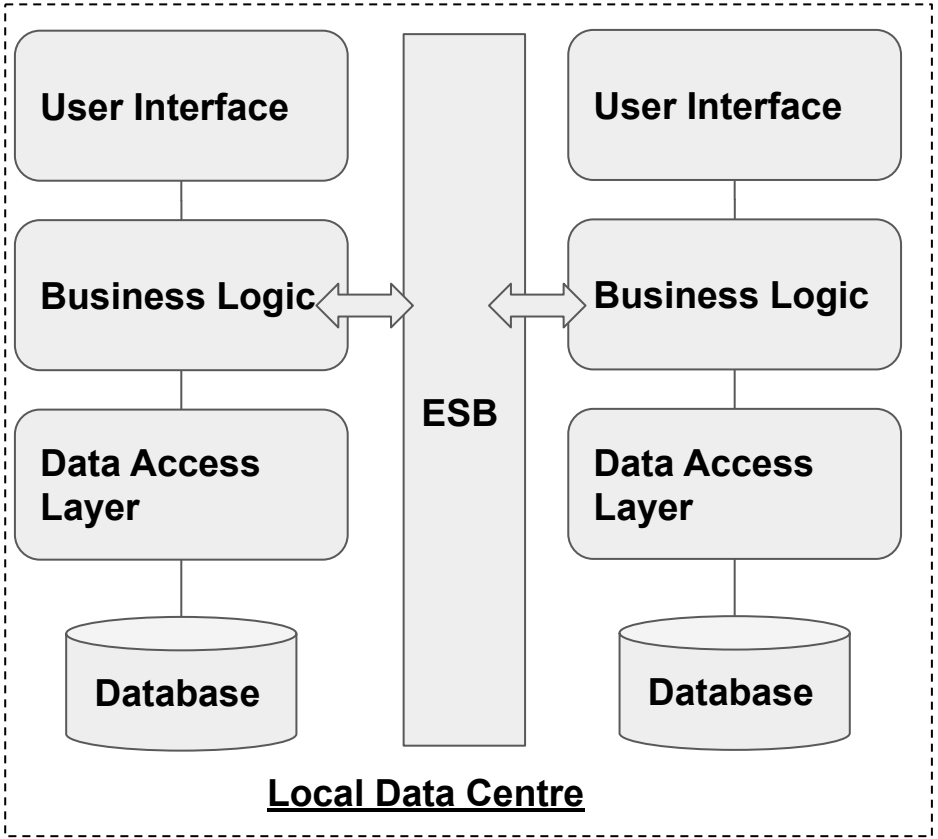
Securing the Multi-cloud, Portable, *-Tier Microservice Application: A live demo on Cloud-Native Application Security Platforms: Curiefense, Deepfense, Sysdig, Snyk Code, and Aqua Security Trivy & tfsec

Nathan Aw
https://www.linkedin.com/in/awnathan
nathan.mk.aw@gmail.com
https://owasp.org/www-chapter-singapore/
5 August 2021

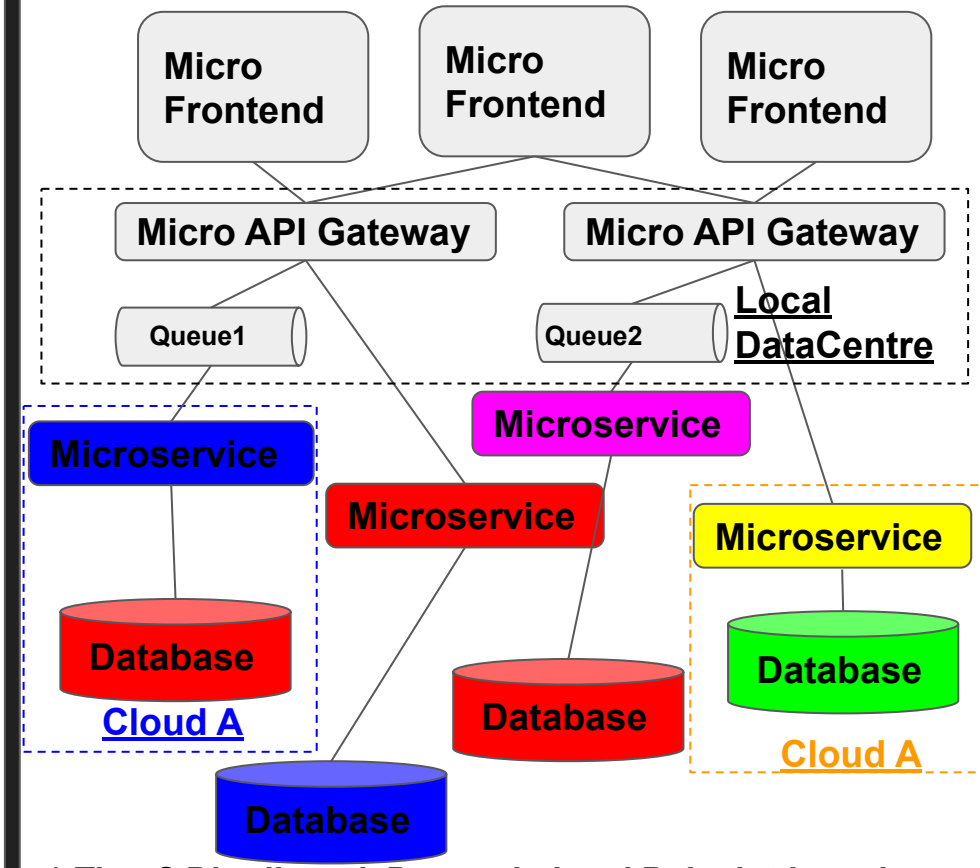
"Are microservices more or less secure than monoliths?"

Opinions/views expressed in the talk are solely my own and do not express the views or opinions of my employer.

The Technical Context: Monolithic vs Microservices (*Highly*-Simplified)

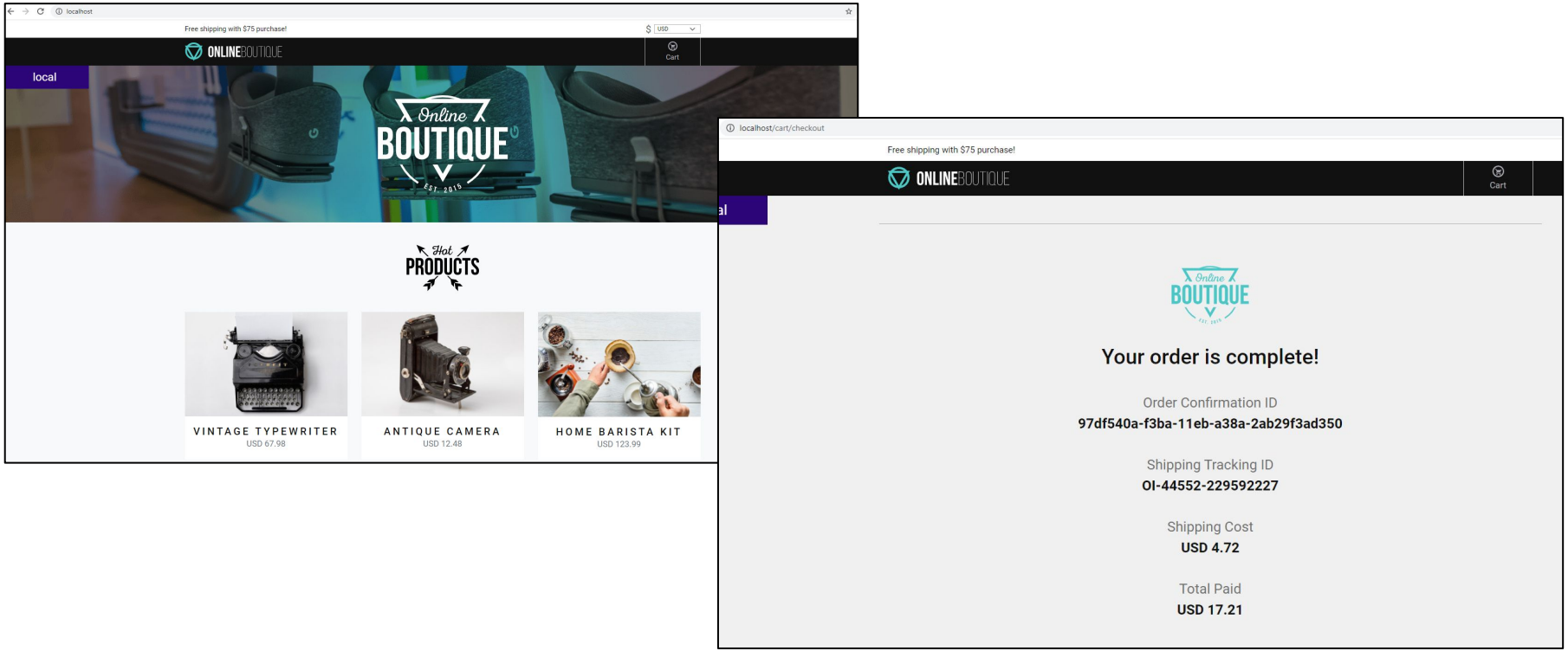


N-Tier / Multi Tier Architecture



*-Tiers? Distributed, Decoupled and Polyglot-based architecture

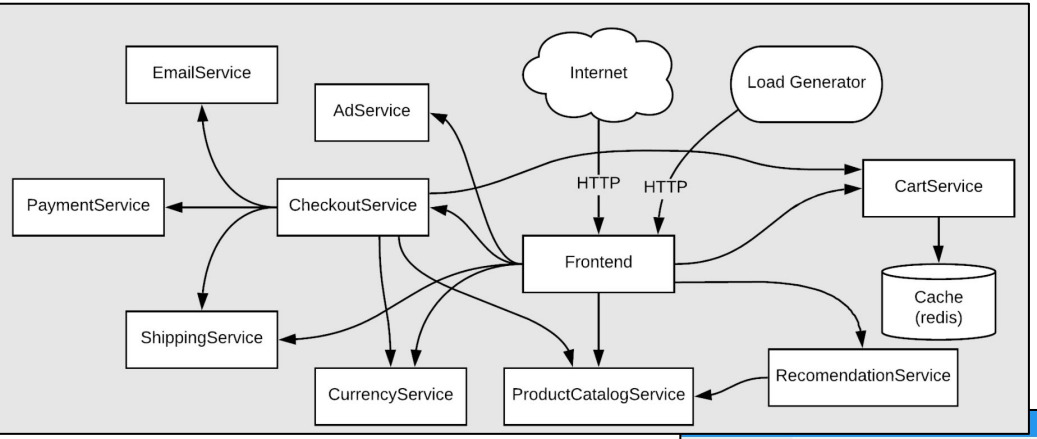
Demo of Online Boutique: A 10-Tier Sample Microservice App on Docker Desktop



“Online Boutique is a cloud-native microservices demo application. Online Boutique consists of a **10-tier** microservices application. The application is a web-based e-commerce app where users can browse items, add them to the cart, and purchase them.”

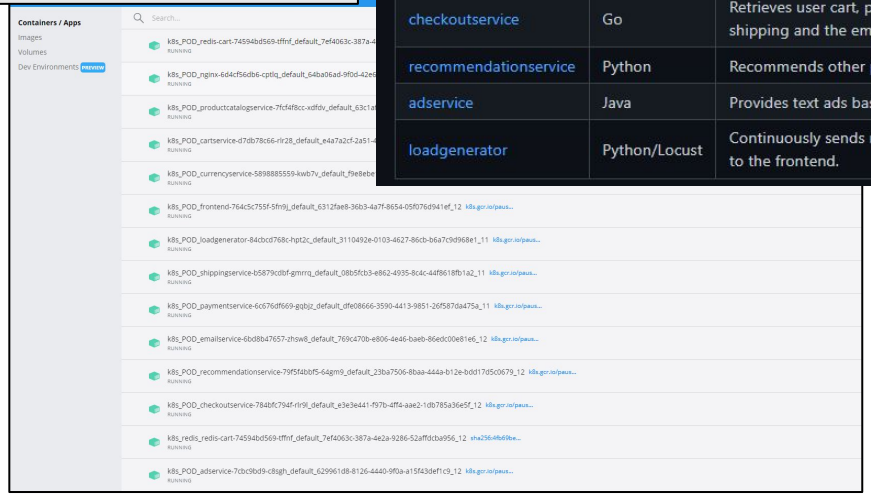
SOURCE: https://github.com/GoogleCloudPlatform/microservices-demo

Architecture of Online Boutique - 11 Microservices written in different languages!



Service	Language	Description
frontend	Go	Exposes an HTTP server to serve the website. Does not require signup/login and generates session IDs for all users automatically.
cartservice	C#	Stores the items in the user's shopping cart in Redis and retrieves it.
productcatalogservice	Go	Provides the list of products from a JSON file and ability to search products and get individual products.
currencyservice	Node.js	Converts one money amount to another currency. Uses real values fetched from European Central Bank. It's the highest QPS service.
paymentservice	Node.js	Charges the given credit card info (mock) with the given amount and returns a transaction ID.
shippingservice	Go	Gives shipping cost estimates based on the shopping cart. Ships items to the given address (mock)
emailservice	Python	Sends users an order confirmation email (mock).
checkoutservice	Go	Retrieves user cart, prepares order and orchestrates the payment, shipping and the email notification.
recommendationservice	Python	Recommends other products based on what's given in the cart.
adservice	Java	Provides text ads based on given context words.
loadgenerator	Python/Locust	Continuously sends requests imitating realistic user shopping flows to the frontend.

Online Boutique is composed of 11 microservices written in different languages (polyglot!) that talk to each other over gRPC.



SOURCE:
https://github.com/GoogleCloudPlatform/microservices-demo

This Meetup Session - Agenda

1. The Technical Context: Monolithic vs Microservices
2. The Context and Challenges of Securing the Multi-cloud, Portable, *-Tier Microservices Application
3. The 4Cs of Cloud-Native Security: Code, Container, Cluster, Cloud
4. Curiefense Demo: Open Native Application Security Platform that protects all forms of web traffic, services, and APIs
5. Deepfence Cloud Native Protection Platform Demo
6. Snyk Code Demo
7. Sysdig Demo
8. Aqua Security Trivy and tfsec Demo
9. Recap
10. “Are microservices more or less secure than monoliths?”

Who I am.

First and foremost a microservices developer with keen interest in AppSec. Working in an **end-user environment** at a financial institution.

Unyielding passion to build secure, scalable **polyglot** microservices that can run **anywhere** (multi-cloud and/or on-premise) and scale limitlessly.

Previous OWASP Presentations:

https://owasp.org/www-chapter-singapore/assets/presos/Securing_your_APIs_-_OWASP_API_Top_10_2019,_Real-life_Case.pdf

https://owasp.org/www-chapter-singapore/assets/presos/Deconstructing_the_Solarwinds_Supply_Chain_Attack_and_Deterring_it_Honing_in_on_the_Golden_SAML_Attack_Technique.pdf

https://owasp.org/www-chapter-singapore/assets/presos/Microservices%20Security%2C%20Container%20Runtime%20Security%2C%20MITRE%20ATT%26CK%C2%AE%20%20for%20Kubernetes%20(K8S)%20and%20Service%20Mesh%20for%20Security.pdf

Opinions/views expressed in the talk are solely my own and do not express the views or opinions of my employer.

The Challenges

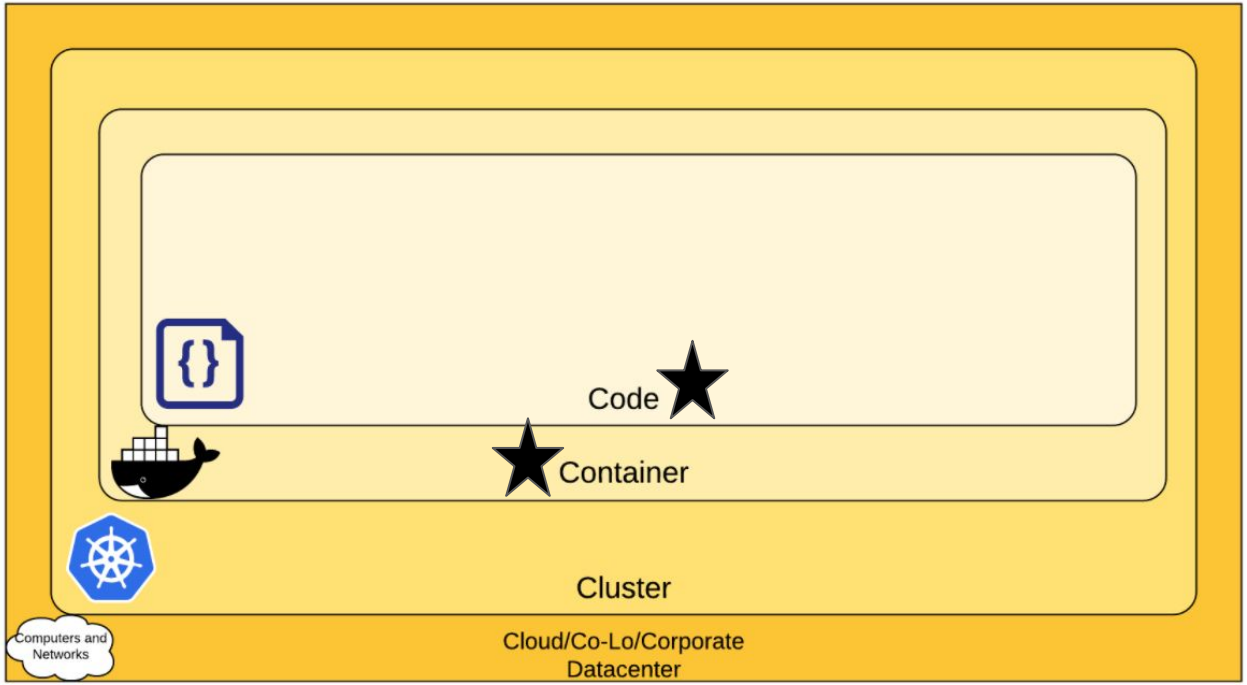
Context | Problem Statement

"The perimeter is no longer just the physical location of the enterprise [data center], and what lies inside the perimeter is no longer a blessed and safe place to host personal computing devices and enterprise applications [microservices]." https://cloud.google.com/security/beyondprod/

Traditional Infrastructure Security	Cloud-Native Security	Implied Requirements for Cloud-native Security
Perimeter-based security (i.e. firewall), with internal communications considered trusted.	**Zero-trust security with service-to-service communication verified, and no implicit trust for services in the environment.**	***Protection of the network at the edge (remains applicable) and no inherent mutual trust between services.***
Fixed IPs and hardware for certain applications.	Greater resource utilization, reuse, and sharing, including of IPs and hardware.	***Trusted machines running code with known provenance.***
IP address-based identity.	Service based identity.	
Services run in a known, expected location.	Services can run anywhere in the environment, including hybrid deployments across the public cloud and private data centers.	

SOURCE: https://cloud.google.com/security/beyondprod/

Overview of Cloud Native Security: The 4C's of Cloud Native security



In today's session and demo, we will look at how we can use some of these cloud-native platforms and tools to secure Code and Container

Note: This layered approach augments the defense in depth computing approach to security, which is widely regarded as a best practice for securing software systems.

https://kubernetes.io/docs/concepts/security/overview/

Curiefense

Curiefense is the open source cloud native application security platform that protects all forms of web traffic, services, and APIs. It includes bot management, WAF, application-layer DDoS protection, session profiling, advanced rate limiting, and much more. Curiefense is integrated with **NGINX and Envoy proxy.** It blocks hostile traffic from sites, services, microservices, containers, service meshes, and more.

Curiefense is an API-first, DevOps oriented web-defense HTTP-Filter adapter for **Envoy and NGINX**. It provides multiple security technologies (WAF, application-layer DDoS protection, bot management, and more) along with real-time traffic monitoring and transparency.

Curiefense is fully controllable programmatically. All configuration data (security rulesets, policies, etc.) can be maintained singularly, or as different branches for different environments, as you choose. All changes are versioned, and reverts can be done at any time.

Curiefense also has a UI console. **Curiefense is currently a CNCF Sandbox Project.**

Envoy is an Open Source Edge and Service Proxy, designed for cloud native applications. Originally built at Lyft, Envoy is a high performance C++ distributed proxy designed for single services and applications, as well as a communication bus and “universal data plane” designed for large microservice “service mesh” architectures. Envoy runs alongside every application and abstracts the network by providing common features in a platform-agnostic manner.

SOURCE: https://docs.curiefense.io/; https://www.curiefense.io/; https://www.envoyproxy.io/; https://www.curiefense.io/blog/announcing-nginx-integration

Curiefense Architecture and its components

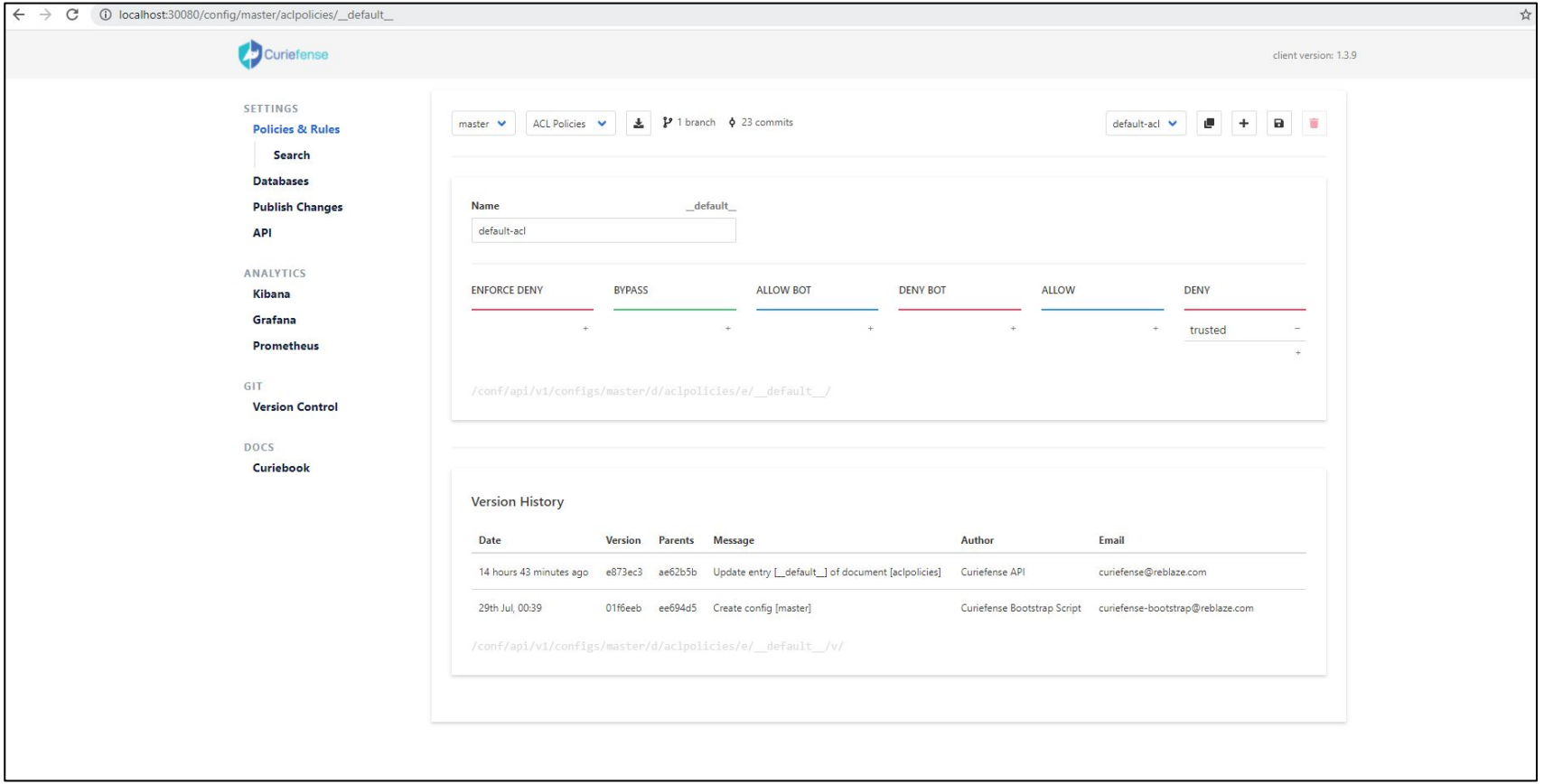
Container Name	Purpose and Functionality
curieproxy	(Represented by the column with the Curiefense logo) Performs traffic filtering
curiesync	Ensures configurations are always in sync with latest policies and rules changes
curietasker	Runs periodic maintenance tasks
curielogger	Pushes Envoy access log to postgresql and metrics to prometheus
confserver	API server to manage configuration
uiserver	UI Management Console
echo	Dummy web server for testing
elasticsearch *	Stores access logs
kibana	Displays logs
filebeat	Sends logs to elasticsearch
grafana *	Dashboards
prometheus *	Stores time series metrics
redis *	Synchronizes session and rules across deployments and Envoy containers
	(*) You may replace these containers with your own if desired.

How Curiefense Works

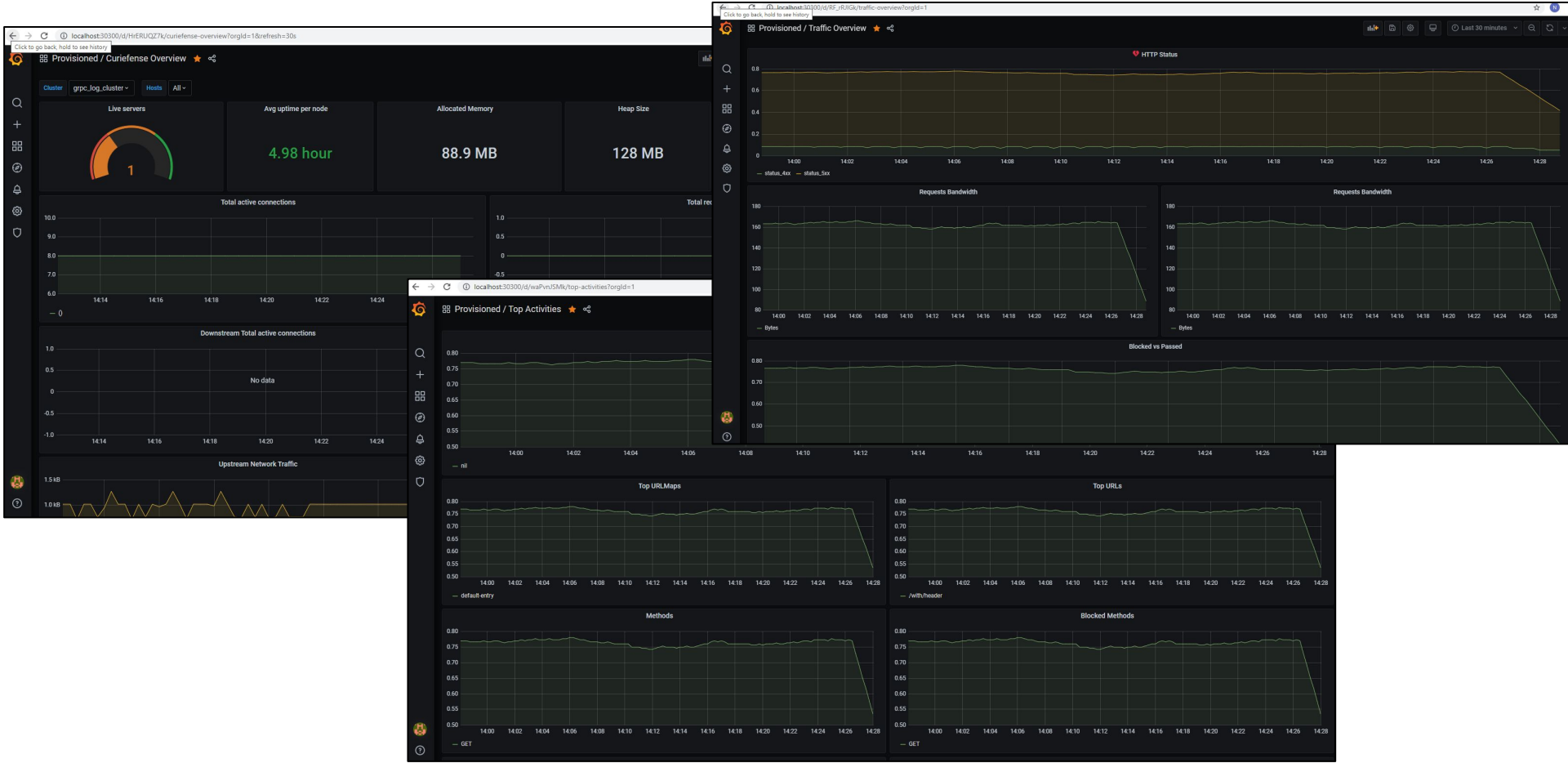
Curiefense runs every incoming request (and in some cases, responses as well) through a series of mechanisms. We will now walk through some of them to understand how traffic is handled and processed.

During the procedures described on the left, we will set up some simple rules and then run some requests through Curiefense. By the end of this process, we will understand how to create security rules and policies, you will observe them being applied, and you will see how Curiefense reports on incoming requests and its reactions to them.

Curiefense Management UI: http://localhost:30080/



Curiefense Grafana: http://localhost:30300/



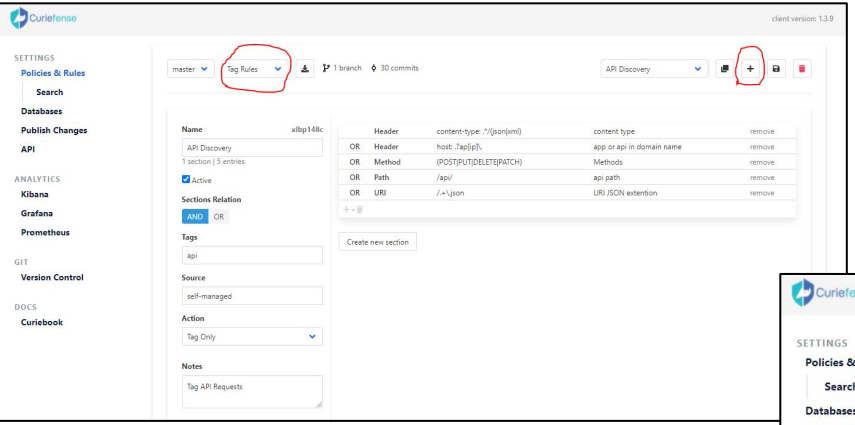
What we are going to do

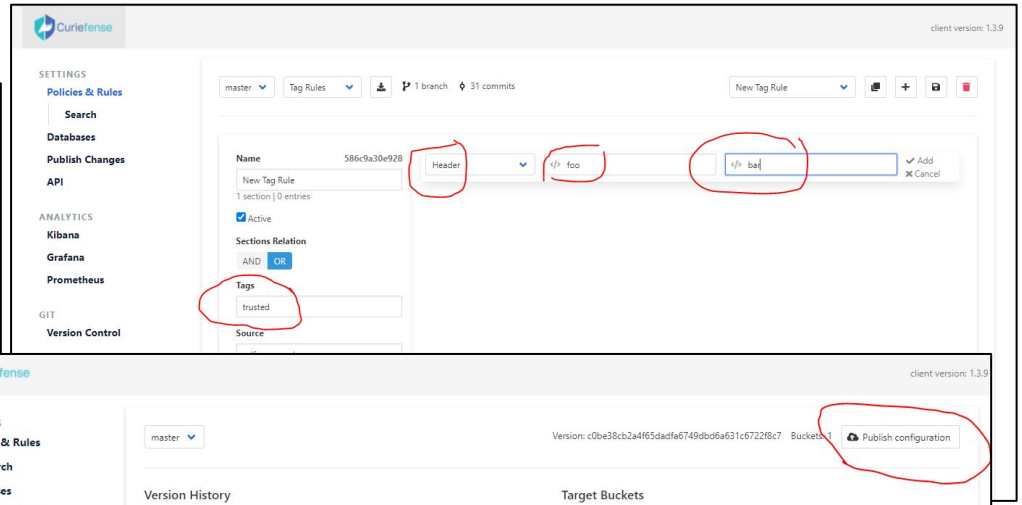
Create a Tag Rule

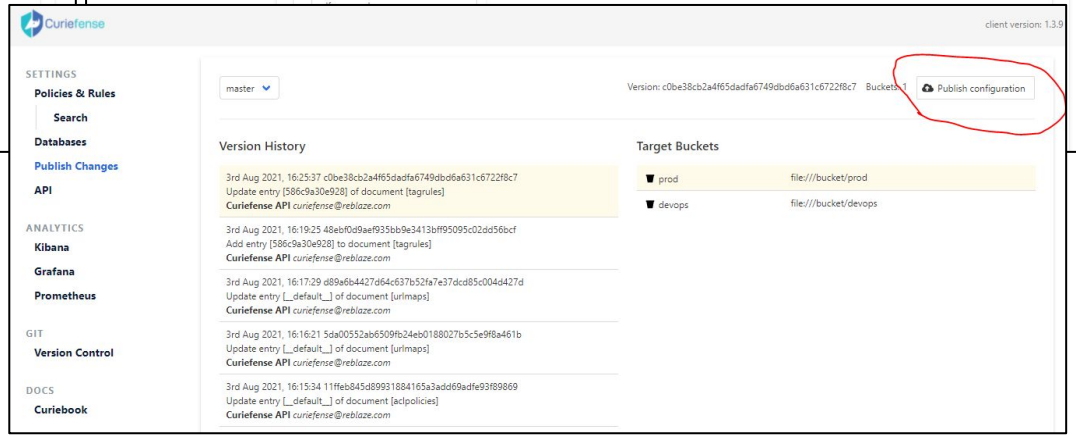
Create an ACL (Access Control List)

Test the ACL

Activate the ACL





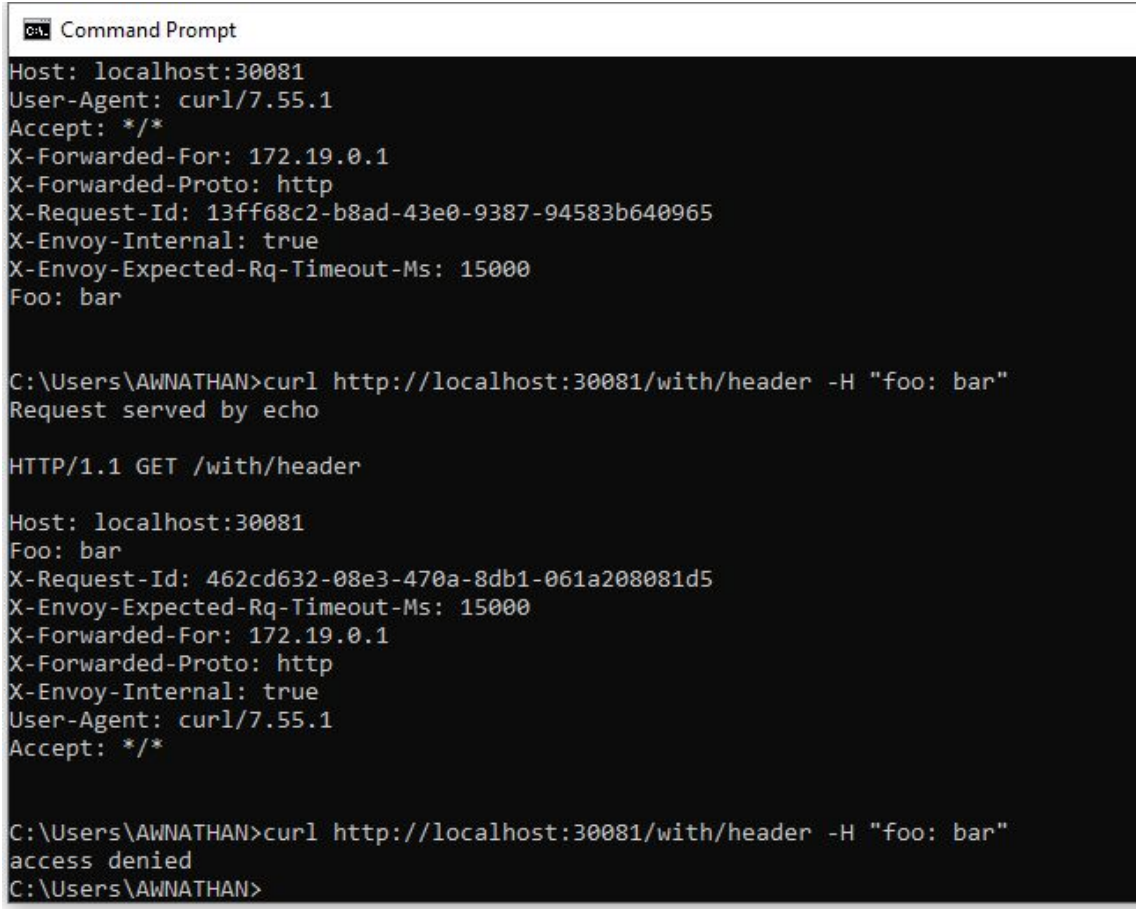


curl http://localhost:30081/no/header

curl http://localhost:30081/with/header -H "nathan: aw"

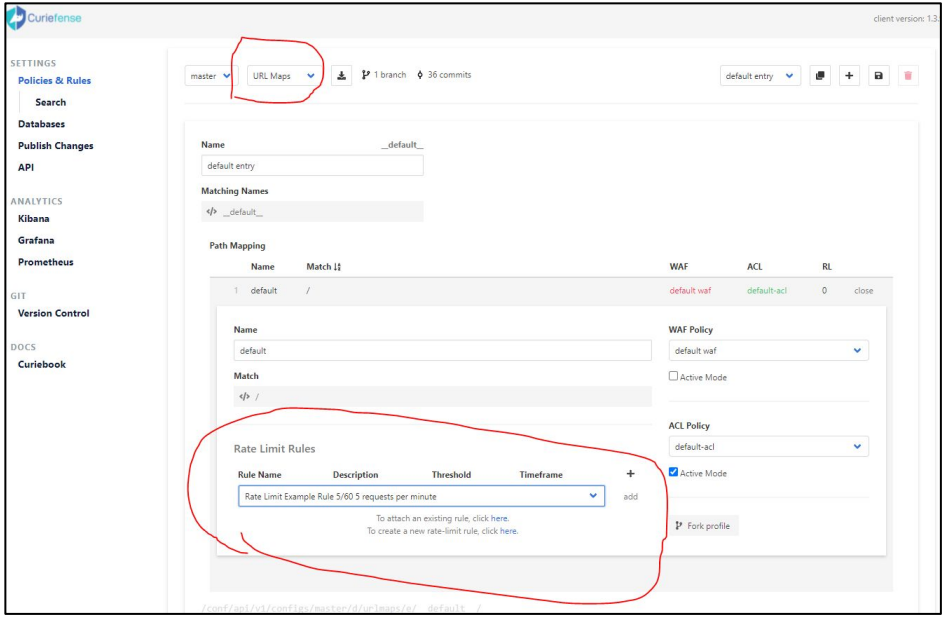
Curiefense

Demo Results

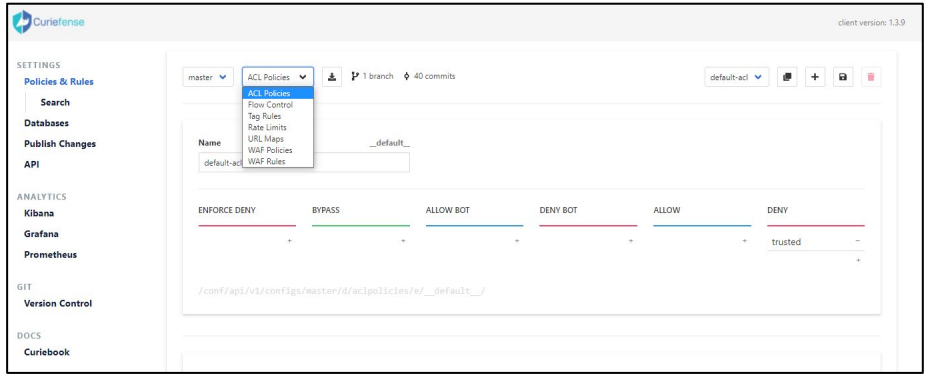


Result of apply ACL on HTTP headers

Curiefense



Adding a Rate Limit & Other Policies and Rules




```
while true;
do
    curl -s -o /dev/null -w "%{http_code}" http://localhost:30081/with/header -H "foo: bar";
    printf ", ";
    sleep 1s;
done
```

```
$ ./curl.sh
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 403, 403, 403,
403, 403, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 403, 403, 403, 403, 403, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503,
503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 503, 403, 403, 403,
403, 403,
```

Curiefense Features (1/3)

Web Application Firewall

Curiefense's full-featured WAF protects against the OWASP Top 10 and many other threats:

1. Negative security features (signatures, block-listing, etc.) exclude known threats.
2. Positive security features (input validation, schema enforcement, etc.) exclude anomalies and zero-days.
3. The platform has capabilities that are not offered by many commercial solutions, including content filtering, payload inspection, behavioral profiling, and more.

Advanced Rate Limiting

Curiefense provides rate-limiting capabilities, definable at any scope (from globally down to individual URLs). Traffic sources are blocked when their requests exceed predefined limits. This mechanism has a wide variety of beneficial uses, including:

1. Prevention of account takeover attacks (credential stuffing, credential discovery, etc.)
2. Prevention of other brute-force threats, e.g. payment card validation
3. Detecting anomalies, e.g. a single user changes geolocation multiple times
4. Blocking inventory denial attacks

And more…..

Curiefense Features (2/3)

Application -Layer DDoS Protection	Curiefense defends against layer 7 DoS/DDoS at all scales, from massive DDoS botnet assaults to single malformed-packet DoS attempts.
API Security	APIs and services are protected with the full range of capabilities that are applied to web applications. The only exceptions are techniques which do not apply to APIs, such as browser environment verification. Curiefense also has additional capabilities that apply only to APIs and services, such as schema enforcement.
DevOps	1. Curiefense can be deployed and controlled via web console, REST API, and/or a CLI tool. 2. JSON or YAML configuration format 3. Git versioning as the configuration storage engine 4. Support for environment branching (e.g., prod/devops/qa)

Curiefense Features (3/3)

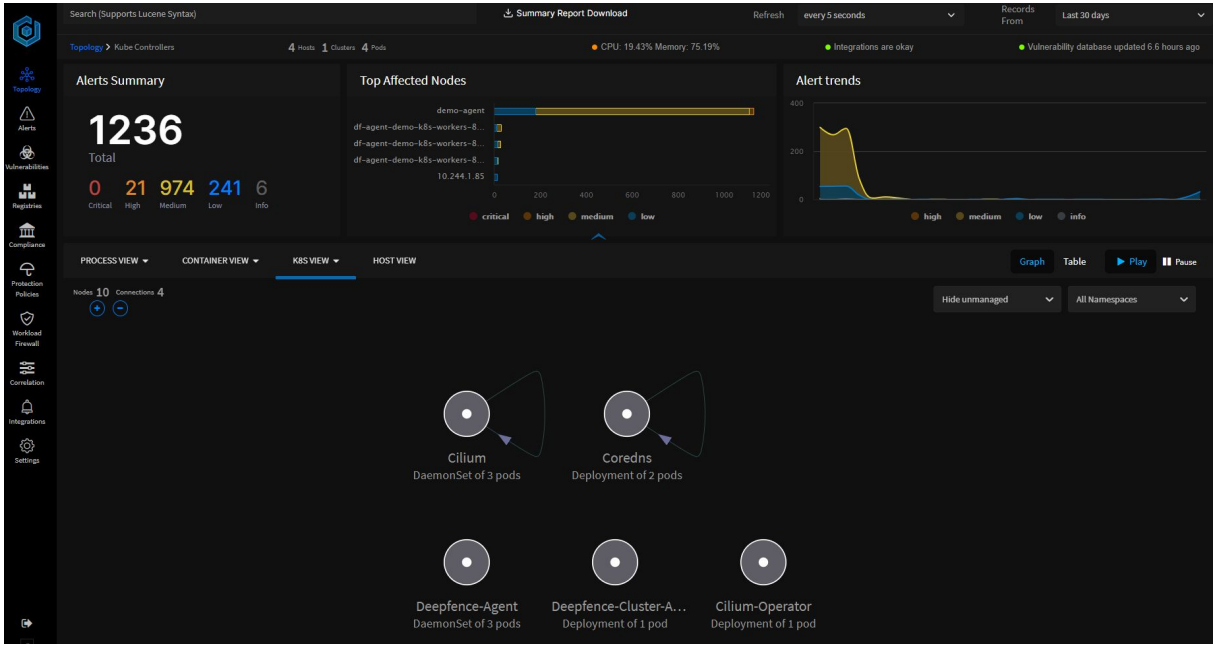
Logging and Real-Time Reporting	Curiefense logs and reports all details (headers, payloads, tags, disposition, etc.) of all requests. Out of the box, it includes Prometheus metrics and Grafana dashboards. Users can customize these, or can swap in their own reporting and visualization frameworks.
Threat Feeds	Curiefense consumes external threat feeds (e.g., IP reputation), auto-updating its security posture as the threat environment evolves. Weekly updates are included for IP reputation, ASNs, etc.

Deepfense: Unified Cloud Native Security Observability

For K8s, Serverless and more.

Deployed as a set of microservices with subatomic footprint, Deepfence is a resilient distributed intrusion prevention system that measures and maps runtime attack surface, and provides full-stack protection from known and unknown threats.

Deepfence is deterministic security for modern workloads.



Microservices Security, Monitoring vs Observability: “What’s broken, and why?” (1/2)

Challenges	An obvious area where a Microservices Architecture adds complexity is communications between services; visibility into service to service communications can be hard to achieve, but is critical to building an optimized and resilient architecture. Most failures in the microservices space occur during the interactions between services, so a view into those transactions helps teams better manage architectures to avoid failures.
Microservices Observability	Observability is about data exposure and easy access to information which is critical when you need a way to see when communications fail, do not occur as expected or occur when they shouldn’t. The way services interact with each other at runtime needs to be monitored, managed and controlled. This begins with observability and the ability to understand the behavior of your microservice architecture. Observability makes it much easier to see what is happening when your services interact with each other, making it easier to build a more efficient, resilient and secure microservice architecture. 'Observability' comes from control theory, an area of engineering concerned with automating control a dynamic system - e.g., the flow of water through a pipe, or the speed of an automobile over inclines and declines - based on feedback from the system.

SOURCE: https://www.infoq.com/news/2018/06/observability-microservices/; https://opentracing.io/docs/overview/what-is-tracing/

Microservices Security, Monitoring vs Observability: “What’s broken, and why?” (2/2)

Two Observability Features

Tracing

Metrics

Distributed tracing, also called distributed request tracing, is a method used to profile and monitor applications, especially those built using a microservices architecture. Distributed tracing helps pinpoint where failures occur and what causes poor performance.

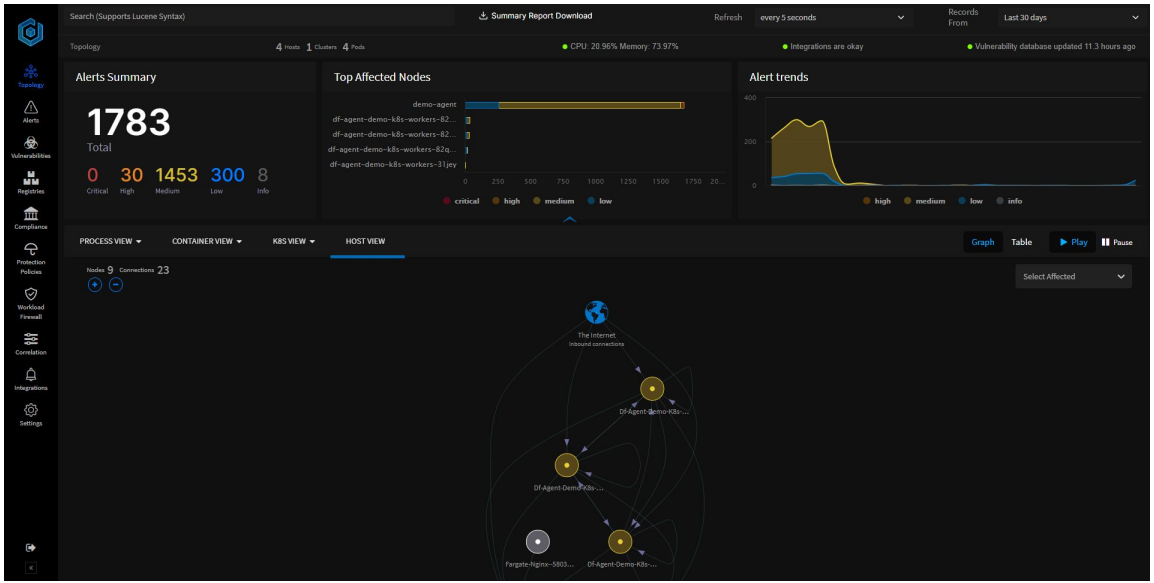
Metrics, based upon the telemetry data which can be gathered automatically across the service mesh. Important metrics to gather include request volume, request duration, latency and request size.

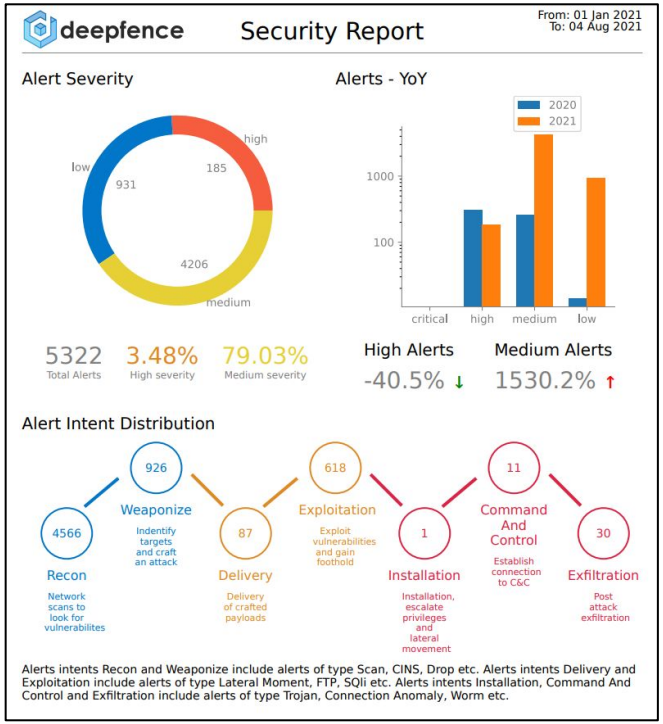
Monitoring vis-a-vis Observability

Observability is often mischaracterized as an overhyped buzzword, or a 'rebranding' of system monitoring in general and application performance monitoring (APM) in particular. In fact, observability is a natural evolution of APM data collection methods that better addresses the increasingly rapid, distributed and dynamic nature of cloud-native application deployments. Observability doesn’t replace monitoring – it enables better monitoring, and better APM.

SOURCE: https://www.infoq.com/news/2018/06/observability-microservices/; https://opentracing.io/docs/overview/what-is-tracing/; https://www.ibm.com/cloud/learn/observability

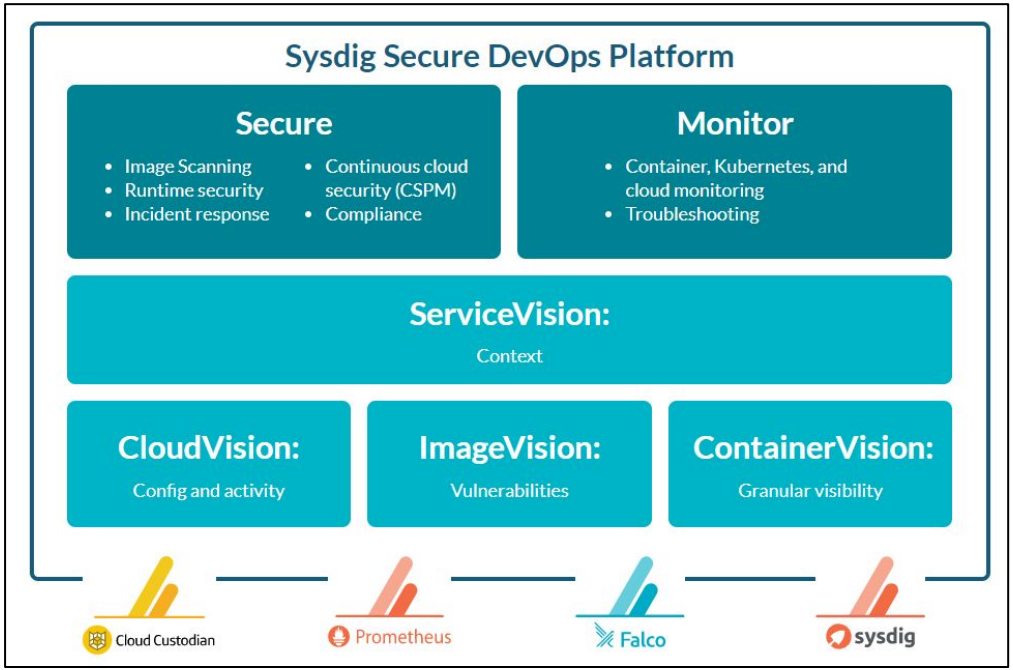
Deepfence Demo

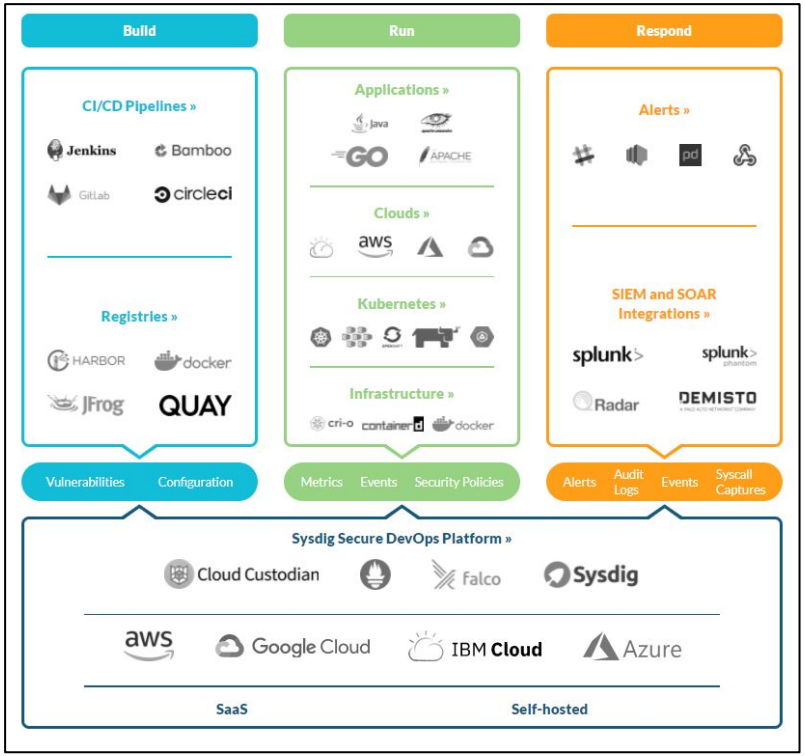




SOURCE: https://deepfence.show/#/topology

Sysdig

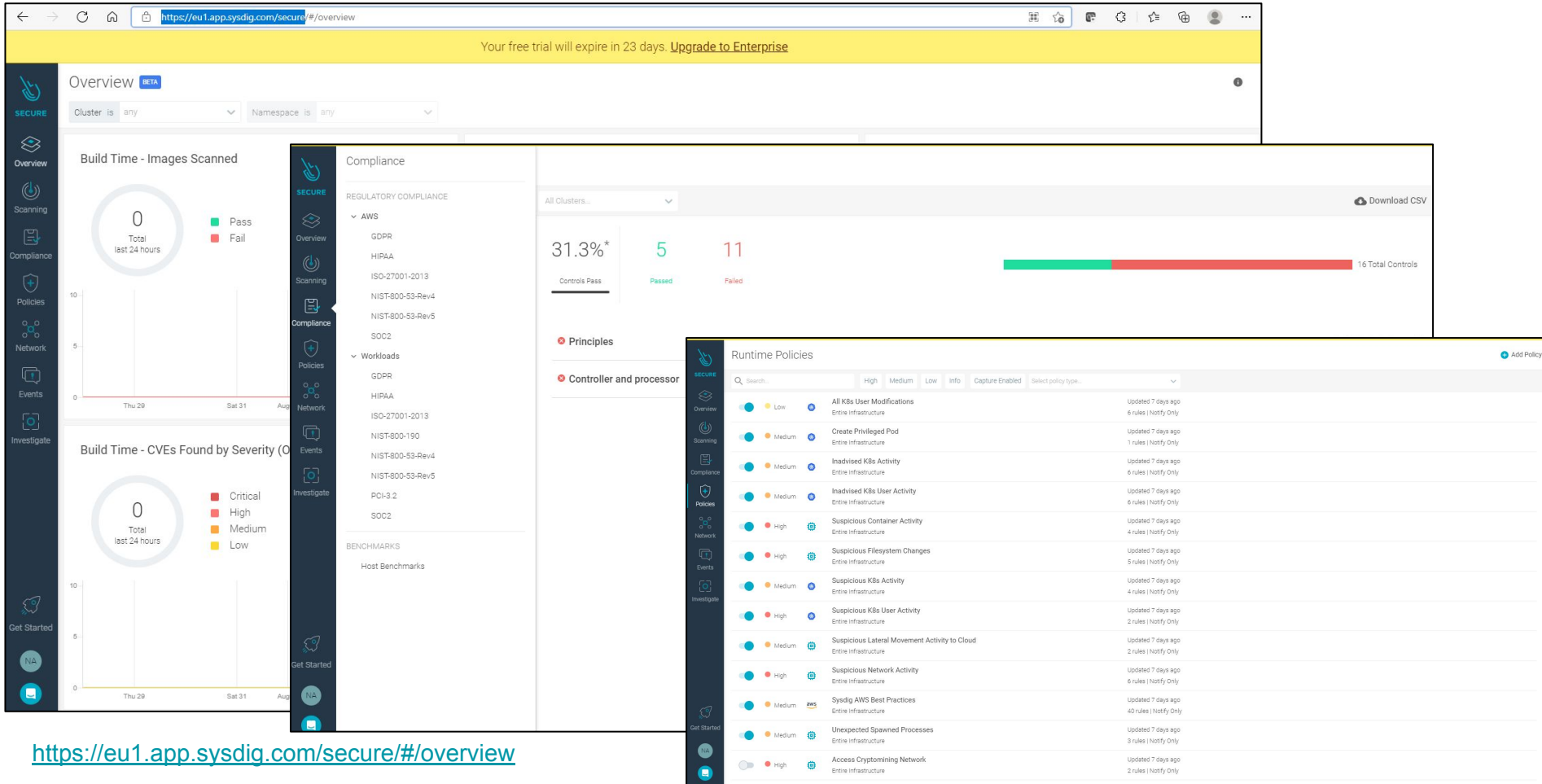




Secure DevOps For containers, Kubernetes, and cloud: Deep Visibility Across Your DevOps Workflow: Unified visibility across workloads and cloud infrastructure from a single security and monitoring event store. Accurately alert on threats, operational issues and compliance risks and respond using a detailed activity record. Easily plug into your existing workflows with out-of-the-box integrations.

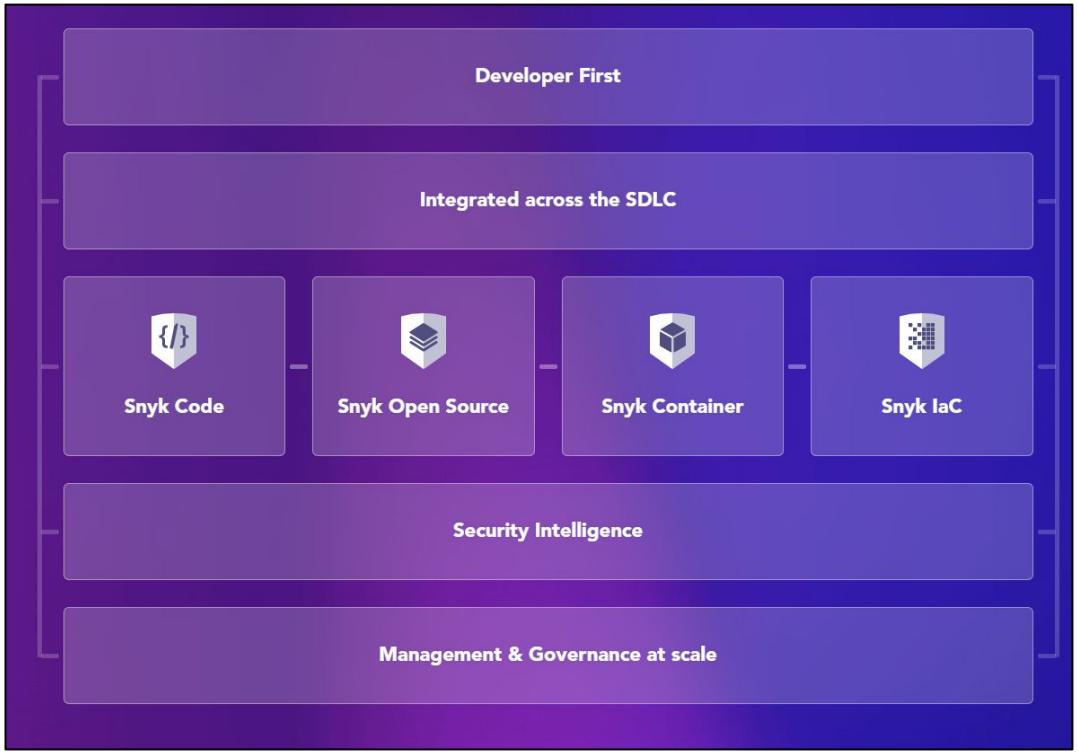
https://sysdig.com/secure-devops-platform/

Sysdig Demo



https://eu1.app.sysdig.com/secure/#/overview

Snyk



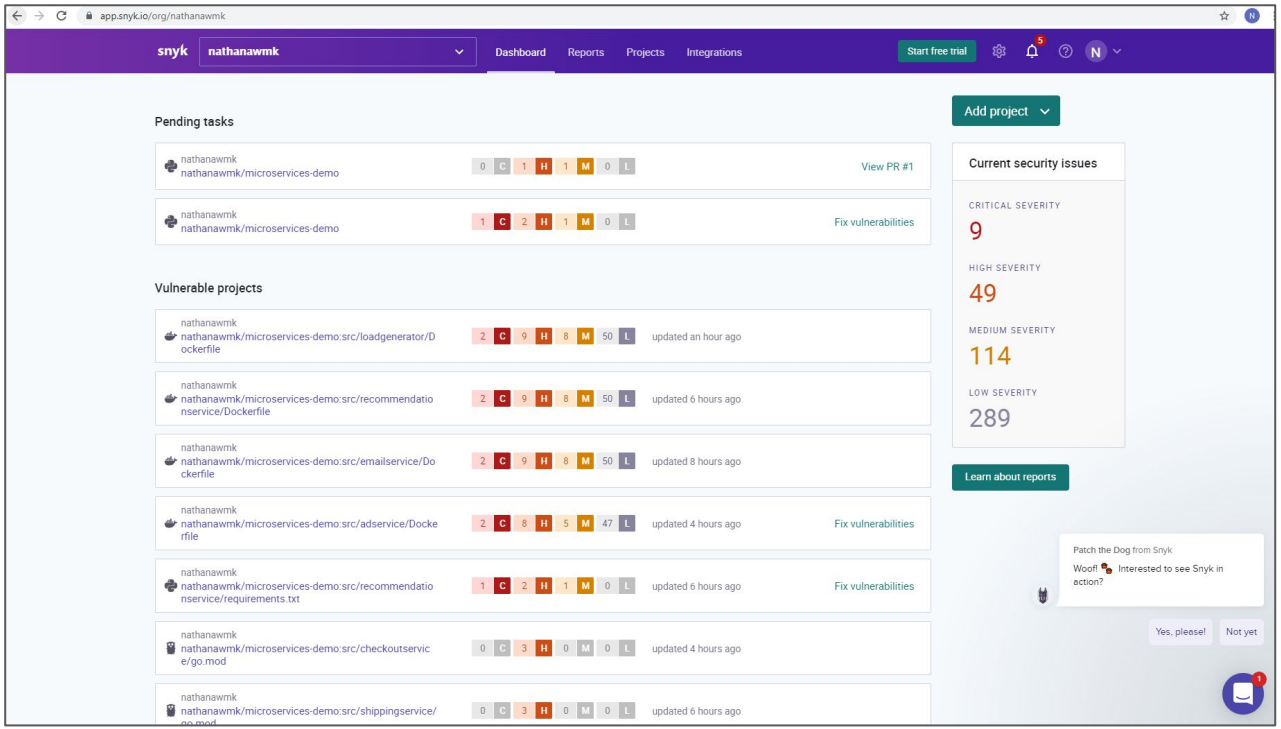
Developer-first Cloud Native Application Security

Security Across the Cloud Native Application Stack: Secure all the components of the modern cloud native application in a single platform

Developer-first: Application security at scale requires developers to be the first step in the security process. Snyk's platform is purpose-built to be easily used by developers to build software securely.

https://snyk.io/

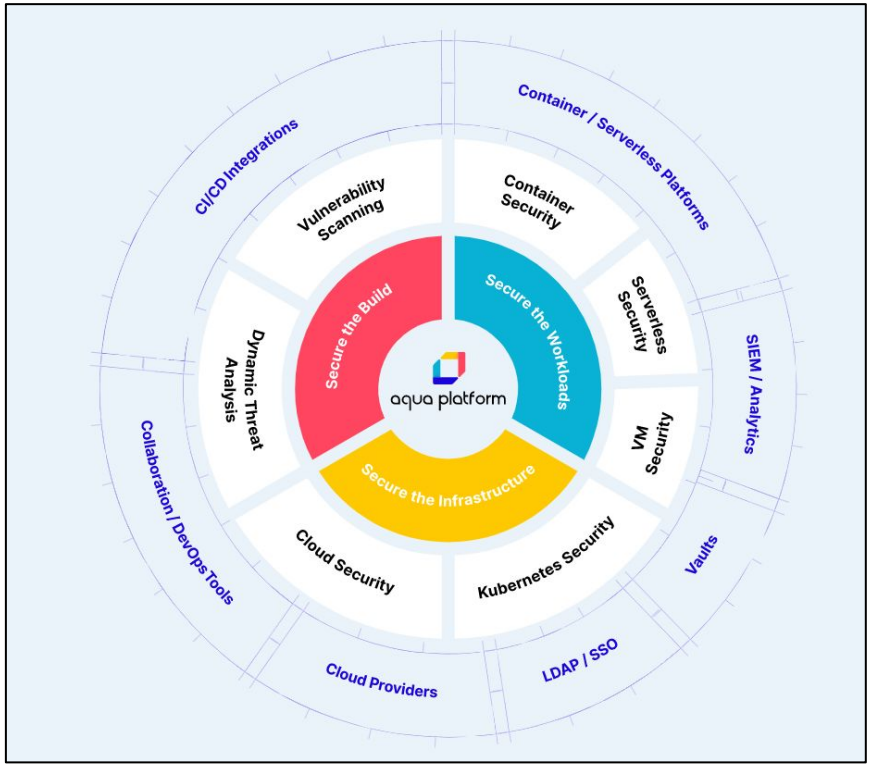
Snyk Code Demo



https://app.snyk.io/org/nathanawmk

Secure code with a developer-friendly experience:
Snyk Code uses a revolutionary approach designed to be developer-first. Conventional Static Application Security Testing (SAST) tools are limited by lengthy scans times and poor accuracy, returning too many false positives, and eroding developer trust. Snyk Code makes developer efforts efficient and actionable.

Aqua Security



Aqua secures your applications wherever you develop and run them: Across clouds, container and serverless platforms, CI/CD pipelines, registries, DevOps tools and modes of deployment, orchestrators, all the way to Security, SIEM, and Analytics.

Secure the Build: Release and update software at DevOps speed with security automation. Detect vulnerabilities and malware early and fix them fast, and allow only safe artifacts to progress through your CI/CD pipeline.

Secure the Infrastructure: Deploy your cloud native applications on any infrastructure while ensuring that cloud services, orchestration and hosts are securely configured and in compliance.

Secure the Workloads: Detect and block policy violations in your workloads using granular controls that are natively architected to provide the optimal response, at scale.

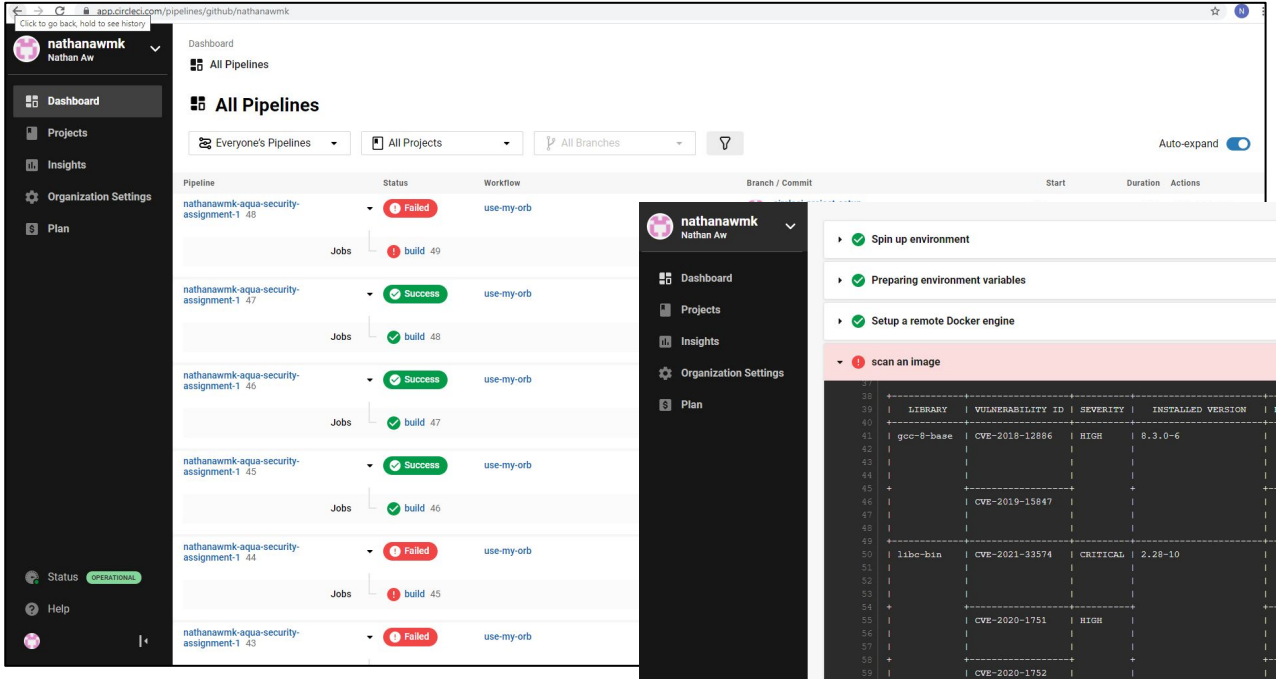
https://www.aquasec.com/

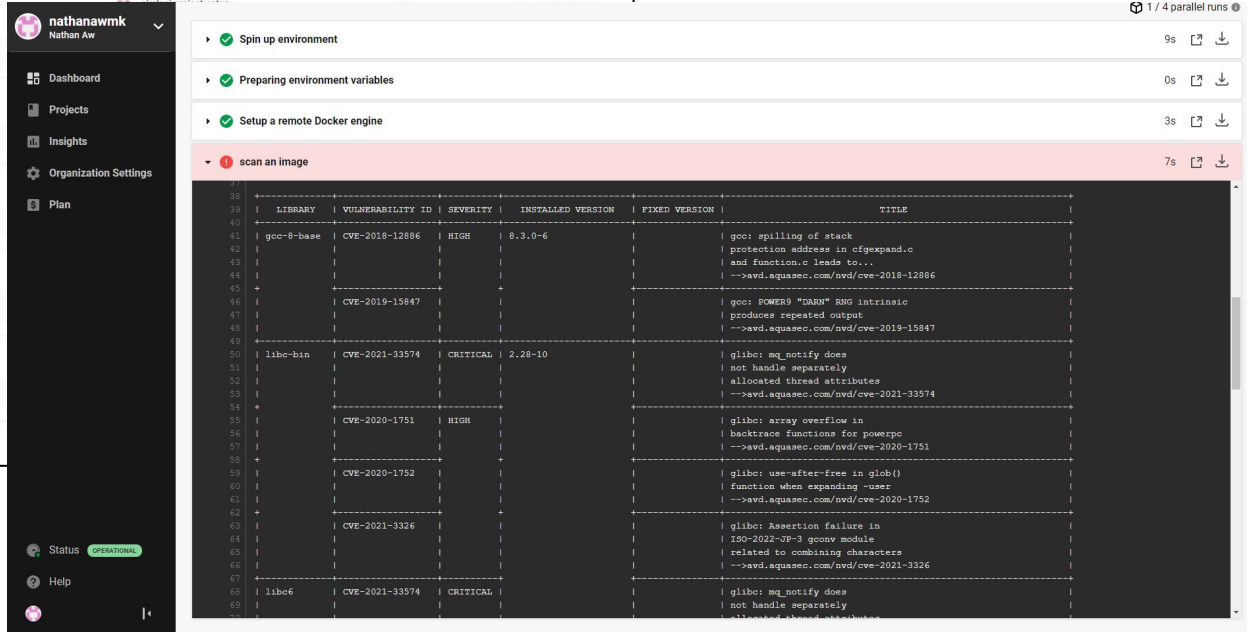
Aqua Security - Trivy + Circle CI Demo

Trivy (tri pronounced like trigger, vy pronounced like envy) is a simple and comprehensive scanner for vulnerabilities in container images, file systems, and Git repositories, as well as for configuration issues. Trivy detects vulnerabilities of OS packages (Alpine, RHEL, CentOS, etc.) and language-specific packages (Bundler, Composer, npm, yarn, etc.). In addition, Trivy scans Infrastructure as Code (IaC) files such as Terraform, Dockerfile and Kubernetes, to detect potential configuration issues that expose your deployments to the risk of attack. Trivy is easy to use. Just install the binary and you're ready to scan.

SOURCE: https://github.com/aquasecurity/trivy

Aqua Trivy - Demo





1. Scan an Alpine Linux Docker Image
2. Scan an Nginx Image

https://app.circleci.com/pipelines/github/nathanawmk

https://github.com/nathanawmk/nathanawmk-aqua-security-assignment-1/blob/circleci-project-setup/.circleci/config.yml

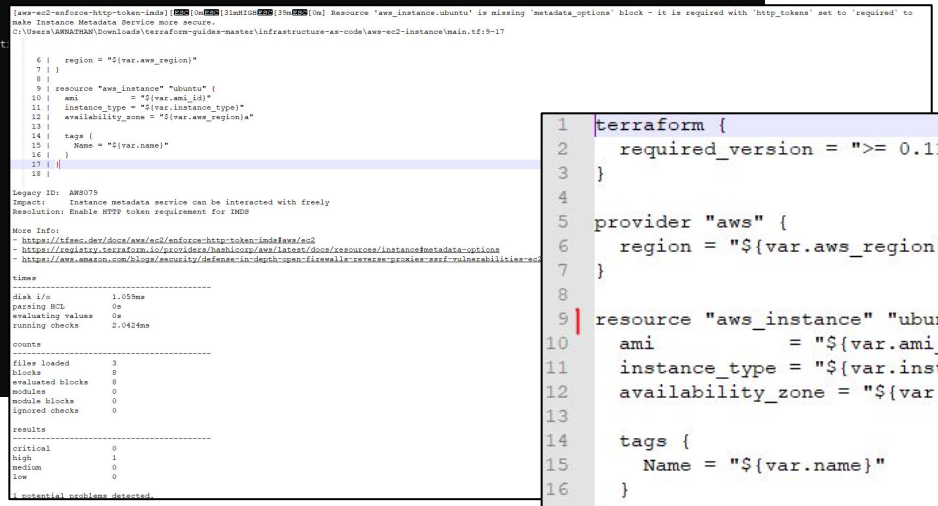
Aqua tfsec- Demo

```
C:\Users\AWNATHAN>cd C:\Users\AWNATHAN\Downloads\terraform-guides-master\infrastructure-as-code\aws-ec2-instance

C:\Users\AWNATHAN\Downloads\terraform-guides-master\infrastructure-as-code\aws-ec2-instance>tfsec

 Result 1

 [aws-ec2-enforce-http-token-imds][0m[31mHIGH[39m[0m] Resource 'aws_instance.ubuntu' is missing `metadata_options` block - it is required with `http_tokens` set to `required` to make Instance Metadata Service more secure.
 C:\Users\AWNATHAN\Downloads\terraform-guides-master\infrastructure-as-code\aws-ec2-instance\main.tf:9-17


      6 |   region = "${var.aws_region}"
      7 | }
      8 |
      9 | resource "aws_instance" "ubuntu" {
     10 |   ami           = "${var.ami_id}"
     11 |   instance_type = "${var.instance_type}"
     12 |   availability_zone = "${var.aws_region}a"
     13 |
     14 |   tags {
     15 |     Name = "${var.name}"
     16 |   }
     17 | }
     18 |

 Legacy ID:  AWS079
 Impact:     Instance metadata service can be interacted with freely
 Resolution: Enable HTTP token requirement for IMDS

 More Info:
 - https://tfsec.dev/docs/aws/ec2/enforce-http-token-imds#aws/ec2
 - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance#metadata-options
 - https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilit

 times
 ------------------------------------------
 disk i/o             10.276ms
 parsing HCL          0s
 evaluating values    0s
 running checks       1.5688ms

 counts
 ------------------------------------------
 files loaded         3
 blocks               8
 evaluated blocks     8
 modules              0
 module blocks        0
 ignored checks       0

 results
 ------------------------------------------
 critical             0
 high                 1
 medium               0
 low                  0

 1 potential problems detected.

C:\Users\AWNATHAN\Downloads\terraform-guides-master\infrastructure-as-code\aws-ec2-instance>_
```

tfsec uses static analysis of your terraform templates to spot potential security issues. Now with terraform CDK support.

cd C:\Users\AWNATHAN\Downloads\terraform-guides-master\infrastructure-as-code\aws-ec2-instance

tfsec > issues.txt



```
terraform {
  required_version = ">= 0.11.0"
}

provider "aws" {
  region = "${var.aws_region}"
}

resource "aws_instance" "ubuntu" {
  ami           = "${var.ami_id}"
  instance_type = "${var.instance_type}"
  availability_zone = "${var.aws_region}a"

  tags {
    Name = "${var.name}"
  }
}
```

https://github.com/aquasecurity/tfsec; https://tfsec.dev/

Recap

Challenge

"The perimeter is no longer just the physical location of the enterprise [data center], and what lies inside the perimeter is no longer a blessed and safe place to host personal computing devices and enterprise applications [microservices]." https://cloud.google.com/security/beyondprod/

Traditional Infrastructure Security	Cloud-Native Security	Implied Requirements for Cloud-native Security
Perimeter-based security (i.e. firewall), with internal communications considered trusted.	**Zero-trust security with service-to-service communication verified, and no implicit trust for services in the environment.**	***Protection of the network at the edge (remains applicable) and no inherent mutual trust between services.***
Fixed IPs and hardware for certain applications.	Greater resource utilization, reuse, and sharing, including of IPs and hardware.	***Trusted machines running code with known provenance.***
IP address-based identity.	Service based identity.	
Services run in a known, expected location.	Services can run anywhere in the environment, including hybrid deployments across the public cloud and private data centers.	

SOURCE: https://cloud.google.com/security/beyondprod/

"Are microservices more or less secure than monoliths?" → *It depends!*

(Sorry to disappoint!)

Multiple Considerations + Tradeoffs

Attack Surfaces

An attack surface is the sum of vulnerabilities in an application — the smaller it is, the better. In a monolith, there is only one attack surface, albeit a sizable one. With microservices, however, each service has its own attack surface, but of lesser size.

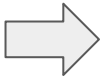
+ 1 for Monolithic

Cascading Vulnerabilities

The individual parts of a monolith are inherently more connected than the services in a microservices architecture. This means that if one aspect of a monolith is exploited or otherwise fails, others may follow. On the other hand, microservices are decoupled by design. While this does mean there are more attack surfaces to secure, the likelihood of a cascading vulnerability (especially polyglot is pursued!) across multiple aspects of the application is much lower. Not to mention, decoupling also makes it easier to address individual issues and roll upgrades without shutting down the entire app.

+ 1 for Microservices

SOURCE: https://blog.sqreen.com/top-10-security-traps-to-avoid-when-migrating-from-a-monolith-to-microservices/; https://nordicapis.com/which-is-more-secure-monolith-or-microservices/

“Are microservices more or less secure than monoliths?”  ***It depends!***

(Sorry to disappoint!)

All things considered

Both microservices and monoliths have pros and cons in terms of security. However, monolithic approach may make security easier for smaller applications.

Some usual security pitfalls observed in Microservices Adoption:

1. Inconsistent logging
2. Relying on one external firewall
3. Not monitoring your service--to-service communication

- Make Observability in Microservices your first-class citizen and foremost concern! (Envoy Proxy, Service Mesh such as Istio, etc)
- Multiple Cloud Native Platform and Tools are available out there to help you secure your Microservices! (Some are demonstrated today.)

References

- https://cloud.google.com/security/beyondprod
- https://nordicapis.com/which-is-more-secure-monolith-or-microservices/
- https://deepfence.io/
- https://www.curiefense.io/
- https://sysdig.com/
- https://www.aquasec.com/

Reach Out

https://www.linkedin.com/in/awnathan
nathan.mk.aw@gmail.com