their inherent, immutable nature improves the scope of patch management in the event a vulnerability is discovered. Security teams need to ensure there are mechanisms in place to continuously discover vulnerabilities within containers and that alerts are generated notifying the pertinent teams of the discovery. This practice is critical hygiene that will significantly reduce the scope of potential exploits.

Monitoring container runtime events can provide valuable signals indicating vulnerabilities. Generating alerts for specific runtime events provides visibility and remediation into potential threats on the host such as privilege escalation, code injection and data exfiltration.

Monitoring the right runtime events can be done through explicit rules, but this approach is static and inflexible. An alternative approach is through the use of runtime behavior analysis of a given container and detecting any anomalies from baseline behavior. Machine learning techniques can reduce false positives in detection of anomalies but it is also very important to understand machine learning algorithms need large data sets for proper training.

Consider a scenario in which a vulnerability is discovered and it impacts containers you have deployed. Continuous vulnerability detection provides immediate visibility into impacted containers and allows your security teams to assess the impact of the vulnerability to the application and enact a remediation plan. Additional monitoring rules and runtime policies can be applied to impacted containers to ensure any potential exploits are flagged and quarantined while patch plans are being put in place. These best practices for threat and vulnerability management give newfound visibility into cloud security threats and the opportunity to apply security automation and orchestration to cloud incident response workflows.

API Access Control

Microservices are exposed through application programming interfaces (API). APIs open a new threat vector. Without the right access controls in place, APIs can be easily exploited for data exfiltration. Access to an API must be controlled through an authentication process - this is part of setting up a TLS session as most API requests utilize HTTPs - and authorization policy. When authorizing API requests, it is common to use a token - most commonly a JSON Web Token (JWT) - that carries information about the service or user making the API request. The JWT is provided to a user or microservice by an identity provider and it contains information on what the user or microservice is allowed to do. Authorization for APIs is most often defined as custom business logic in applications today, but this approach does not work well because changes in authorization policies forces changes in applications.

A better and more scalable approach is to use a centralized mechanism to define API authorization policies with distributed enforcement outside of application business logic. A change in authorization policy should not require a change in application logic.

In enterprises, security teams want to control policies tied to API access and need auditing and visibility capabilities for compliance. There are two critical steps to ensuring that developers and security are aligned:

I. Developers define microservice APIs, the dependencies these APIs have on internal or external applications, and scopes (create, update, read, delete) associated to each of the APIs.

II. The security teams define runtime policies associated to these APIs that control which entities - a user or another microservice - are authorized to access specific APIs. All API access attempts need to be logged for auditability and compliance.

Conclusion

Organizations of all sizes and levels of security maturity are rapidly adopting microservices. These implementations are driving the need for security teams to fully understand the full set of security requirements in order to not be misled into thinking that legacy security solutions will protect them. Comprehensive microservices security requires a combination of maintaining proper hygiene, monitoring, logging, and compliance. Most importantly in a microservices environment is following the principles of Zero Trust least privilege access control. It is important to take these principles into consideration at all layers of the security stack.

About Aporeto

Aporeto is a Zero Trust security solution for microservices, containers and the cloud. Fundamental to Aporeto's approach is the principle that everything in an application is accessible to everyone and could be compromised at any time. Aporeto uses identity context, vulnerability data, threat monitoring and behavior analysis to build and enforce authentication, authorization and encryption policies for applications. With Aporeto, enterprises implement a uniform security policy decoupled from the underlying infrastructure, enabling workload isolation, API access control and application identity management across public, private or hybrid cloud.

For more information, visit:
www.aporeto.com