DAST The winning approach to microservices security

INTRODUCTION

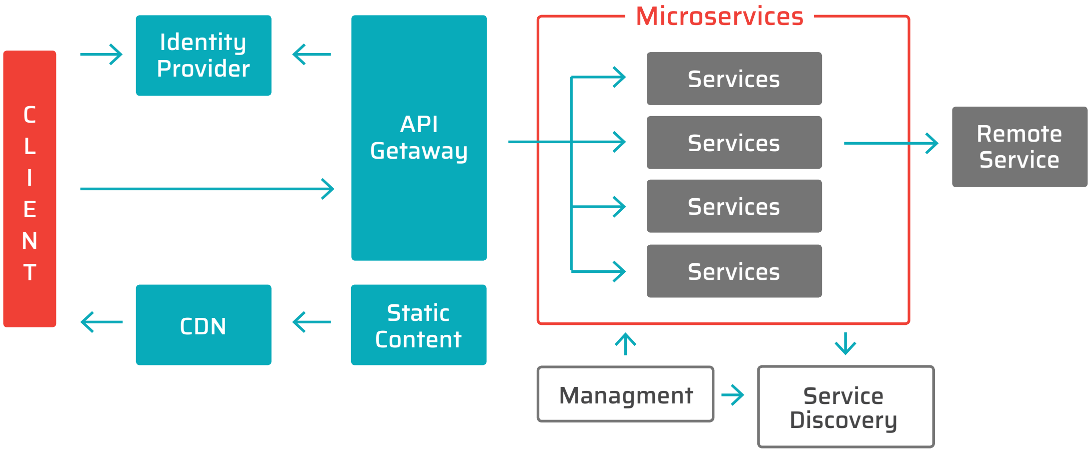
Concepts and approaches to how software is built have evolved over the years from monolithic to a distributed architecture based on composable application components – microservices.

Microservices are single-function chunks of an application's business modules, often isolated within interoperable containers. The distributed approach makes it easier for developers to update and expand applications, resulting in an architecture comprised of reusable and more manageable pieces. While microservices have freed software development from many constraints of monolithic architecture, this new approach exposes applications to additional threats and vulnerabilities that need to be addressed and remediated.

Microservices security testing

Functional decomposition of traditional applications results in a significant number of microservice instances. The mix of new connections between microservices being continuously open throughout the network, along with a number of APIs being exposed to the public, introduces a large number of entry points susceptible to exploitation. Fighting off threats to microservices is only possible by tracking all interactions between application endpoints to expose weaknesses in both runtime behavior and single points of entry.

The Dynamic Application Security Testing method is the only solution able to scan all the functions of the microservice architecture that make up an application and verify its general health and capacity to handle production runtime without unwanted exposure to vulnerabilities.



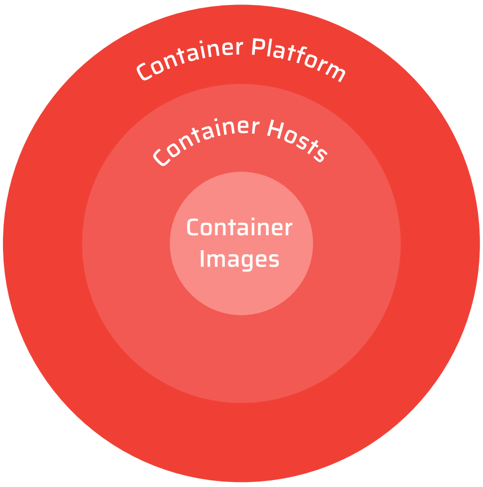
Tech giants using microservices NETFLIX Uber amazon

CONTAINERS

A container is a standard unit of software that packages up the code and its dependencies in a wrapper that can seamlessly move through different computing environments.

Components that a container image consist of are:

- Operating system binaries
- Operating system libraries
- Language runtimes
- Middleware (message buses, application servers)
- Databases, datastores
- Developer code



MICROSERVICES

Microservices architecture is a variant of service-oriented architecture (SOA) for developing large applications consisting of services that are divided into chunks by the business domain. Microservices are more resilient to service disruption thanks to decentralized deployment and decomposition of application functionalities.

Development and runtime principles

Single-purpose functionality

Modular architecture achieved by splitting applications into different composable microservices, with each performing only a single function of a business domain application functionalities.

Decentralized governance and deployment

Each team can distinctly define precisely how their component should be interacted with, resources are shared and replicated across different nodes

Scalability

Microservices can scale to handle increased workloads by adding computing resources to a running host (vertical scalability) or by adding more instances to an existing cluster (horizontal scalability)

Elasticity

The ability for the application component to expand and return to its original state if required by an internal system event or other service runtime criteria

Fault tolerance

The ability for each microservice to tolerate and handle exceptions or crashes without causing any disruption to the application

Consistent orchestration

Host images and containers are tightly orchestrated and follow the development cycle while continuously keeping the consistent configuration

Microservices security challenges

- Segmentation and isolation
- The complexity of multi-cloud deployments
- Identity management and access control
- Data and message integrity
- The rapid rate of change, deprecation cycle

Defensive practices

Defense-in-depth

After splitting applications to microservices, it is essential to reduce host and container footprint while protecting all components of the architecture threat model. The defense-in-depth approach limits the impact of damage that could be done if a vulnerability in a microservice is exploited.

Container

- Enable user namespaces
- Use application-specific AppArmor or SELinux if possible
- Use application-specific seccomp whitelist if possible
- Harden kernel settings
- Bind services to local or private network interfaces
- Deploy immutable containers

Host

- Run read-only images
- Limit SSH access with public key authentication and isolate over VPN
- Deploy with well understood and controlled configuration

Platform

- Use role-based authorization and user demarcation
- Use centralized multi-factor authentication
- Isolate environments to development, staging, and production

Load	Stress	Resiliency
Can the application handle expected amounts of traffic without disruptions or misbehavior?	Does the application crash when affected by unintended amounts of introduced workload?	Is the application resilient to failures and crashes of architecture components?

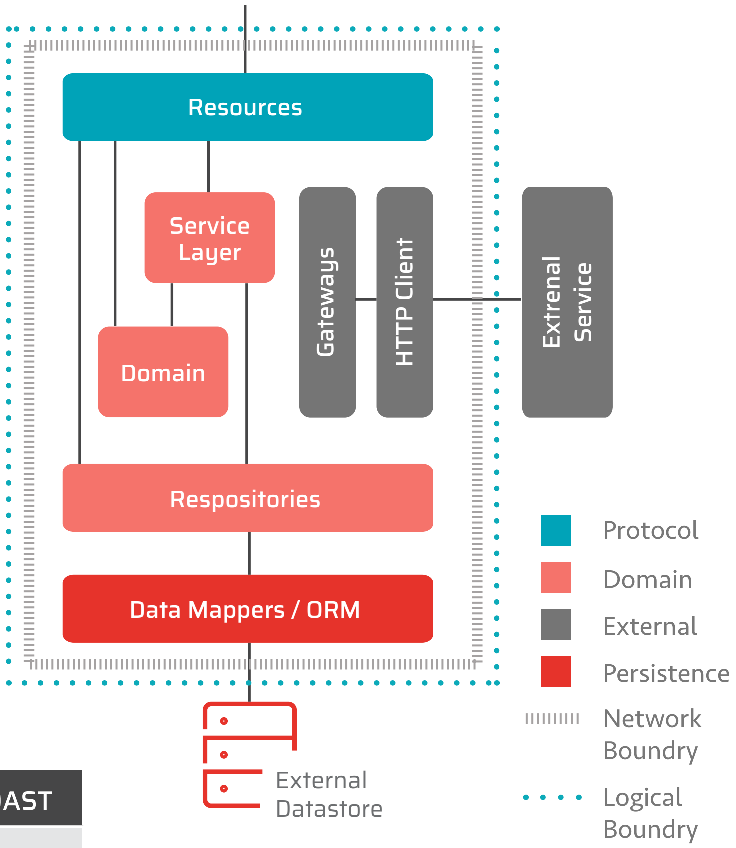
DAST - A winning approach to microservices security

Microservices perform a wide range of functions by passing messages within the environment to initiate requests and form responses. Applications interact with and connect to many services, gateways, or resource handlers, which means complex network traffic is continuously introduced and distributed over the network. Automated tests are able to provide effective vulnerability scanning of a running application only if they can provide full coverage and expose vulnerabilities throughout complex dynamics and runtime behavior of interactions between microservices. Verifying an application's runtime health requires Dynamic Application Security Testing which will pinpoint weaknesses end-to-end, without leaving any loopholes that negatively affect production deployment.

DAST ensures end-to-end security testing with high scenario coverage to identify and expose vulnerabilities of a running application

Common microservice functions

- Establishing and maintaining communication channels
- Transferring data over the network to another service
- Fighting off unauthorized actions and illicit network traffic
- Handling system and service crashes
- Pulling new code through CI/CD pipeline
- Logging and reporting events



SAST vs. DAST

	SAST	DAST
Scan with end-to-end coverage	✗	✓
Confirm readiness for deploying an application to production	✗	✓
Detect and handle microservice architecture scan barriers	✗	✓
Track, record and scan application traffic in its runtime	✗	✓

Advantages of end-to-end DAST

An end-to-end test examines the entire system and verifies that it meets the requirements and achieves its goals. The main purpose of end-to-end testing is to know if the composable parts of the microservice architecture as a whole meet business objectives while the application is in its running state. In a DAST end-to-end scan, the system is treated as a black box while probing and interacting with public entry points, APIs, and with as much of the running system as possible.

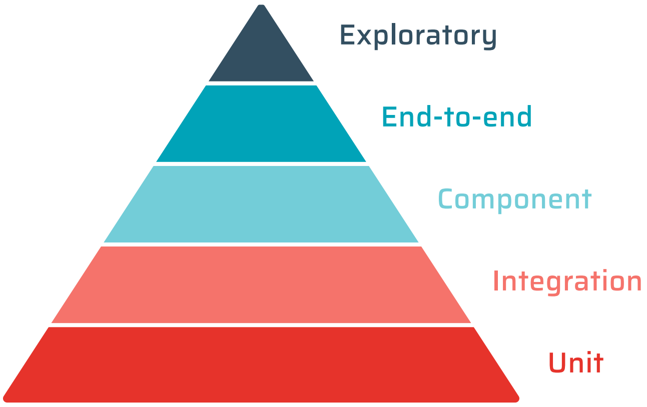
As microservice architecture evolves over time, along-side the rapid rate of changes and the application life cycle, end-to-end testing allows adaptation and learning about applications' dynamics. The ability to detect and expose vulnerabilities regardless of the complex ever-going architectural changes provides the highest confidence that business objectives will be met in the production runtime of the application.

> End-to-end Dynamic Application Security Testing bridges gaps between moving parts of a microservice architecture by verifying the correctness and behavior of each function of a running application. This ensures that the services are not exposed to vulnerabilities in runtime and therefore minimizes the likelihood of negative outcomes when the application is published to production.

Application testing pyramid

The test pyramid describes methods of application testing and shows relations between a relative number of tests that should be done for each testing method.

At the top of the pyramid is Exploratory testing, used for exploring and learning about the system through interactions, exposing it in ways that were previously considered out of scope.



End-to-end tests

Verify that co-dependent and composable parts of the microservice architecture that make a running application meet business objectives.

Component tests

Limit the scope of tested parts of the architecture, manipulating the system through internal code interfaces.

Integration tests

Verify communication paths and interactions between components to detect interface defects.

Units

Determine if the smallest chunks of application code behave as expected.

CONCLUSION

The transition from the traditional monolithic architecture to microservices raises many questions, including security implications introduced through increased exposure of entry points and communication channels that are now involved in application runtime.

Granular control over application components is both a good and a bad thing. While decentralization and composability allow faster development and delivery, new issues arise directly from the increasing complexity.

Isolating microservices within containers does offer limited protection, but the attack surface still remains wide since the application has dispersed and exposed its functional parts all over the infrastructure. Although composable and deployed with a single purpose, microservices still depend on other components that are constantly interacting with each other over the network to deliver the business objectives of a running application.

When it comes to application security testing, the scanning of source code for vulnerabilities achieves only so much in the first phases of the software development life cycle. In order to confirm that applications built on microservices are ready to face the public and be deployed to production without unwanted exposure to vulnerabilities, the final phase of development requires end-to-end scan coverage of Dynamic Application Security Testing that can track and handle complex growth and dynamics of a decentralized microservice architecture mesh.