

# **Enabling Zero Trust Architecture (ZTA) with Least-Privilege Identity-Based Micro-segmentation using Service Mesh and Securing Production Identity Framework for Everyone (SPIFFE) (Demo) & Building Secure Software Factory (SSF) to Defend the Digital Cloud-Native Software Supply Chain against attacks: Helpful Cloud-Native Security Checklists and Demo on The Update Framework**

Nathan Aw

<https://www.linkedin.com/in/awnathan>

[nathan.mk.aw@gmail.com](mailto:nathan.mk.aw@gmail.com)

<https://nathanawmk.github.io/>

19 Oct 2021 (Tue)

“Zero trust is a concept, not an action.” - KEN WESTIN, SECURITY RESEARCHER

“Zero trust is not a technology, it’s not something you buy, it’s a **strategy**.” - Gregory Touhill, Director of the Computer Emergency Readiness Team Carnegie Mellon University Former CISO in the Obama Administration

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

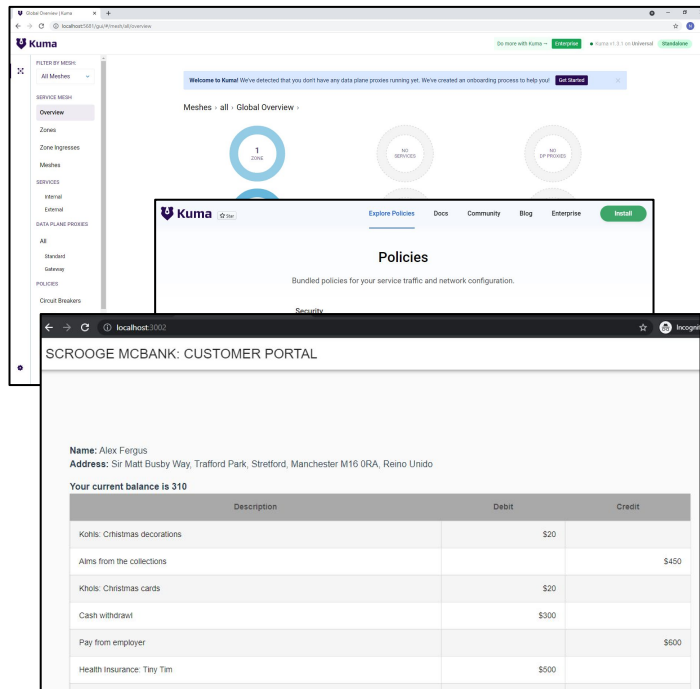
# Today's Presentation Split into Two Major Parts (1/2):

## 1. Enabling Zero Trust Architecture (ZTA) with Least-Privilege, Identity-Based Micro-segmentation using Securing Production Identity Framework for Everyone (SPIFFE) and Service Mesh

- a. **Demo** on SPIFFE X.509 IDs with Envoy and Open Policy Agent Authorization
- b. **Demo** on Kuma Mesh → Explore Policies → mTLS (SPIFFE compatible)
  - i. <http://localhost:5681/gui/#/mesh/all/overview>

## 2. Building an end-to-end Secure Software Factory

- a. Some Handy Principles, Checklists and Guidance
  - i. <https://github.com/OWASP/DevSecOpsGuideline>
  - ii. <https://github.com/ukncsc/zero-trust-architecture>
  - iii. <https://github.com/ukncsc/secure-development-and-deployment>
- b. End-to-end Secure Software Factory - How it may look like
- c. **Demo** on The Update Framework (TUF)



### References:

<https://kuma.io/>  
<https://spiffe.io/>  
<https://theupdateframework.io/>

# **Context & Problem Statement**

# Today's Presentation Split into Two Major Parts (1/2):

1

## Enabling Zero Trust with Zero Trust Architecture (ZTA)

### Context: The “WHY”

In today's world, a single enterprise may operate several internal networks, remote offices with their own local infrastructure, remote and/or mobile individuals, and cloud services. This complexity has outstripped legacy methods of perimeter-based network security as there is no single, easily identified perimeter for the enterprise. Perimeter-based network security has also been shown to be insufficient since once attackers breach the perimeter, further lateral movement is unhindered.

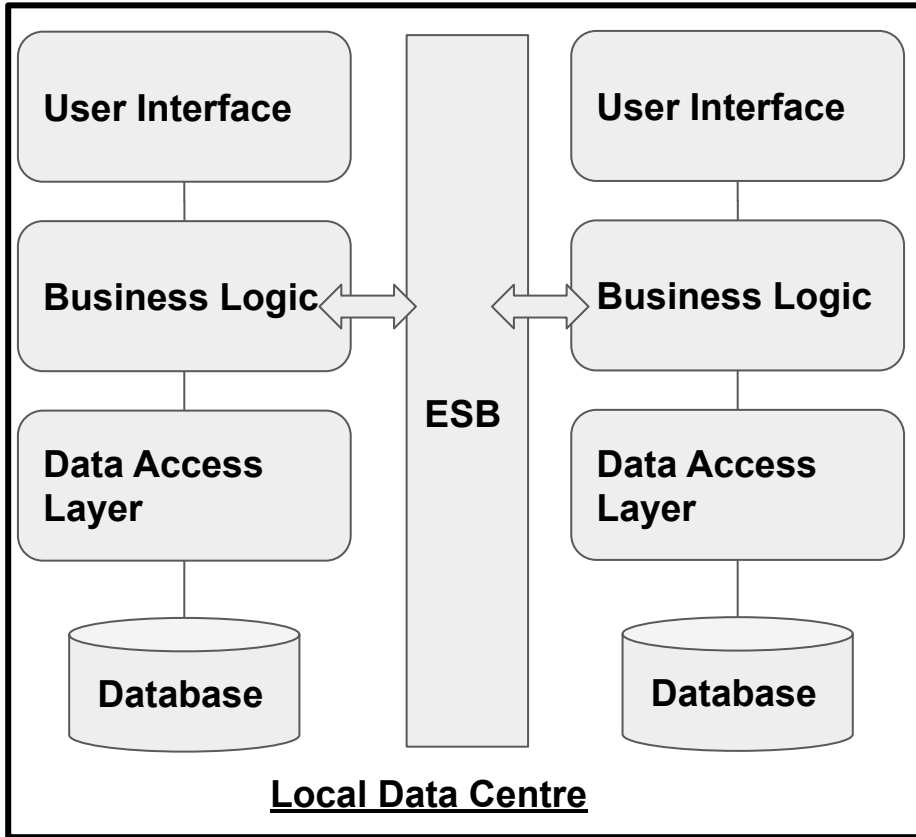
Coupled with the adoption of cloud-native microservices and Event Brokers (e.g., Kafka) and the corresponding increase of traffic flowing east-west (service to service interaction) -- instead of north-south traffic flow (i.e., client-server interaction) -- has further reduced the relevance of perimeter-based network security.

Zero trust (ZT) is an evolving set of cybersecurity paradigms that move defenses from static, network-based perimeters to focus on users, assets, and resources. Zero trust assumes there is no implicit trust granted to assets or user accounts based solely on their physical or network location (i.e., local area networks versus the internet) or based on asset ownership (enterprise or personally owned). Never Trust - Always Verify Paradigm is necessary in today's new world.

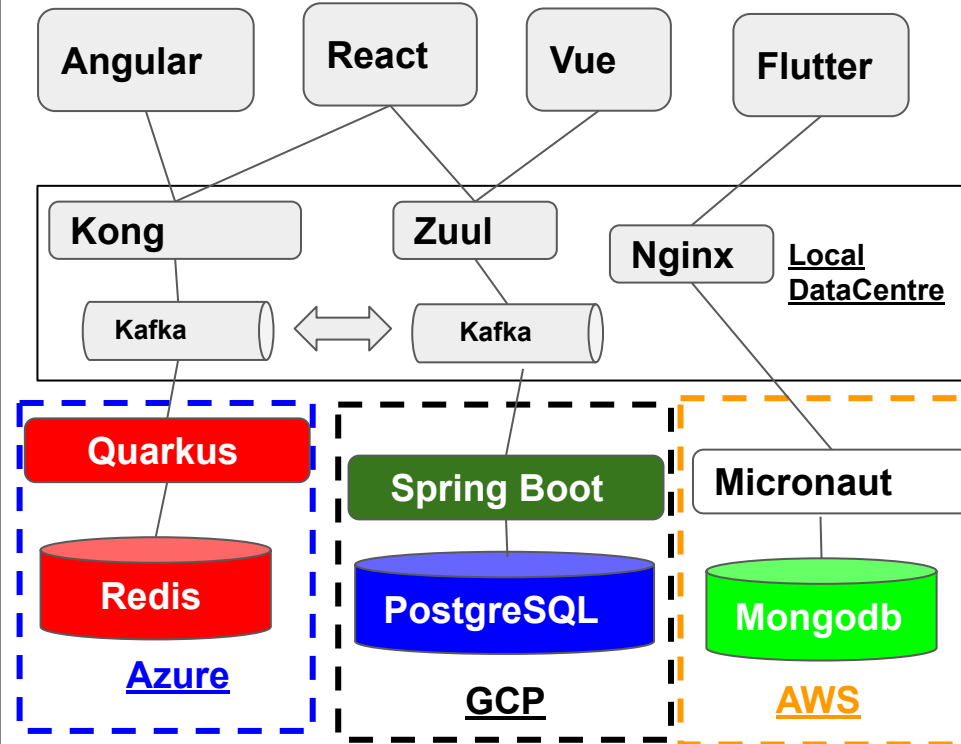


**Security in a cloud-native world requires an urgent fundamental rethink, a radical pivot in approach from one that is fortress/perimeter based to Zero Trust with Identity-based Micro-segmentation**

# Recap: Monolithic vs Microservices/Cloud-Native Architecture



N-Tier / Multi-Tier Architecture:  
Perimeter is well-defined



In a hybrid, multi-cloud world, microservices \*-tier architecture, Perimeter defenses can't stop every attack; the attack surface is simply too wide -- the perimeter too porous. Compartmentalization needed to contain attackers, and the visibility and control to detect and respond.

# 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

Perimeter-based security (i.e. firewall), with internal communications considered trusted.



### Cloud-Native Security

**Zero-trust security with service-to-service communication verified, and no implicit trust for services in the environment.**



### Implied Requirements for Cloud-native Security



***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.

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.



***Trusted machines running code with known provenance.***

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

# Today's Presentation Split into Two Major Parts (2/2):

2

## Context - The "WHY"

From the **2021 State of the Software Supply Chain**, In 2021 the world witnessed a **650%** increase in software supply chain attacks, aimed at exploiting weaknesses in upstream open source ecosystems. For some perspective, the same statistic was 430% in the 2020 version of the report.

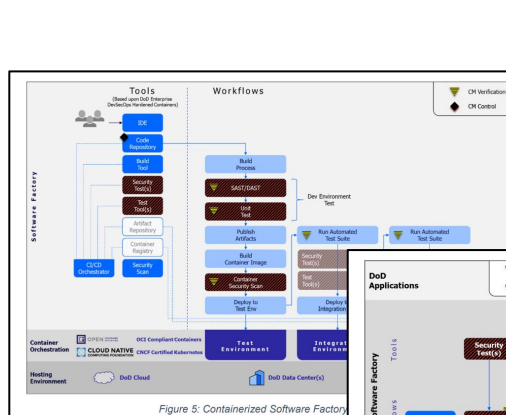


Figure 5: Containerized Software Factory

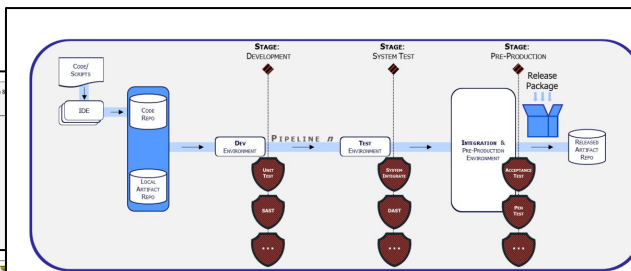
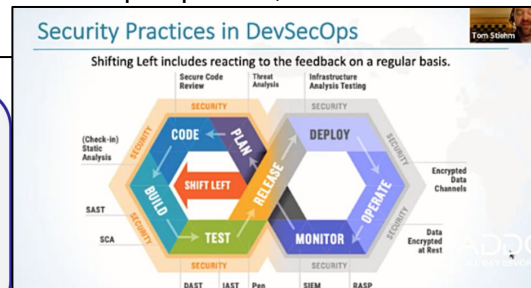
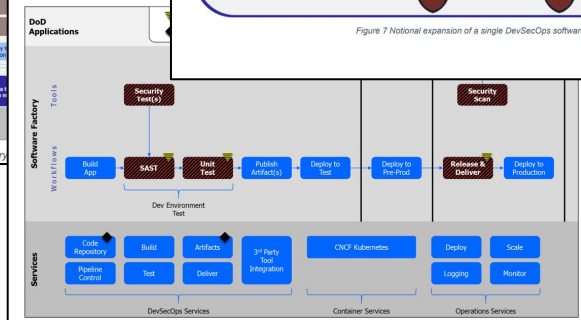
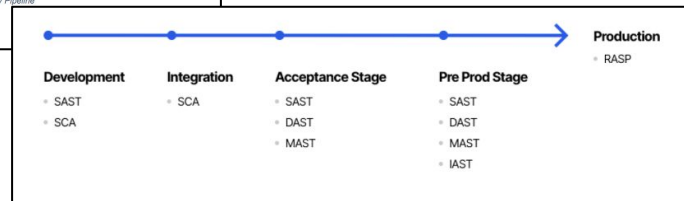


Figure 7: Notional expansion of a single DevSecOps software factory Pipeline



Shifting Left includes reacting to the feedback on a regular basis.



To defend against software supply chain attacks, a trusted secure end-to-end software factory is an imperative.

SOURCE: [https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing\\_Multi\\_cloud\\_Portable\\_Tier\\_Microservices\\_Applications\\_A\\_live\\_demo\\_on\\_cloud\\_native\\_application\\_security\\_platforms.pdf](https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing_Multi_cloud_Portable_Tier_Microservices_Applications_A_live_demo_on_cloud_native_application_security_platforms.pdf); <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf> ; <https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsReferenceDesign.pdf>

# Who I am

Over 9 years experience as a microservices developer with a keen interest in AppSec and emerging frameworks + frontier technologies. Currently, working in an **end-user environment** at a financial institution.

A firm believer and practitioner of **zero trust** plus advocate of secure coding practices, my passion and current focus is to build and rollout **asynchronous polyglot-based microservices that are both zero-trust, performant** which can securely run anywhere (multi-cloud and/or on-premise) and scale without limits.

Through actual hands-on setup of a Secure Software Factory (SSF), understand the importance of setting up a first-class secure software factory that is able to industrialise “shift left” practises that translates to quicker delivery of trusted and secure digital services to its customers. More at <https://nathanawmk.github.io/>, <https://sg.linkedin.com/in/awnathan>

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

[https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing\\_Multi\\_cloud\\_Portable\\_Tier\\_Microservices\\_Applications\\_A\\_live\\_demo\\_on\\_cloud\\_native\\_application\\_security\\_platforms.pdf](https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing_Multi_cloud_Portable_Tier_Microservices_Applications_A_live_demo_on_cloud_native_application_security_platforms.pdf);

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



# Zero Trust, Micro-segmentation and Service Mesh

## (1/3)

### Zero Trust & Zero Trust Architecture

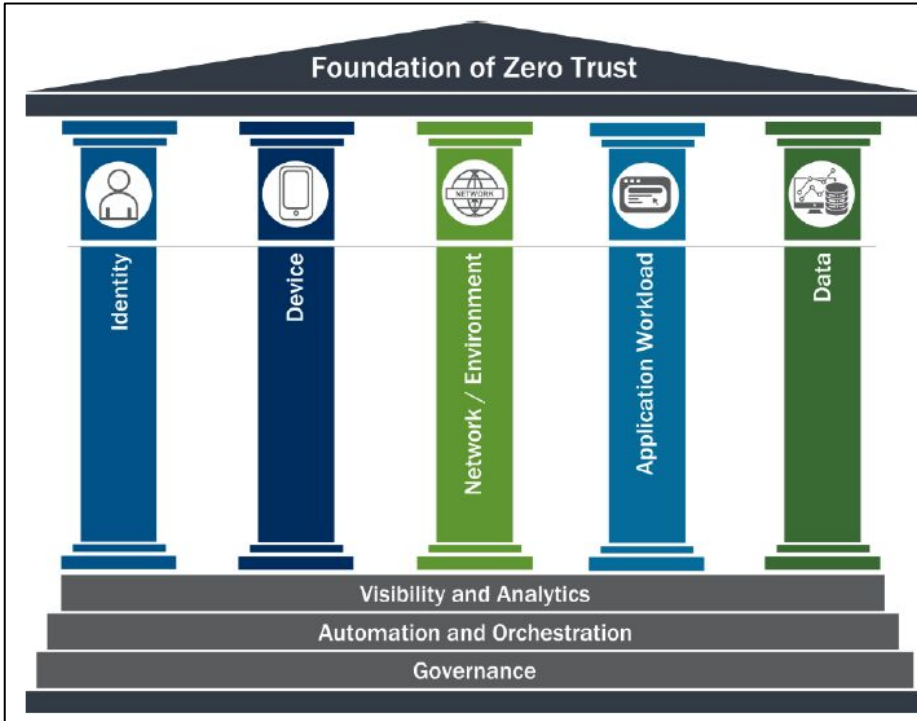
Zero trust (ZT) is the term for an **evolving set of cybersecurity paradigms** that move defenses **from static, network-based perimeters to focus on users, assets, and resources.**

A zero trust architecture (ZTA) uses zero trust principles to plan industrial and enterprise infrastructure and workflows. Zero trust assumes there is **no implicit trust** granted to assets or user accounts based solely on their physical or network location (i.e., local area networks versus the internet) or based on asset ownership (enterprise or personally owned).

**Authentication and authorization (both subject and device) are discrete functions performed before a session to an enterprise resource is established.**

Zero trust focuses on protecting resources (assets, services, workflows, network accounts, etc.), not network segments, as the network location is no longer seen as the prime component to the security posture of the resource.

# Foundation of Zero Trust: Zero Trust Maturity Model






	Identity	Device	Network / Environment	Application Workload	Data
Traditional	<ul style="list-style-type: none"> <li>Password or multifactor authentication (MFA)</li> <li>Limited risk assessment</li> </ul>	<ul style="list-style-type: none"> <li>Limited visibility into compliance</li> <li>Simple inventory</li> </ul>	<ul style="list-style-type: none"> <li>Large macro-segmentation</li> <li>Minimal internal or external traffic encryption</li> </ul>	<ul style="list-style-type: none"> <li>Access based on local authorization</li> <li>Minimal integration with workflow</li> <li>Some cloud accessibility</li> </ul>	<ul style="list-style-type: none"> <li>Not well inventoried</li> <li>Static control</li> <li>Unencrypted</li> </ul>
	Visibility and Analytics   Automation and Orchestration   Governance				
Advanced	<ul style="list-style-type: none"> <li>MFA</li> <li>Some identity federation with cloud and on-premises systems</li> </ul>	<ul style="list-style-type: none"> <li>Compliance enforcement employed</li> <li>Data access depends on device posture on first access</li> </ul>	<ul style="list-style-type: none"> <li>Defined by ingress/egress micro-perimeters</li> <li>Basic analytics</li> </ul>	<ul style="list-style-type: none"> <li>Access based on centralized authentication</li> <li>Basic integration into application workflow</li> </ul>	<ul style="list-style-type: none"> <li>Least privilege controls</li> <li>Data stored in cloud or remote environments are encrypted at rest</li> </ul>
	Visibility and Analytics   Automation and Orchestration   Governance				
Optimal	<ul style="list-style-type: none"> <li>Continuous validation</li> <li>Real time machine learning analysis</li> </ul>	<ul style="list-style-type: none"> <li>Constant device security monitor and validation</li> <li>Data access depends on real-time risk analytics</li> </ul>	<ul style="list-style-type: none"> <li>Fully distributed ingress/egress micro-perimeters</li> <li>Machine learning-based threat protection</li> <li>All traffic is encrypted</li> </ul>	<ul style="list-style-type: none"> <li>Access is authorized continuously</li> <li>Strong integration into application workflow</li> </ul>	<ul style="list-style-type: none"> <li>Dynamic support</li> <li>All data is encrypted</li> </ul>
	Visibility and Analytics   Automation and Orchestration   Governance				

The Zero Trust Maturity Model represents a gradient of implementation across five distinct pillars, where minor advancements can be made over time toward optimization. The pillars, depicted here, include **Identity**, **Device**, **Network**, **Application Workload**, and **Data**. Each pillar also includes general details regarding Visibility and Analytics, Automation and Orchestration, and Governance. This maturity model (**Traditional** → **Advanced** → **Optimal**) is one of many paths to support the transition to zero trust.

SOURCE: [https://www.cisa.gov/sites/default/files/publications/CISA%20Zero%20Trust%20Maturity%20Model\\_Draft.pdf](https://www.cisa.gov/sites/default/files/publications/CISA%20Zero%20Trust%20Maturity%20Model_Draft.pdf)

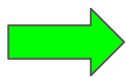
# Zero Trust, Micro-segmentation and Service Mesh (2/3)

	<b>Applications and data</b>	Data classification, authentication and authorization of microservices, container security and cross-system integration via APIs
	<b>The network</b>	Automation, microsegmentation, stateful session management, encryption and secure routing, virtualization, firewalls, SD-WAN and SASE
	<b>User and device identity</b>	Biometrics, multifactor authentication, identity and access management, and device certification



**Secure access to applications and infrastructure using workload/ application identity rather than IP addresses**

*This is not a comprehensive list of concepts that enable Zero Trust*



**Zero Trust in Cloud-Native, Microservices-based Applications through Micro-segmentation with SPIFFE and Service Mesh is the focus today**

# Zero Trust, Micro-segmentation and Service Mesh (3/3)

## Micro-segmentation

Micro-segmentation is a security technique that creates secure zones in cloud deployments and allows organizations to isolate workloads from one another and secure them individually.

Micro-segmentation policies can take many forms, including controls based on environment type, regulatory scope, application, and infrastructure tier.

**Workload identity** is the key element that enables **Zero Trust** with Identity-Based Microsegmentation.

### Some micro-segmentation/segmentation examples:

1. **Istio's** authorization feature, also known as Istio Role Based Access Control, provides micro-segmentation for services in an Istio mesh
2. AWS provides segmentation capabilities using built-in **security groups**
3. **Prisma Cloud Identity-Based Microsegmentation** assigns every protected host and container with a cryptographically signed workload identity
4. Illumio Core delivers live visibility and micro-segmentation that works on anything (virtual machines, bare metal, and containers), anywhere (data center, private or public cloud) by activating and centrally managing the native security controls in the workload.



**Workload/Application Identity enables Zero Trust!**

<https://docs.paloaltonetworks.com/prisma/prisma-cloud/prisma-cloud-admin/get-started-with-prisma-cloud/prisma-cloud-how-it-works.html>;  
[https://cdn2.hubspot.net/hubfs/407749/Downloads/Illumio\\_Article\\_SecurityWeek\\_The\\_Truth\\_About\\_Micro-Segmentation\\_2018\\_04.pdf](https://cdn2.hubspot.net/hubfs/407749/Downloads/Illumio_Article_SecurityWeek_The_Truth_About_Micro-Segmentation_2018_04.pdf);  
<https://istio.io/latest/blog/2018/istio-authorization/>; <https://aws.amazon.com/blogs/apn/improving-security-in-the-cloud-with-micro-segmentation/>;  
<https://www.paloaltonetworks.com/prisma/cloud/identity-based-microsegmentation/>; <https://www.paloaltonetworks.com/blog/prisma-cloud/aporeto-integration-prisma-cloud/>;  
<https://www.paloaltonetworks.com/prisma/cloud/cloud-network-security/>; <https://d1.awsstatic.com/Marketplace/solutions-center/downloads/Aporeto-Datasheet-final.pdf>

# Micro-Segmentation vs Zero Trust

Micro-segmentation	Zero Trust
<ul style="list-style-type: none"><li>• A security technique/approach to achieve Zero Trust</li><li>• An architectural design that serves a foundation for Zero Trust</li><li>• Software - e.g., Prisma Cloud - Aporeto, illumio</li></ul>	<ul style="list-style-type: none"><li>• A principle of “never trust, always verify,”</li><li>• A paradigm or model that assumes breach</li></ul>

SOURCE:

<https://ipwithease.com/microsegmentation-vs-zero-trust/>

<https://www.paloaltonetworks.com/blog/prisma-cloud/aporeto-integration-prisma-cloud/>

<https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture>

<https://www.illumio.com/sites/default/files/2021-02/micro-segmentation-secure-beyond-breach-20eb10%20%281%29.pdf>

# Zero Trust, Micro-segmentation and the Service Meshes



## Network Service Mesh

The Hybrid/Multi-cloud IP Service Mesh

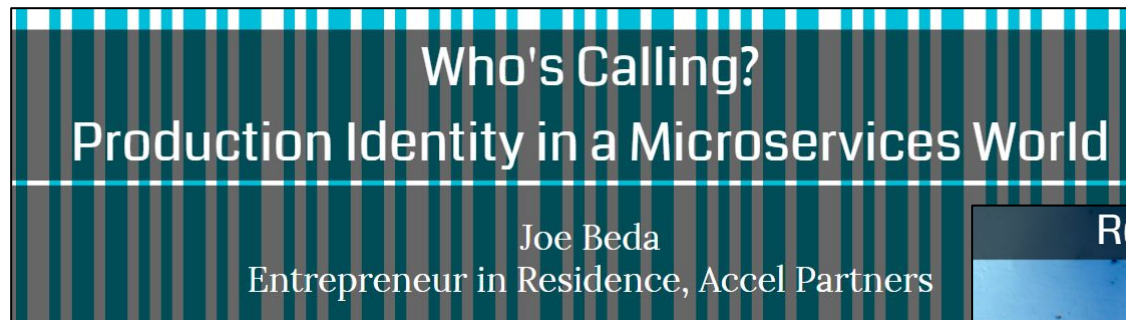


Istio-based service mesh & gateway

1. **Minimizes the attack surface:** Microsegmentation, when implemented correctly, enables the verification of the identities of all services before allowing communication between them. It also checks that this communication is part of recognized workflows. This makes it an effective response to the proliferation of threats within the production network and the increased points of privileged access.
2. **Provides policy-driven network security:** Microsegmentation allows DevOps to apply a shift-left security approach for containerized applications by attaching (or baking in) an identity to an application or container image as early as possible during the CI/CD pipeline stage.
3. **Decouples security from infrastructure:** Enforcing security policies has no impact on the network and therefore cannot break the network. The workload is moved to a different infrastructure, and because its identity and the policy rules don't change there is no need to maintain policy on these changes.
4. **Visibility:** Service mesh increases the visibility of running workloads and the communication between them. Service meshes provide connection telemetry and allow topology visualization based on these telemetries to deliver a clear insight of all connections inside the mesh.

SOURCE: <https://www.portshift.io/blog/microsegmentation-cloud-native-zero-trust/>

# “Who’s calling? Production Identity in a Microservices World.” Does Reachability implies authorization?



Joe Beda is one of the co-creator of Kubernetes



 **Does Reachability implies authorization? Certainly not!**

SOURCE: <http://slides.eightypercent.net/spiffe-intro/index.html#1>; <https://www.vmware.com/asean/company/leadership/joe-beda.html>

# General Definitions

- **Identity** – A characteristic or a set of data to uniquely identify a user, process or device. Sometime expressed with an Identity document (e.g. SVID - SPIFFE Verifiable Identity Document)
- **Authn** – Authentication – The verification of the identity of a user, process, or device.
  - Transport or Peer Authn – Authentication with the immediate peer of the connection even when acting as an intermediary of a flow. Supported by most service meshes.
  - Origin or Request Authn - Authn based on the source and destination of the flow (ie. request) regardless of intermediaries. Not widely supported by service meshes.
- **Authz** – Authorization – The determination of the access levels or privileges that should be granted to an identity.
- **Certificate** – Contains a public key and an identity. Typically X.509 in this space.
- **SVID** – SPIFFE Verifiable Identity Document – An identity document defined in the SPIFFE specification.
- **Transport Layer Security (TLS)** – A cryptographic protocol designed to provide communications security over a computer network.
- **Zero Trust** – Security concepts that assume no peers can be trusted regardless of whether they appear to be within the same network or security domain.

SOURCE: “*Security in the World of Service Meshes*”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)



# Definitions – TLS flavors

- **Mutual TLS (mTLS)** – A form of TLS where both sides authenticate the identity of the peer. The “gold standard” for service mesh implementations of peer Authn. Not yet supported by all service meshes.
- **Server (side) TLS** – A form of TLS where the server presents a certificate to the client which is used to authenticate the server’s identity. Commonly used in “web surfing” scenarios as the clients are unknown or outside of server’s security domain.
- **Client (side) TLS** – A form of TLS where the client presents a certificate to the upstream server which is used to authenticate the client’s identity. Least commonly used variant.

SOURCE: “*Security in the World of Service Meshes*”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# References

- **Network Service Mesh** – <https://networkservicemesh.io/>
- **Linkerd** – <https://linkerd.io/>
- **Istio** – <https://istio.io/>
- **SPIFFE** – <https://spiffe.io/>
- **SPIRE** - <https://github.com/spiffe/spire>
- **X.509** - <https://www.itu.int/rec/T-REC-X.509>
- **Transport Layer Security (TLS)** – <https://tools.ietf.org/html/rfc8446>

**Note:** Much of the content in this talk was directly or indirectly derived from the above sources.

SOURCE: “Security in the World of Service Meshes”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Establishing Identity in Service Mesh

# Service Identities – The starting point

- In a service mesh world, establishing the identity of the workload providing a service is critical. Examples:
  - **Kubernetes**: Kubernetes service account
  - **GKE/GCE**: may use GCP service account
  - **GCP**: GCP service account
  - **AWS**: AWS IAM user/role account
  - **On-premises (non-Kubernetes)**: user account, custom service account, service name, Istio service account, or GCP service account. The custom service account refers to the existing service account just like the identities that the customer's Identity Directory manages.

<https://istio.io/docs/concepts/security/#istio-identity>

The new kid on the block for establishing identity is SPIFFE (specification) and SPIRE (implementation) – more on that below

SOURCE: “Security in the World of Service Meshes”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Conversion of that identity into a certificate

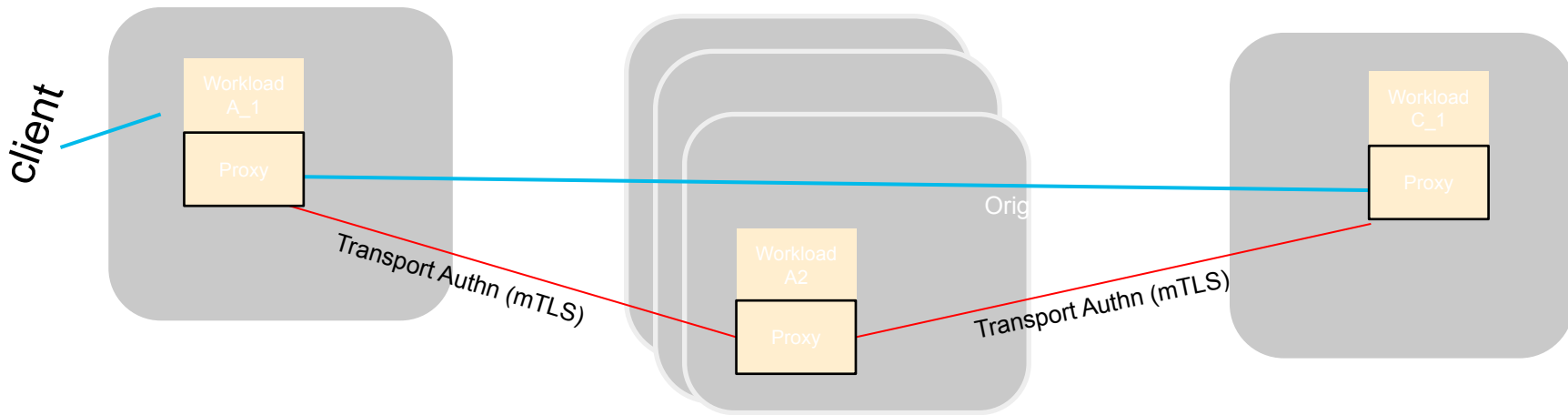
- A private key within the workload pod is generated and made available to the proxy.
- A certificate signing request is sent to the control plane.
- The control plane signs either with a self-signed root certificate or an external source (e.g. vault).
- The control plane provides the proxy a certificate scoped to the identity of the POD (e.g. K8s service-account).
- Control plane will manage rotation.
- Some meshes may use certificates mounted to the proxies via other means.

# AuthN - Authentication

# Authentication Layers

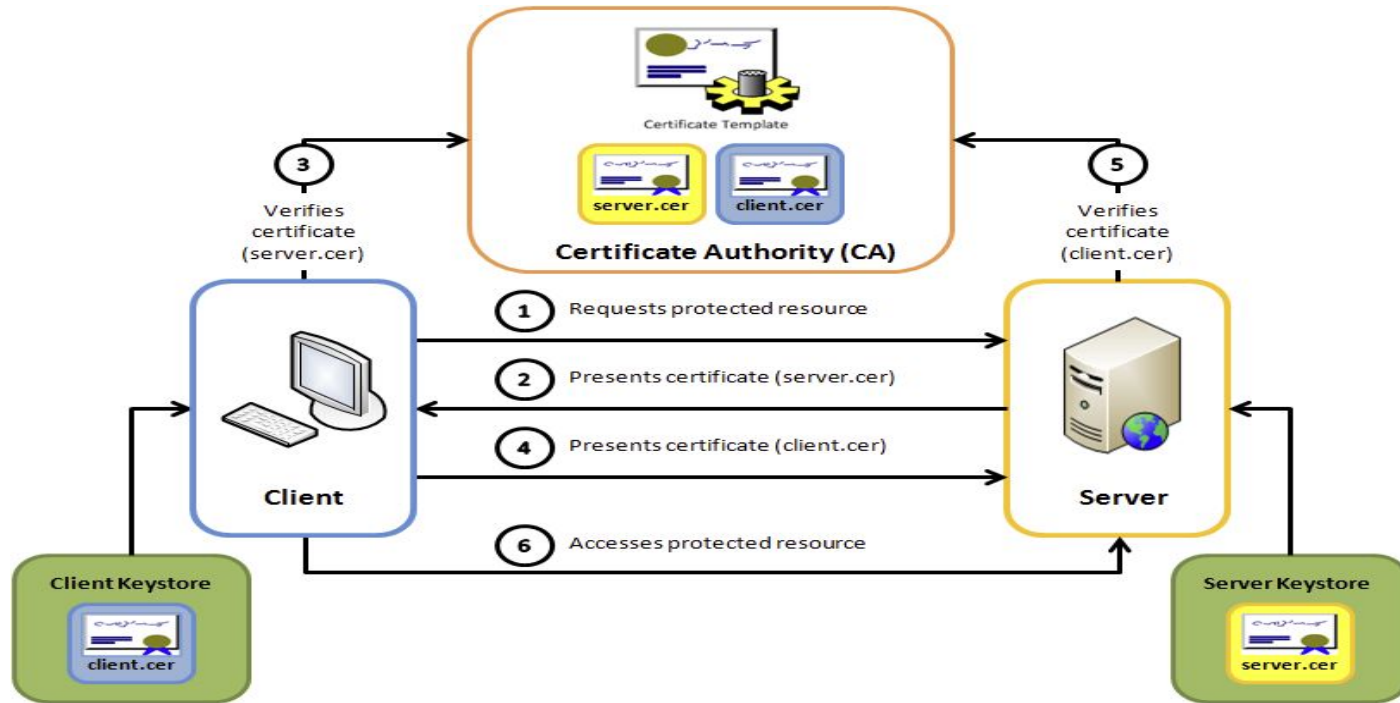
Peer or Transport Authn—tied explicitly to the network layer parameters/proto  
e.g. mTLS handshake certificate content

Request or Origin Authn—ability to correlate the origin of the flow (ie. request)  
regardless of intermediaries



All meshes offer some form of Peer/Transport Authentication. Some meshes additionally support Request/Origin Auth in the proxy. Request/Origin auth can still be done by the workload without participation by the mesh or proxy.

# Authn sequence for mTLS



**Mutual SSL authentication / Certificate based mutual authentication**



# Single-sided TLS vs. mTLS

- Although mutual TLS is the gold standard other options must be supported for several reasons:
  - It is necessary to accept traffic into the mesh from clients that can't be authenticated (e.g. clients not within the same administrative or security domain).
  - Similarly, it is necessary to send to clients or servers that can't be authenticated.
  - Multiple meshes without a common identity framework may need to share traffic.
- For these cases one sided TLS authentication is useful.
  - Server side is available as an option in most service meshes. The server side presents its identity via a certificate that is traceable back to a well known issuer.
  - Client side is also possible but not widely supported. The client side presents its identity, and the server is configured with a means to authenticate the client. Often this requires some amount of direct certificate population.
- Clear text is a final fall back – but diminishes one of the key benefits of using a service mesh in the first place.

SOURCE: “*Security in the World of Service Meshes*”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Authorization

- Once a peer has been authenticated, the next step is to decide if it is authorized.
- Authorization determines whether a sender and receiver have the necessary permissions or privileges to talk to one another.
- The authorization is often expressed in a policy and sometimes referred to as access control.
- To be effective there must be some means of coupling the authorization with the identities determined with Authn.
- Although common in service meshes, it is not universally supported in all service meshes.
- The authorization may be implemented in either the client-side proxy or server-side proxy or both.
- NOTE – Authorization here only covers Service Mesh authorization. The application may have its own authorization functionality independent of the service mesh.

SOURCE: “*Security in the World of Service Meshes*”

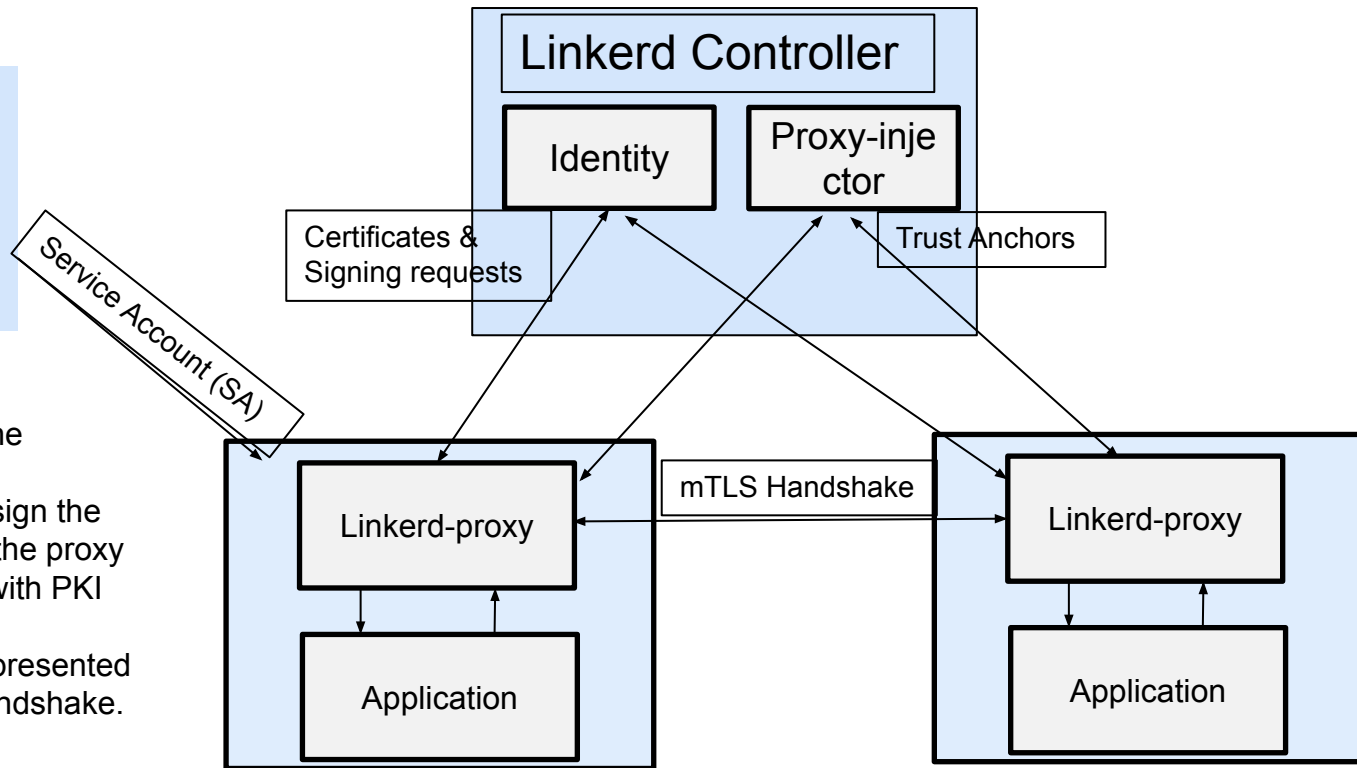
[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Components

# Authn Architecture (with Linkerd)



K8s

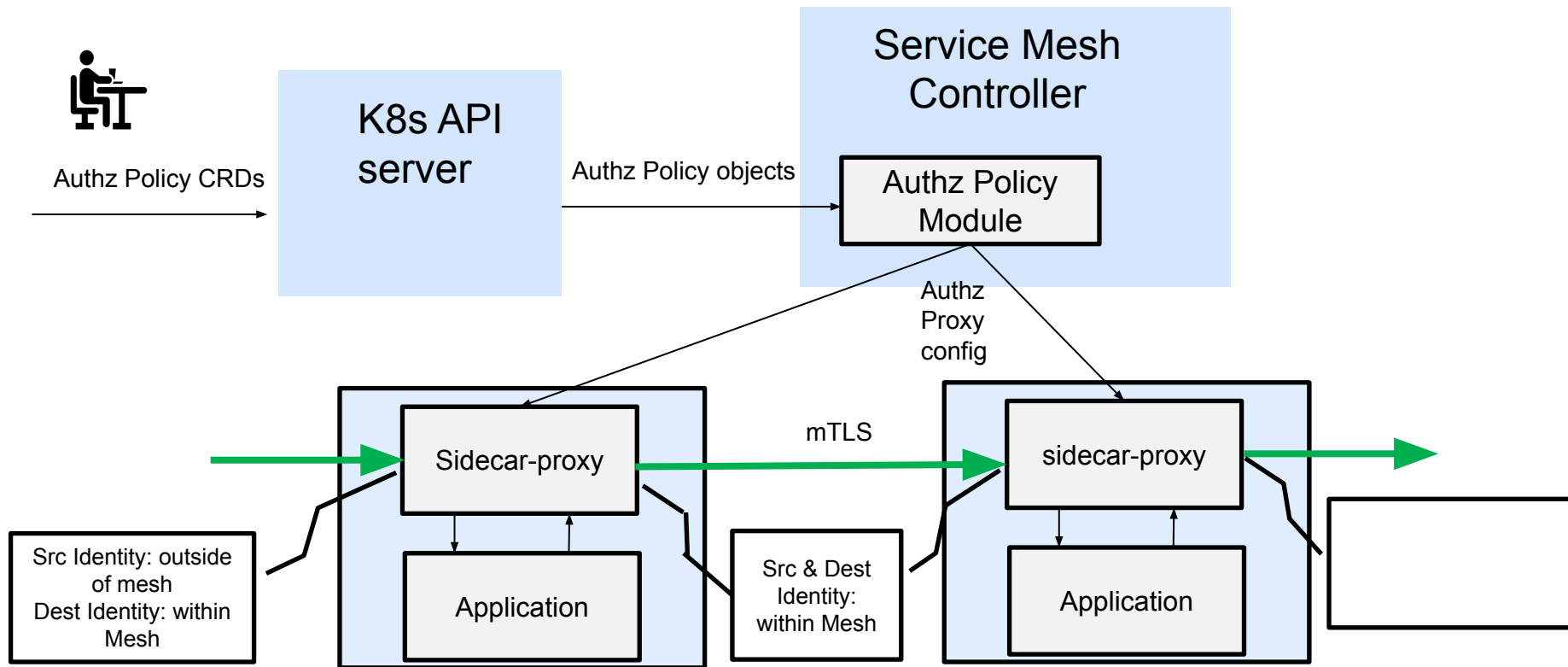


- Service Account provides the workload an Identity.
- The Linkerd Controller will sign the certificate and provide it to the proxy so the identity is verifiable with PKI techniques.
- The verifiable certificate is presented by both sides during the handshake.

SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Authorization Architecture



SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Authorization Example (From SMI)

```
kind: TrafficTarget
metadata:
  name: path-specific
  namespace: default
spec:
  destination:
    kind: ServiceAccount ← Destination identity specified with service account
    name: service-a
    namespace: default
    port: 8080
  rules:
    ← Match criteria
    - kind: HTTPRouteGroup
      name: the-routes
      matches:
        - metrics
  sources:
    - kind: ServiceAccount ← Source identity specified with service account
      name: prometheus
      namespace: default
```

SOURCE: “*Security in the World of Service Meshes*”

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Service Mesh Multi-cluster Identity and Trust Models

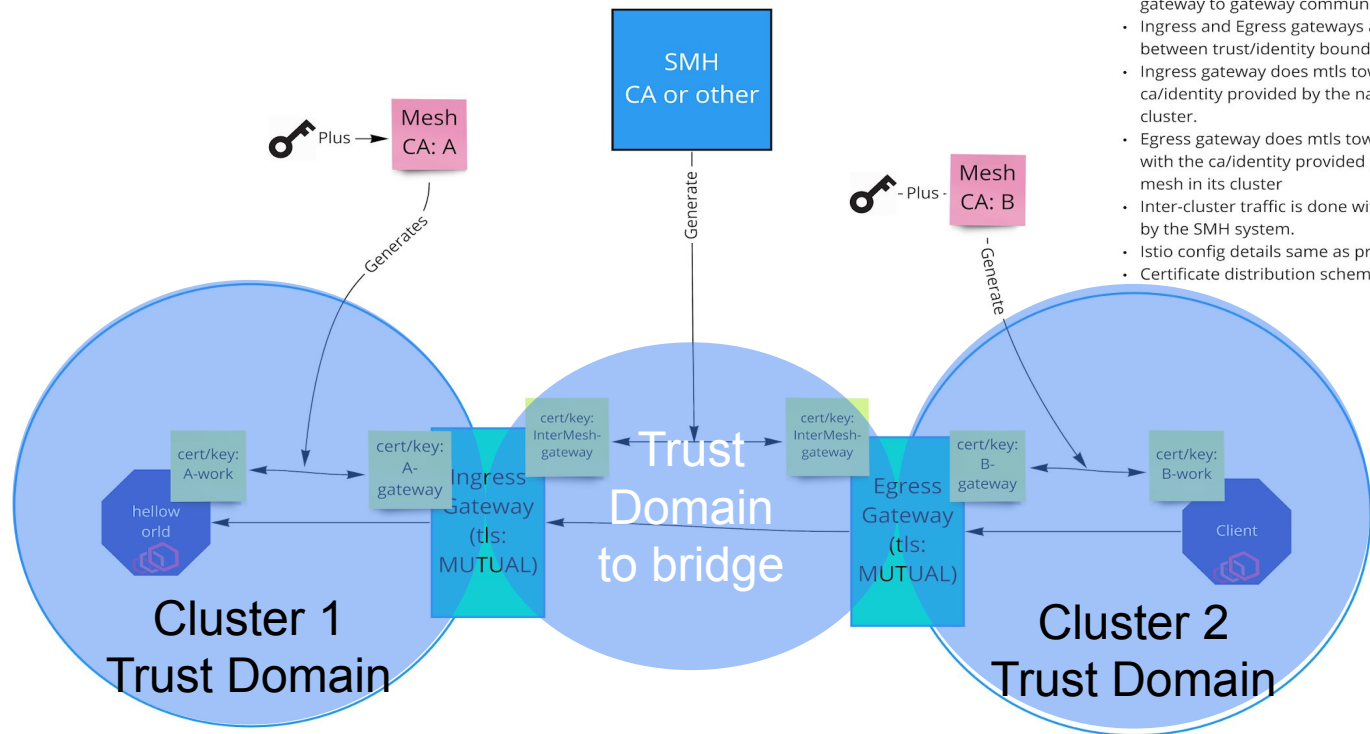
# Shared Root Trust

- The most common identity model used for Inter-cluster traffic in service mesh offerings when mTLS is supported between clusters.
- Nearly all service mesh implementations provide examples of how-to config this model.
- Relies on all certificates being traceable back to a common root so all parties can authenticate one another.



# Bridged trust via an intermediary

Via ServiceMeshHub and Istio

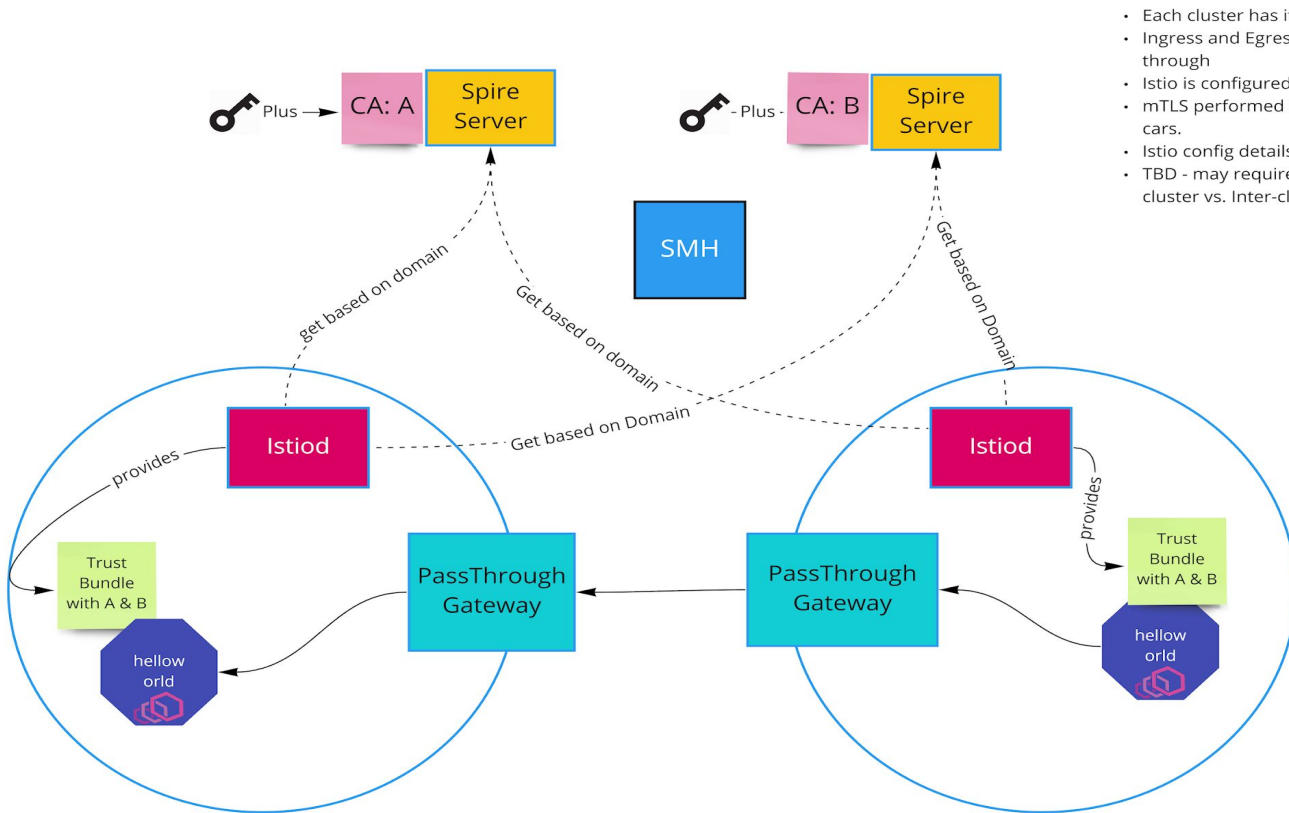


- Each cluster has its own CA system
- An additional CA or trust domain is created for gateway to gateway communication.
- Ingress and Egress gateways are the transition point between trust/identity boundaries
- Ingress gateway does mTLS toward upstream with the ca/identity provided by the native service mesh in its cluster.
- Egress gateway does mTLS toward client/downstream with the ca/identity provided by the native service mesh in its cluster
- Inter-cluster traffic is done with CA/identity provided by the SMH system.
- Istio config details same as prior
- Certificate distribution scheme is TBD

SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Federated Identity – with SPIFFE and SPIRE



- Each cluster has its own CA system
- Ingress and Egress gateways are set up for pass-through
- Istio is configured with a Spiffe endpoint per domain.
- mTLS performed directly in client and upstream side cars.
- Istio config details TBD
- TBD - may require different request path for in cluster vs. Inter-cluster requests.

SOURCE: "Security in the World of Service Meshes"  
[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# NSM's use of SPIFFE/SPIRE

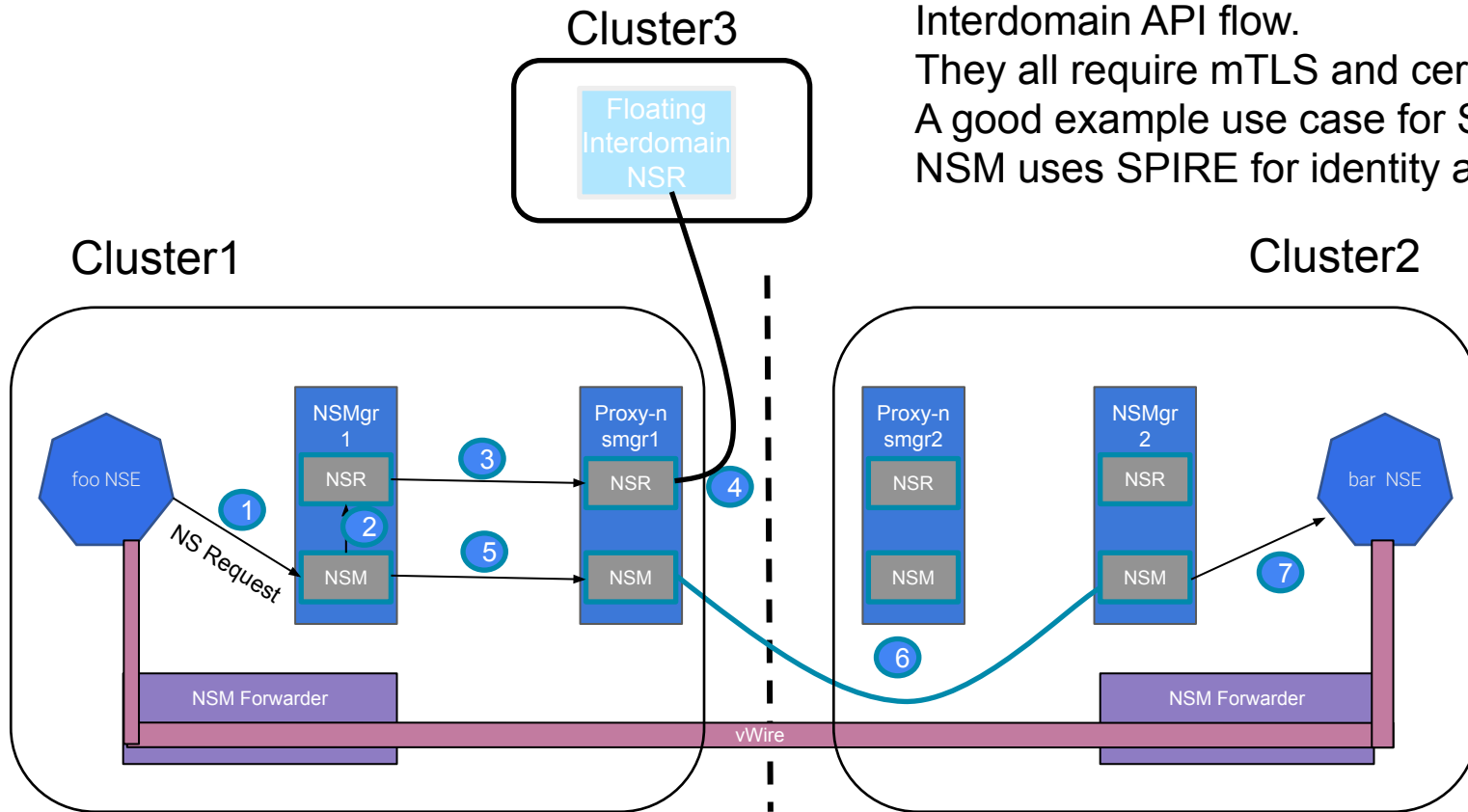
- The Network Service Mesh project use SPIRE servers to provide identity within its implementation.
- It is used to secure API communication between all clients and servers of the API.
- For those familiar with NSM these include NSCs, NSEs, NSRs, NSMs, etc.
- In the floating Interdomain use case the API communication crosses multiple cluster boundaries and potentially multiple security domains.
- As such, it is a great use case to display SPIFFE's and SPIRE's capabilities in a multi-cluster environment.

SOURCE: "*Security in the World of Service Meshes*"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# NSM Interdomain Flow

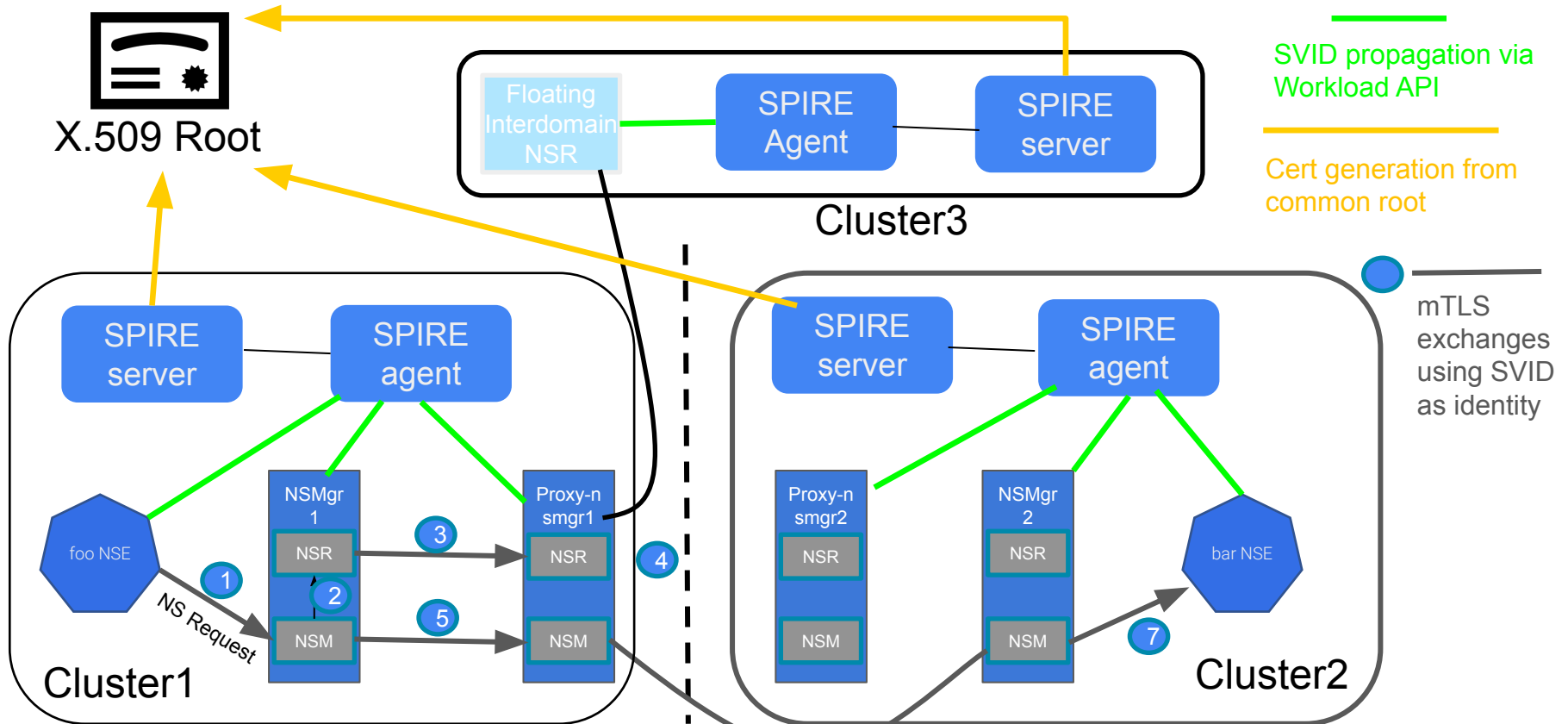
The flow below exemplifies an NSM floating Interdomain API flow. They all require mTLS and certifiable identity. A good example use case for SPIFFE/SPIRE. NSM uses SPIRE for identity and attestation



SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# NSM SPIRE Agent and Server topology



SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# SPIFFE/SPIRE: A new way to establish Identity



# What is SPIFFE

SPIFFE is a new set of APIs and associated tooling that provides a uniform language for describing service identity in a wide range of workloads (including orchestrated systems), verifying that identity, and providing a workload with documents that serve as proof of that identity. It is inspired in large part by Google's Low Overhead Authentication Service.

Historically, “service identity” is defined very differently depending on the organisation and the specific system they are running. A “service” for example may be an Apache web-server running on a set of Amazon EC2 VMs in an auto-scaling group, but might also be an application in Pivotal Cloud Foundry running in Google Compute Engine, or an Oracle database running on a HP Enterprise Proliant server.

SPIFFE provides a general purpose framework for defining and verifying service identity in a wide range of environments such as these, and ensuring services can easily and safely retrieve the identities that represent them. The verification SPIFFE performs is conducted both for the infrastructure the workload is running on (node level) and how that workload was provisioned onto the infrastructure (process level).

This is achieved with a set of coupled API driven components that vastly simplify and automate the process, ensuring an operator can employ industry best practices for establishing identity for a wide range of systems without needing to reason about the complexities of PKI infrastructure or node/process attestation.

SOURCE: Design Document: SPIFFE Reference Implementation (SRI)

# SPIFFE & SPIRE



- SPIFFE is the specification.
- SPIRE is an implementation of the specification.
- SPIFFE is a set of open-source specifications for a framework capable of bootstrapping and issuing identity to services across heterogeneous environments and organizational boundaries. ( <https://spiffe.io/docs/latest/spiffe/overview/> )
- Provides an identity framework suitable for today's cloud native applications.
- Service meshes are moving toward supporting SPIFFE. The extent of support varies based on service mesh.
- As an example, Network Service Mesh has adopted SPIFFE and uses SPIRE to provide the implementation.
- Kuma, Consul Connect and Istio all support parts of the SPIFFE specification.

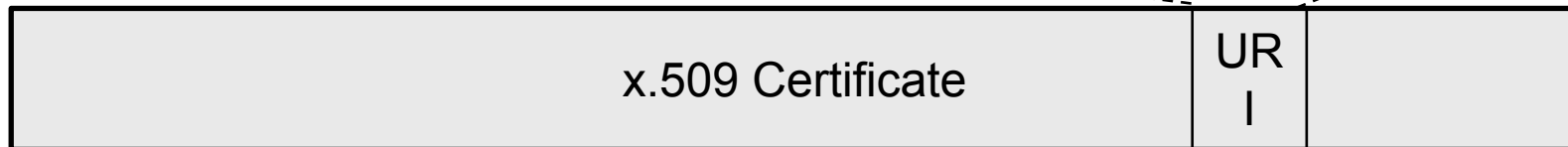
SOURCE: "Security in the World of Service Meshes"

[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)



# SVID as defined by SPIFFE

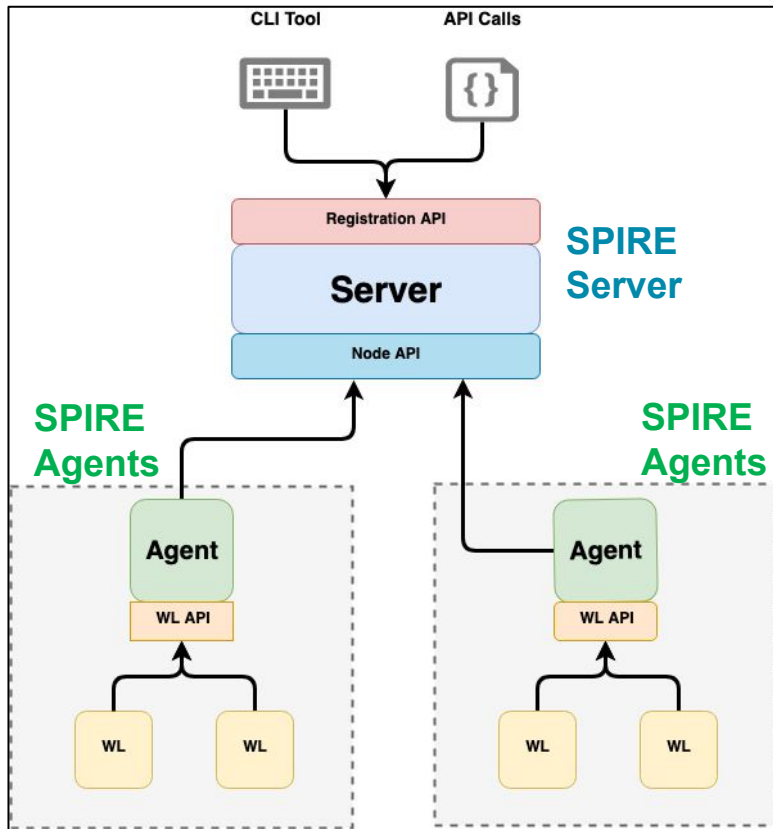
`spiffe://<domain>/ns/<namespace>/sa/<serviceaccount>`



SOURCE: "Security in the World of Service Meshes"

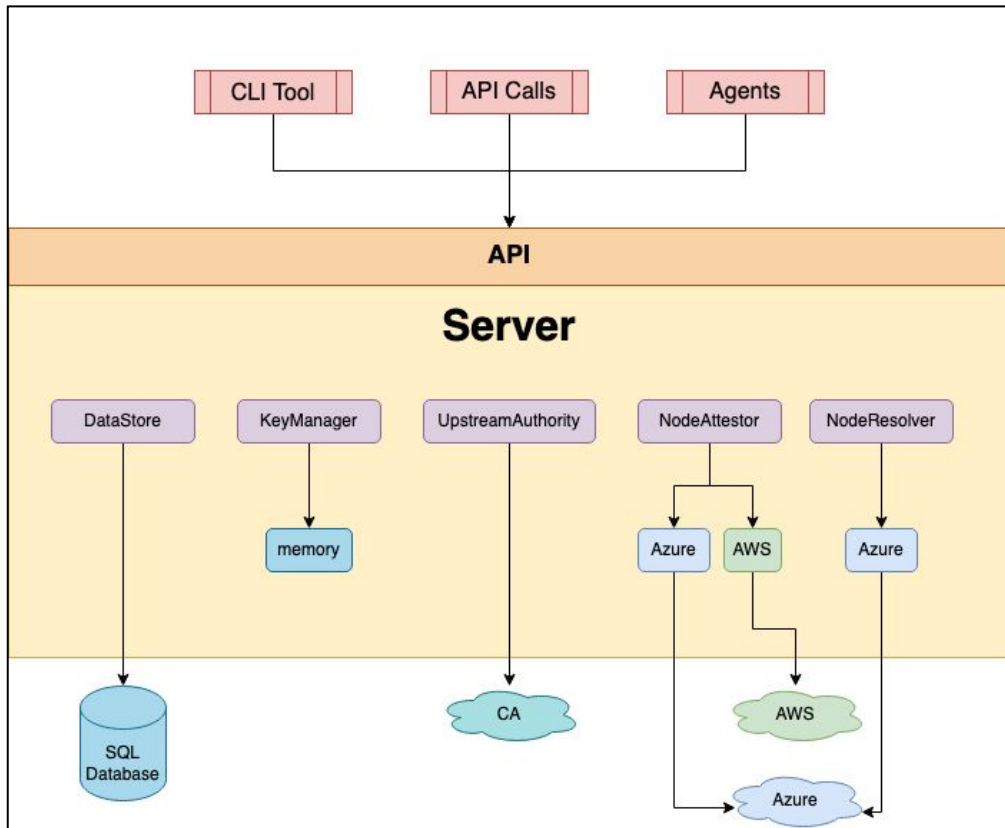
[https://www.cncf.io/wp-content/uploads/2020/11/Mesh\\_security\\_webinar\\_final.pptx](https://www.cncf.io/wp-content/uploads/2020/11/Mesh_security_webinar_final.pptx)

# Spire Server and Agent



1. A SPIRE deployment is composed of a **SPIRE Server** and one or more **SPIRE Agents**.
2. A **Server** acts as a signing authority for identities issued to a set of workloads via **Agents**.
3. The **Server** also maintains a registry of workload identities and the conditions that must be verified in order for those identities to be issued.
4. **Agents** expose the SPIFFE Workload API locally to workloads, and must be installed on each node on which a workload is running.

# SPIRE Server Design Details



**A SPIRE Server is responsible for managing and issuing all identities in its configured SPIFFE trust domain.** It stores registration entries (which specify the selectors that determine the conditions under which a particular SPIFFE ID should be issued) and signing keys, uses node attestation to authenticate agents' identities automatically, and creates SVIDs for workloads when requested by an authenticated agent.

The behavior of the server is determined through a series of plugins. SPIRE comes with several plugins included, but additional plugins can be built to extend SPIRE for specific use cases. Types of plugins include:

Node attester plugins which, together with agent node attestors, verify the identity of the node the agent is running on. See the section Node Attestation for more information.

Node resolver plugins which expand the set of selectors the server can use to identify the node by verifying additional properties about the node. See the section Node Resolution for more information.

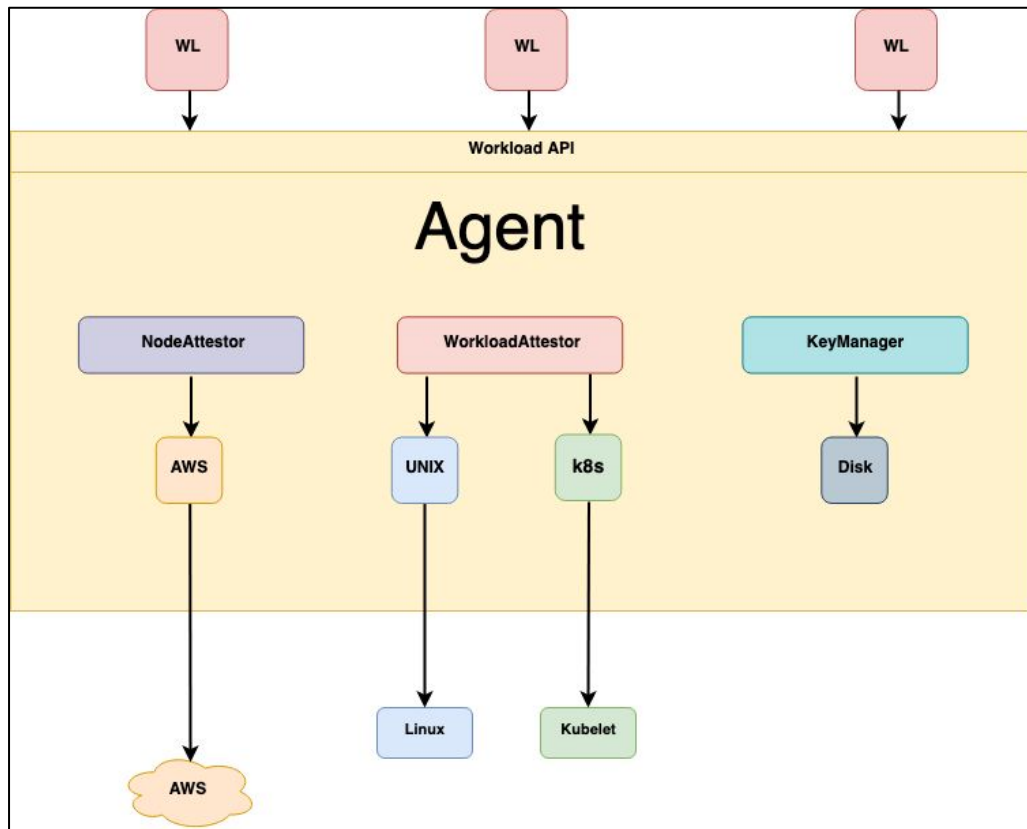
Datastore plugins, which the server uses to store, query, and update various pieces of information, such as registration entries, which nodes have attested, what the selectors for those nodes are. There is one built-in datastore plugin which can use a MySQL, SQLite 3, or PostgreSQL database to store the necessary data. By default it uses SQLite 3.

Key manager plugins, which control how the server stores private keys used to sign X.509-SVIDs and JWT-SVIDs.

Upstream authority plugins. By default the SPIRE Server acts as its own certificate authority. However, you can use an upstream authority plugin to use a different CA from a different PKI system.

You customize the server's behavior by configuring plugins and various other configuration variables. See the SPIRE Server Configuration Reference for details.

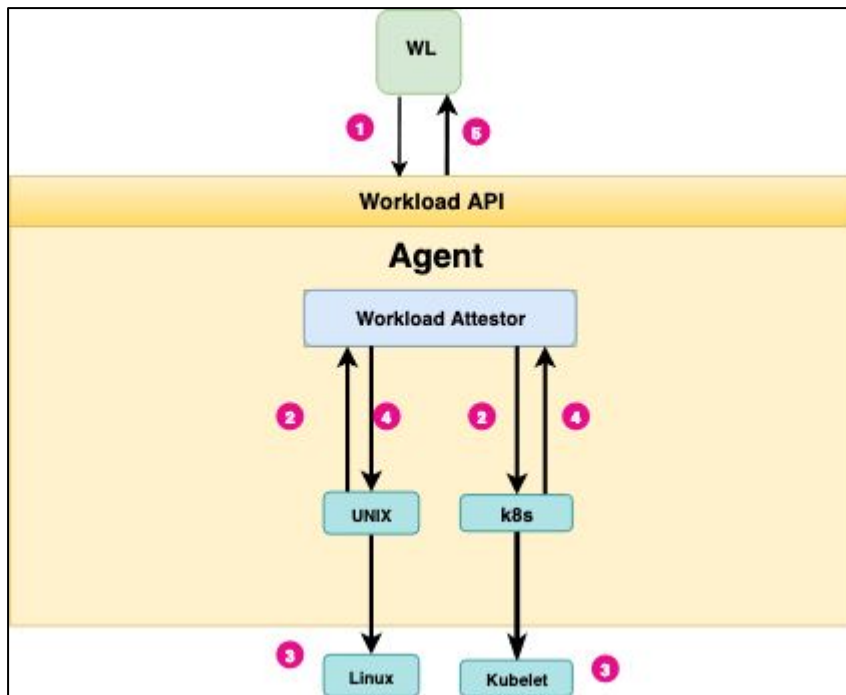
# SPIRE Node Agent Design Details



A SPIRE Agent runs on every node on which an identified workload runs. The agent:

1. Requests SVIDs from the server and caches them until a workload requests its SVID
2. Exposes the SPIFFE Workload API to workloads on node and attests the identity of workloads that call it
3. Provides the identified workloads with their SVIDs

# Workload Attestation (1/2)



Workload attestation asks the question: “Who is this process?” The agent answers that question by interrogating locally available authorities (such as the node’s OS kernel, or a local kubelet running on the same node) in order to determine the properties of the process calling the Workload API.

These properties are then compared against the information provided to the server when you registered the workload’s properties using selectors.

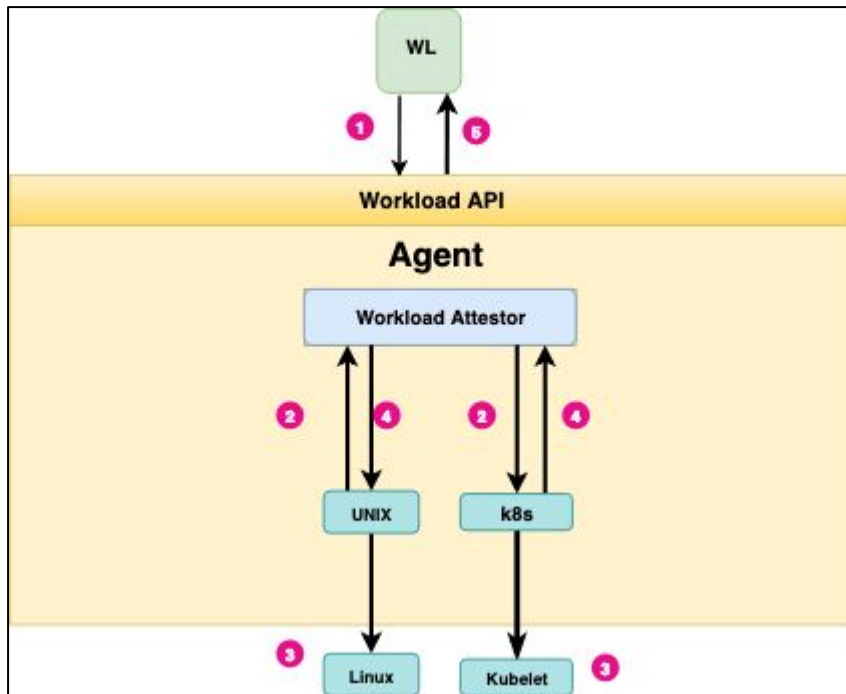
These types of information might include:

- How the process is scheduled by the underlying operating system. On a Unix-based systems, this might be by User ID (uid), Group ID (gid), filesystem path, etcetera.)
- How the process is scheduled by an orchestration system such as Kubernetes. In this case, the workload might be described by the Kubernetes Service Account or namespace it is running in.

While both agents and servers play a role in node attestation, only agents are involved in workload attestation.

The diagram illustrates the steps in workload attestation.

# Workload Attestation (2/2)



## Summary of Steps: Workload Attestation

1. A workload (WL) calls the Workload API to request an SVID. On Unix systems this is exposed as a Unix Domain Socket.
2. The agent interrogates the node's kernel to identify the process ID of the caller. It then invokes any configured workload attestor plugins, providing them with the process ID of the workload.
3. Workload attestors use the process ID to discover additional information about the workload, querying neighboring platform-specific components – such as a Kubernetes kubelet as necessary. Typically these components also reside on the same node as the agent.
4. The attestors return the discovered information to agent in the form of selectors.
5. The agent determines the workload's identity by comparing discovered selectors to registration entries, and returns the correct cached SVID to the workload.

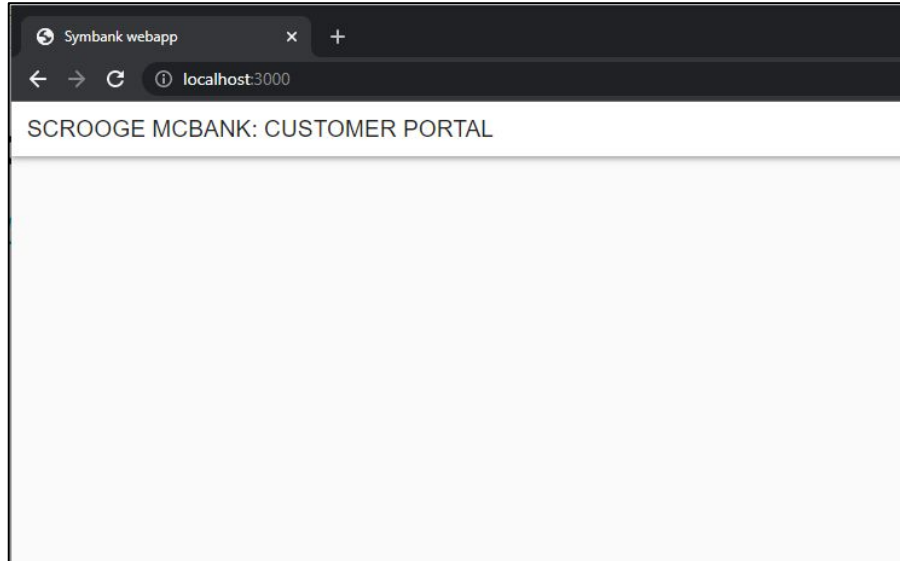
# Spiffe Walkthrough and Demo

1. Basics and Setup
2. Integrating with Envoy using X.509 certs
3. Integrating with Envoy using JWT
4. Using SPIFFE X.509 IDs with Envoy and Open Policy Agent Authorization
5. Using SPIFFE JWT IDs with Envoy and Open Policy Agent Authorization

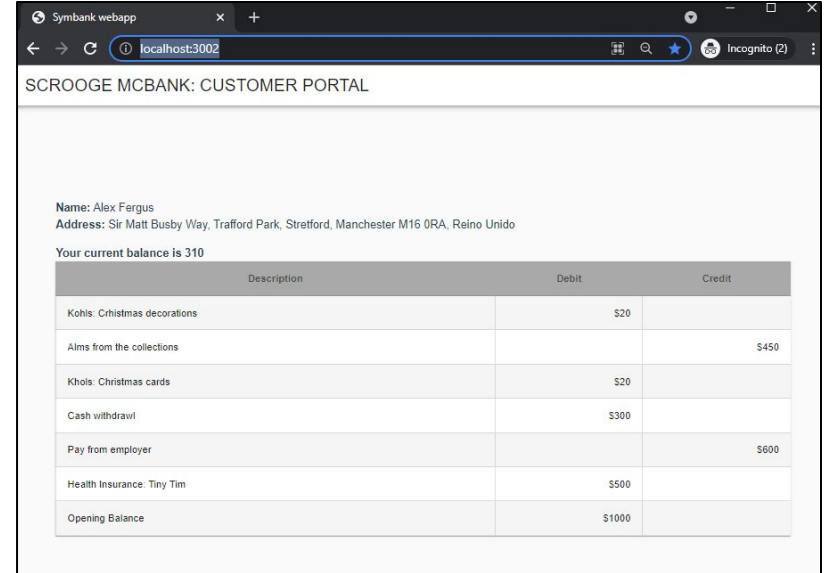
SOURCE: <https://github.com/spiffe/spire-tutorials>; <https://github.com/spiffe/spire-tutorials/blob/master/k8s/envoy-x509>;  
<https://github.com/spiffe/spire-tutorials/blob/master/k8s/envoy-jwt>; <https://github.com/spiffe/spire-tutorials/blob/master/k8s/envoy-opa>

# Two Frontends

<http://localhost:3000/>



<http://localhost:3002/>



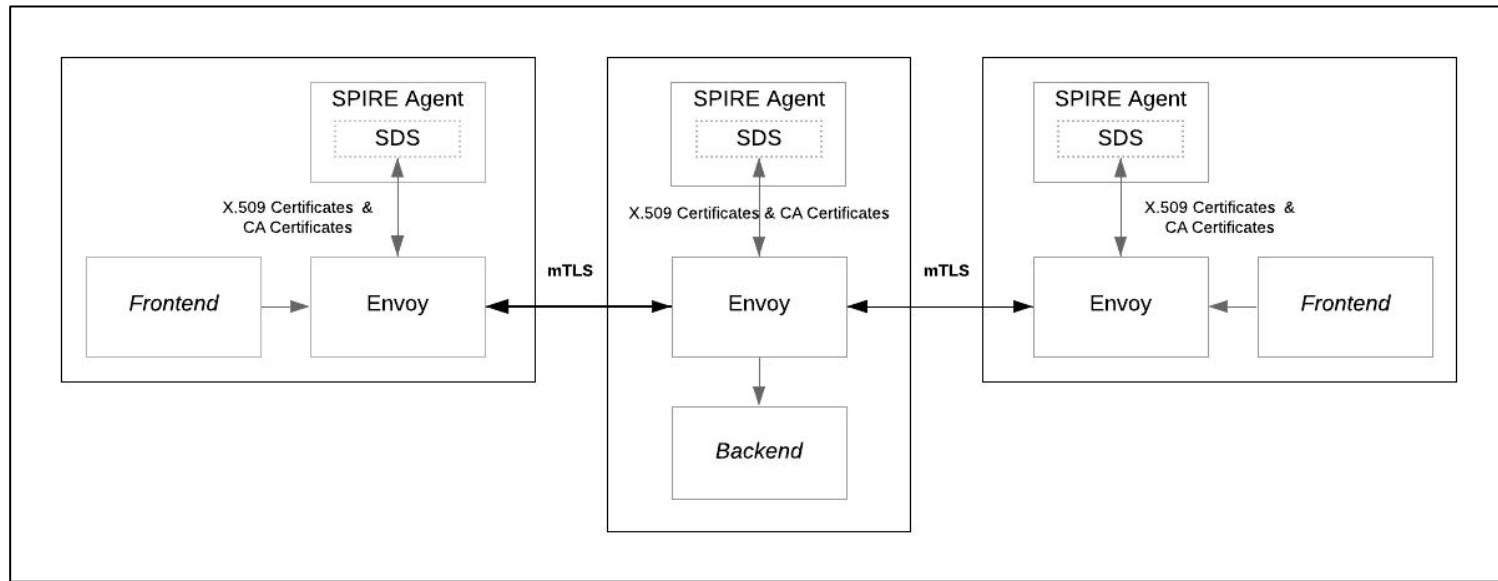


# Spiffe Walkthrough and Demo - Basics and Setup

1. Create the appropriate Kubernetes namespaces and service accounts to deploy SPIRE
2. Deploy the SPIRE Server as a Kubernetes statefulset
3. Deploy the SPIRE Agent as a Kubernetes daemonset
4. Configure a registration entry for a workload
5. Fetch an x509-SVID over the SPIFFE Workload API

```
1.  cd C:\Users\AWNATHAN\Desktop\Others\spire-tutorials-master\k8s\quickstart\
2.  kubectl apply -f spire-namespace.yaml
3.  kubectl get namespaces
4.  kubectl apply -f server-account.yaml -f spire-bundle-configmap.yaml -f
    server-cluster-role.yaml
5.  kubectl apply -f server-configmap.yaml -f server-statefulset.yaml -f server-service.yaml
6.  kubectl get statefulset --namespace spire
7.  kubectl apply -f agent-account.yaml -f agent-cluster-role.yaml
8.  kubectl apply -f agent-configmap.yaml -f agent-daemonset.yaml
9.  kubectl get daemonset --namespace spire
10. kubectl exec -n spire spire-server-0 -- /opt/spire/bin/spire-server entry create -spiffeID
    spiffe://example.org/ns/spire/sa/spire-agent -selector k8s_sat:cluster:demo-cluster -selector
    k8s_sat:agent_ns:spire -selector k8s_sat:agent_sa:spire-agent -node
11. kubectl exec -n spire spire-server-0 -- /opt/spire/bin/spire-server entry create -spiffeID
    spiffe://example.org/ns/default/sa/default -parentID
    spiffe://example.org/ns/spire/sa/spire-agent -selector k8s:ns:default -selector k8s:sa:default
12. kubectl apply -f client-deployment.yaml
13. kubectl exec -it $(kubectl get pods -o=jsonpath='{.items[0].metadata.name}' -l app=client) --
    /bin/sh
14. /opt/spire/bin/spire-agent api fetch -socketPath /run/spire/sockets/agent.sock
15. exit
```

# Spiffe Walkthrough and Demo - Integrating with Envoy using X.509 certs: Configure Envoy to Perform X.509 SVID Authentication (1/2)



<https://github.com/spiffe/spire-tutorials/tree/master/k8s/envoy-x509>

# Spiffe Walkthrough and Demo - Integrating with Envoy using X.509 certs: Configure Envoy to Perform X.509 SVID Authentication (2/2)

```
cd C:\Users\AWNATHAN\Desktop\Others\spire-tutorials-master\k8s\envoy-x509
```

```
kubectl apply -k k8s/.
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/backend -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app:backend -selector k8s:container-name:envoy
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/frontend -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app:frontend -selector k8s:container-name:envoy
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/frontend-2 -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app:frontend-2 -selector k8s:container-name:envoy
```

```
kubectl get services
```

```
http://localhost:3000/
```

```
http://localhost:3002/
```

```
kubectl apply -f backend-envoy-configmap-update.yaml
```

```
kubectl scale deployment backend --replicas=0
```

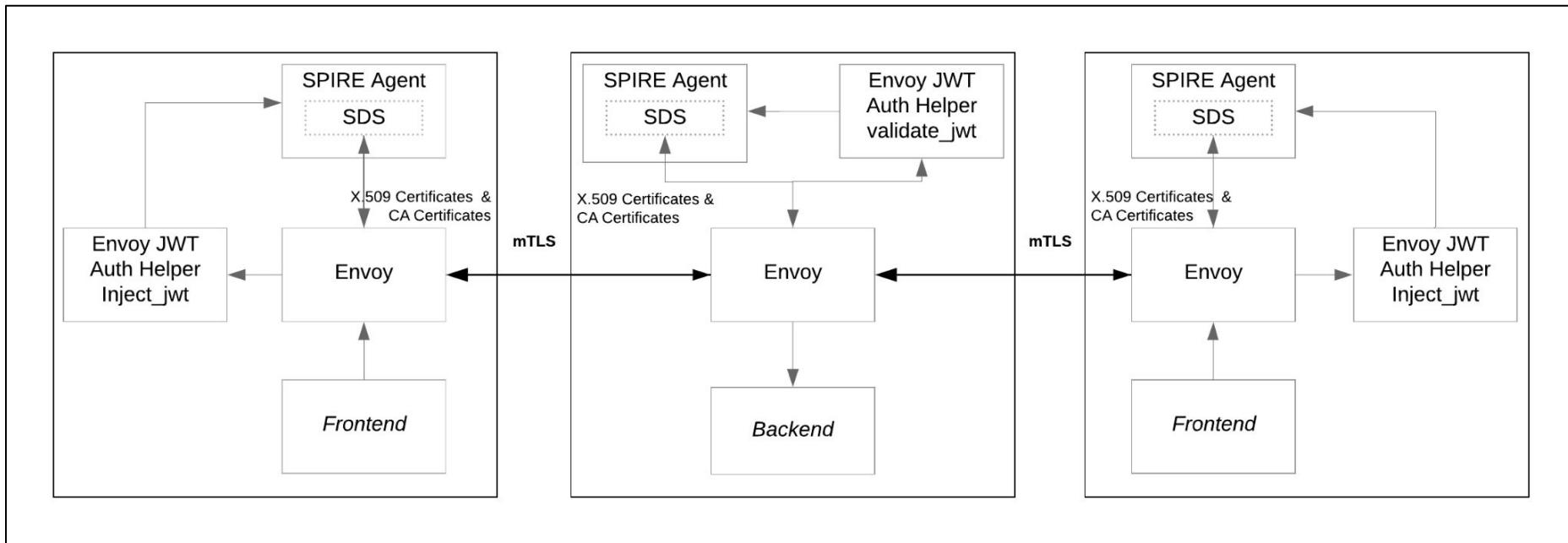
```
kubectl scale deployment backend --replicas=1
```

```
http://localhost:3000/
```

```
http://localhost:3002/
```

<https://github.com/spiffe/spire-tutorials/tree/master/k8s/envoy-x509>

# Spiffe Walkthrough and Demo - Integrating with Envoy using JWT (1/2)



# Spiffe Walkthrough and Demo - Integrating with Envoy using JWT (2/2)

```
cd C:\Users\AWNATHAN\Desktop\Others\spire-tutorials-master\k8s\envoy-jwt
```

```
kubectl delete deployment backend
```

```
kubectl delete deployment frontend
```

```
kubectl apply -k k8s/.
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/backend -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app=backend -selector k8s:container-name:auth-helper
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/frontend -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app=frontend -selector k8s:container-name:auth-helper
```

```
kubectl logs -f --selector=app=frontend -c auth-helper
```

```
kubectl logs -f --selector=app=backend -c auth-helper
```

```
http://localhost:3000/
```

```
http://localhost:3002/
```

```
kubectl delete deployment frontend-2
```

```
kubectl apply -k k8s/frontend-2/.
```

```
cd .\k8s\frontend-2\
```

```
kubectl exec -n spire spire-server-0 -c spire-server -- /opt/spire/bin/spire-server entry create -parentID spiffe://example.org/ns/spire/sa/spire-agent -spiffeID spiffe://example.org/ns/default/sa/default/frontend-2 -selector k8s:ns=default -selector k8s:sa=default -selector k8s:pod-label:app=frontend-2 -selector k8s:container-name:auth-helper
```

```
kubectl logs -f --selector=app=frontend-2 -c auth-helper
```

```
http://localhost:3002/
```

SOURCE:

<https://github.com/spiffe/spire-tutorials/tree/master/k8s/envoy-jwt>

# Open Policy Agent (1/2)



Open Policy Agent

## Declarative Policy

Context-aware, Expressive, Fast, Portable



Allow everyone to run GET on /pets

Allow frontend service to run GET on /pets/owners

```
package envoy.authz

# allow all GET requests to /pets
default allow = false
allow {
  input.attributes.request.http.method == "GET"
  input.attributes.request.http.path == "/pets"
}
```

Open In Playground

## Architectural Flexibility

Balance integration, availability, consistency

### Daemon



Deploy OPA as a separate process on the same host as your service. Integrate OPA by changing your service's code, importing an OPA-enabled library, or using a network proxy integrated with OPA.

### Library



Embed OPA policies into your service. Integrate OPA as a Go library that evaluates policy, or integrate a WebAssembly runtime and use OPA to compile policy to WebAssembly instructions.

Policy-based control for cloud native environments: Flexible, fine-grained control for administrators across the stack.

Stop using a different policy language, policy model, and policy API for every product and service you use. Use OPA for a unified toolset and framework for policy across the cloud native stack. Whether for one service or for all your services, use OPA to decouple policy from the service's code so you can release, analyze, and review policies (which security and compliance teams love) without sacrificing availability or performance.

Declarative Policy: Context-aware, Expressive, Fast, Portable

Architectural Flexibility: Balance integration, availability, consistency

1. Deploy OPA as a separate process on the same host as your service. Integrate OPA by changing your service's code, importing an OPA-enabled library, or using a network proxy integrated with OPA.
2. Embed OPA policies into your service. Integrate OPA as a Go library that evaluates policy, or integrate a WebAssembly runtime and use OPA to compile policy to WebAssembly instructions.

SOURCE:

<https://www.openpolicyagent.org/>

# Open Policy Agent (2/2) - Rego Playground



Open Policy Agent

The Rego Playground

Examples ▾

Search

All Access Control Envoy Kubernetes

Access Control

Role-based  
An example of classic Role-based Access Control (RBAC)

Attribute-based  
An example of classic Attribute-based Access Control (ABAC)

Envoy

Hello World  
Allow everyone access to public APIs

JWT Decoding  
Verify a JWT and use the claims for authorization

Roles  
Allow access to an API based on whether the user is an admin

URL Extraction  
Grant the user access to her own user profiles

Kubernetes

Hello World  
Ensure every resource has a 'costcenter' label in the appropriate format

Label Existence  
Ensure every resource has a costcenter label and that the costcenter value is in the correct format

Image Safety  
Ensure every image in a pod comes from a trusted registry

Ingress Conflicts  
Ensure no ingress gets created that conflicts with an existing ingress

INPUT

```
1 {  
2   "user": "alice",  
3   "action": "read",  
4   "object": "id123",  
5   "type": "dog"  
6 }  
7
```

DATA

```
1 {  
2   "user_roles": {  
3     "alice": [  
4       "admin"  
5     ],  
6     "bob": [  
7       "employee",  
8       "billing"  
9     ],  
10    "eve": [  
11      "customer"  
12    ]  
13  },  
14  "role_grants": {  
15
```

OUTPUT

```
1
```

Built by sttyro

OPA v0.33.1

SOURCE: <https://play.openpolicyagent.org/p/DqXNKeLm20>

# Spiffe Walkthrough and Demo - Using SPIFFE X.509 IDs with Envoy and Open Policy Agent Authorization

```
cd  
C:\Users\AWNATHAN\Desktop\Others\spire-tutorials-master\k8s\envoy-opa
```

```
kubectl apply -k k8s/.
```

```
kubectl get services
```

```
kubectl edit configmap backend-opa-policy-config
```

```
spiffe://example.org/ns/default/sa/default/frontend-2
```

```
kubectl scale deployment backend --replicas=0
```

```
kubectl scale deployment backend --replicas=1
```

SOURCE:

<https://github.com/spiffe/spire-tutorials/tree/master/k8s/envoy-opa>



# Spiffe Walkthrough and Demo - Using SPIFFE JWT IDs with Envoy and Open Policy Agent Authorization

```
cd C:\Users\AWNATHAN\Desktop\Others\spire-tutorials-master\k8s\envoy-jwt-opa
```

```
kubectl apply -k k8s/.
```

```
kubectl scale deployment backend --replicas=0
```

```
kubectl scale deployment backend --replicas=1
```

```
kubectl edit configmap backend-opa-policy-config
```

```
svc_spiffe_id == "spiffe://example.org/ns/default/sa/default/frontend-2"
```

```
kubectl scale deployment backend --replicas=0
```

```
kubectl scale deployment backend --replicas=1
```

```
http://localhost:3000/
```

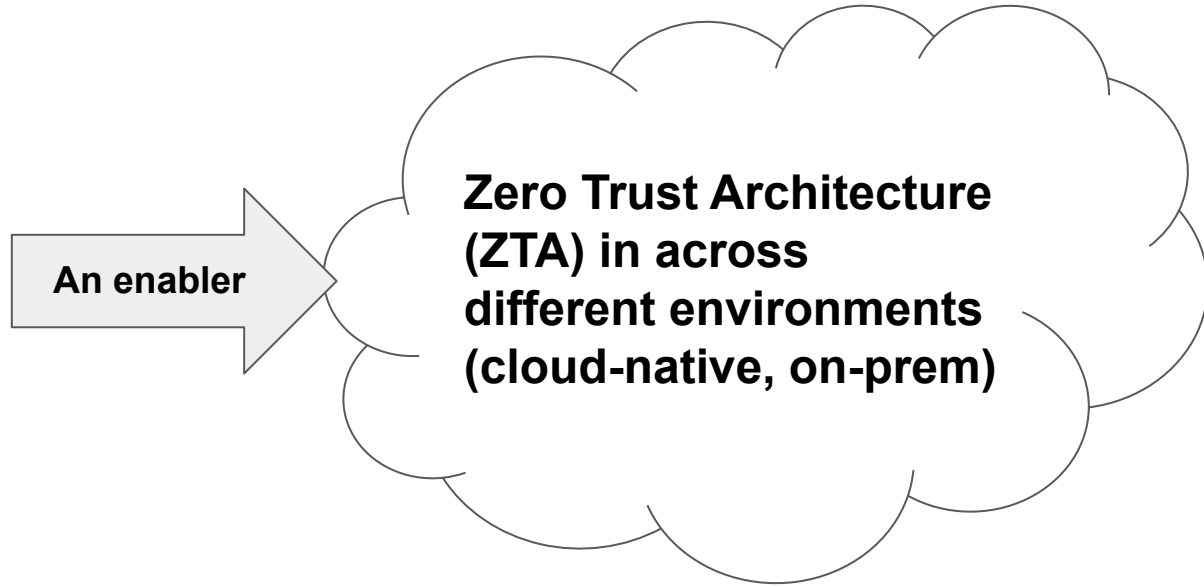
```
http://localhost:3002/
```

SOURCE:

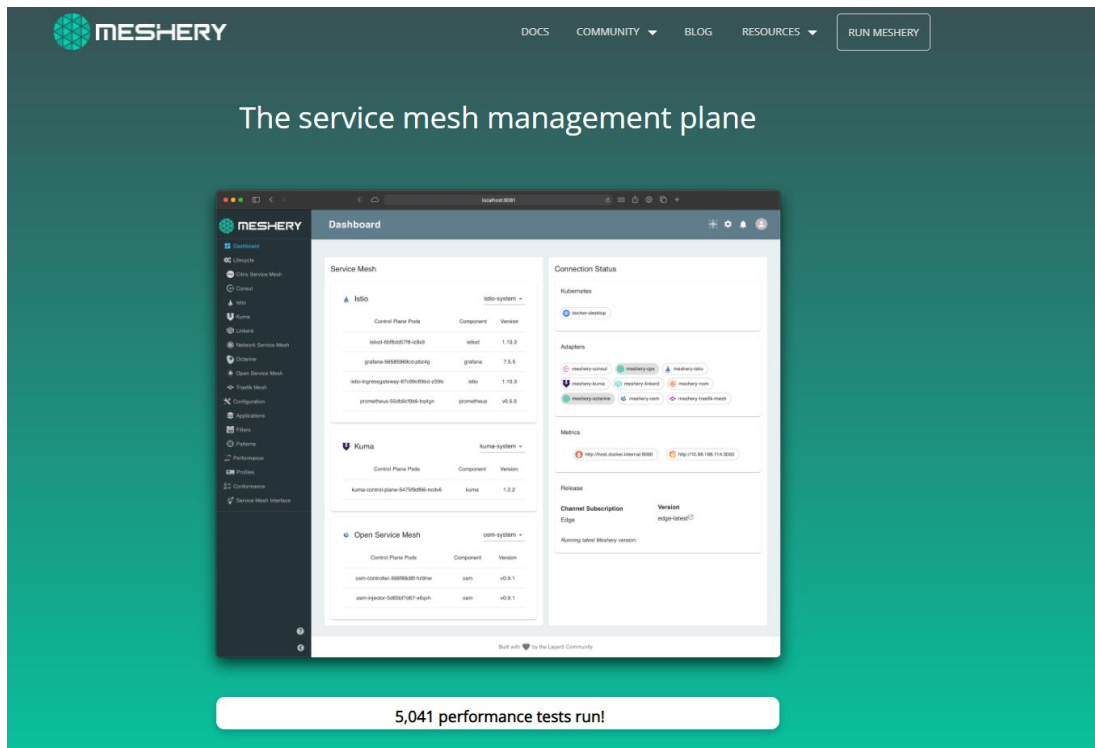
<https://github.com/spiffe/spire-tutorials/tree/master/k8s/envoy-jwt-opa>

## Recap

**Identity-Based  
Micro-segmentation  
with Securing  
Production Identity  
Framework for  
Everyone (SPIFFE),  
Service Mesh and  
Open Policy Agent**



# Another Useful Project for Service Mesh - Meshery: The Service Mesh Management Plane



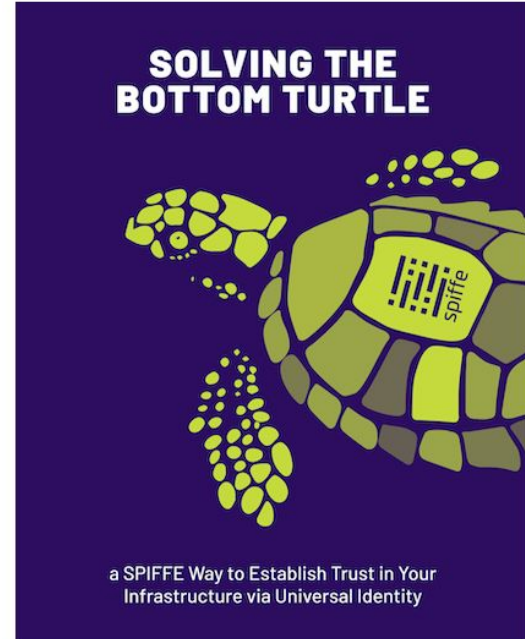
**Meshery is the open source, service mesh management plane that enables the adoption, operation, and management of any service mesh and their workloads.**

# Solving the Bottom Turtle - a very fascinating read! (Please check it out!)

This book presents the SPIFFE standard for service identity, and SPIRE, the reference implementation for SPIFFE. These projects provide a uniform identity control plane across modern, heterogeneous infrastructure. Both projects are open source and are part of the Cloud Native Computing Foundation. As organizations grow their application architectures to make the most of new infrastructure technologies, their security models must also evolve.

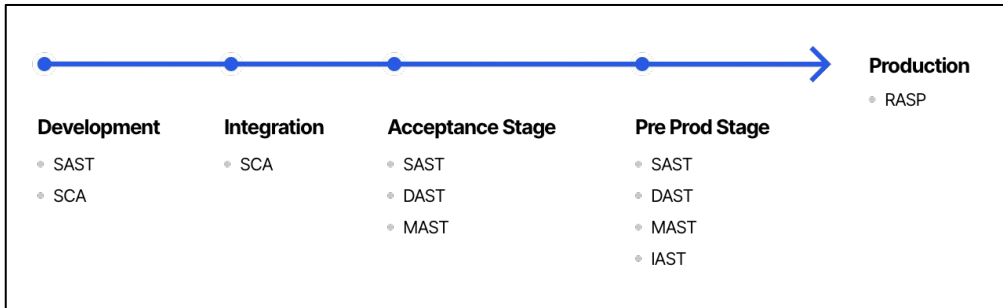
Software has grown from one monolith on one box, to dozens or hundreds of tightly linked microservices that may be spread across thousands of virtual machines in public clouds or private data centers.

In this new infrastructure world, SPIFFE and SPIRE help keep systems secure. This book strives to distill the experience from the foremost security experts and SPIFFE community members to provide a deep understanding of the identity problem and how to solve it. With these projects, developers and operators can build software using new infrastructure technologies while allowing security teams to step back from expensive and time-consuming manual security processes.

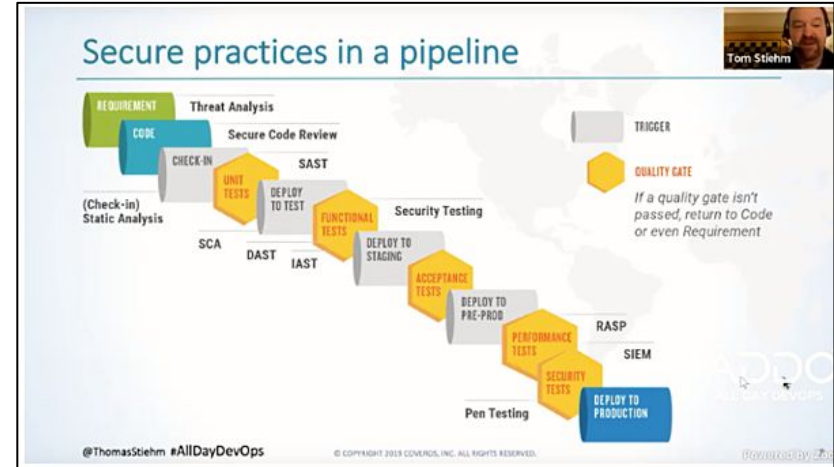


**SOURCE:** <https://spiffe.io/book/>

# Building an End-to-End Secure Software Factory (SSF) to Defend the Digital Cloud-Native Software Supply Chain against attacks: Helpful Cloud-Native Security Checklists and Notary (The Update Framework) (Demo)

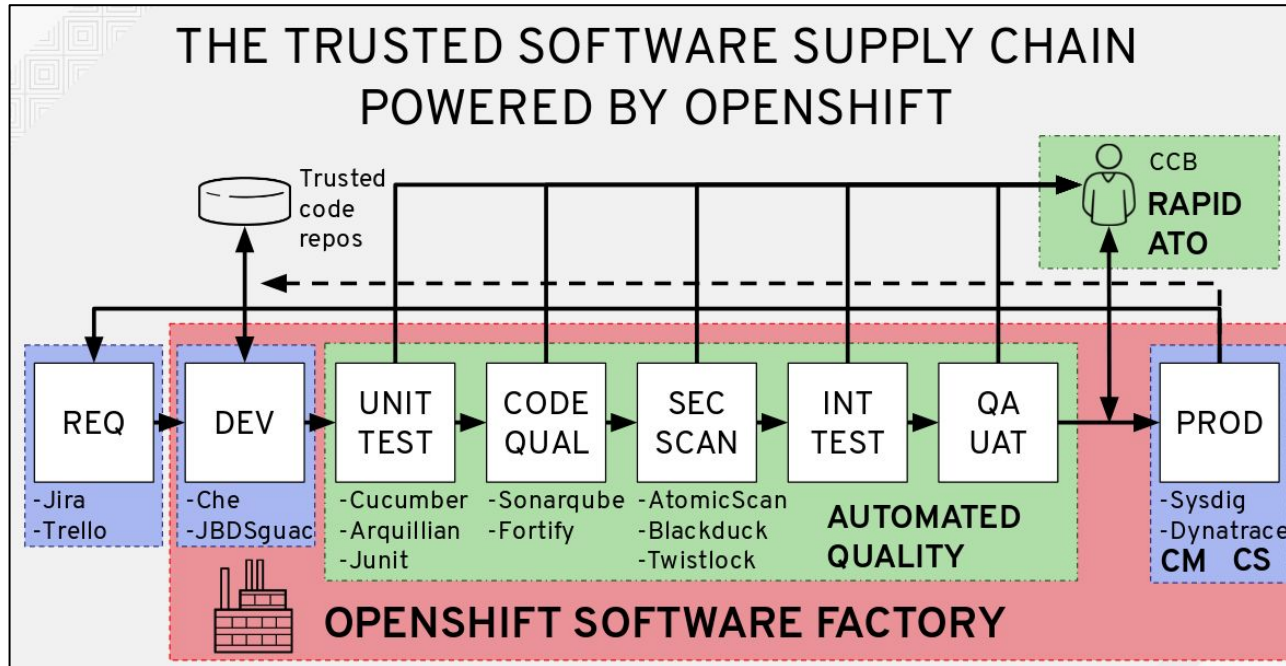


**Acronyms overload!**  
**Runtime Application Self Protection (RASP),**  
**etc...**



Bring your developers along this journey! Involve them at every step so they won't feel forced/compelled/threatened. Secure buy-in through tender loving care (TLC).

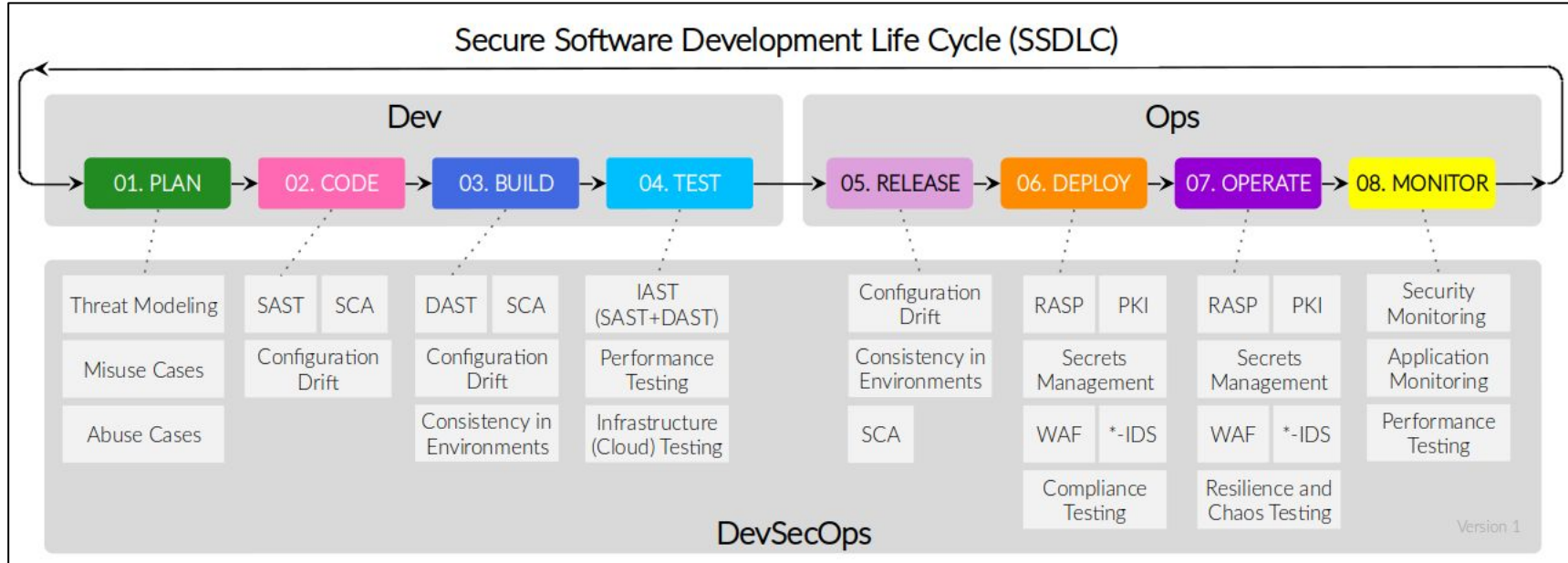
# Trusted Software Supply Chain



Each phase in our Trusted Software Supply Chain will have a policy defining the stage gate for success and will generate a documentation artifact that will later be used as part of the ATO process - **Authority to Operate (ATO)**

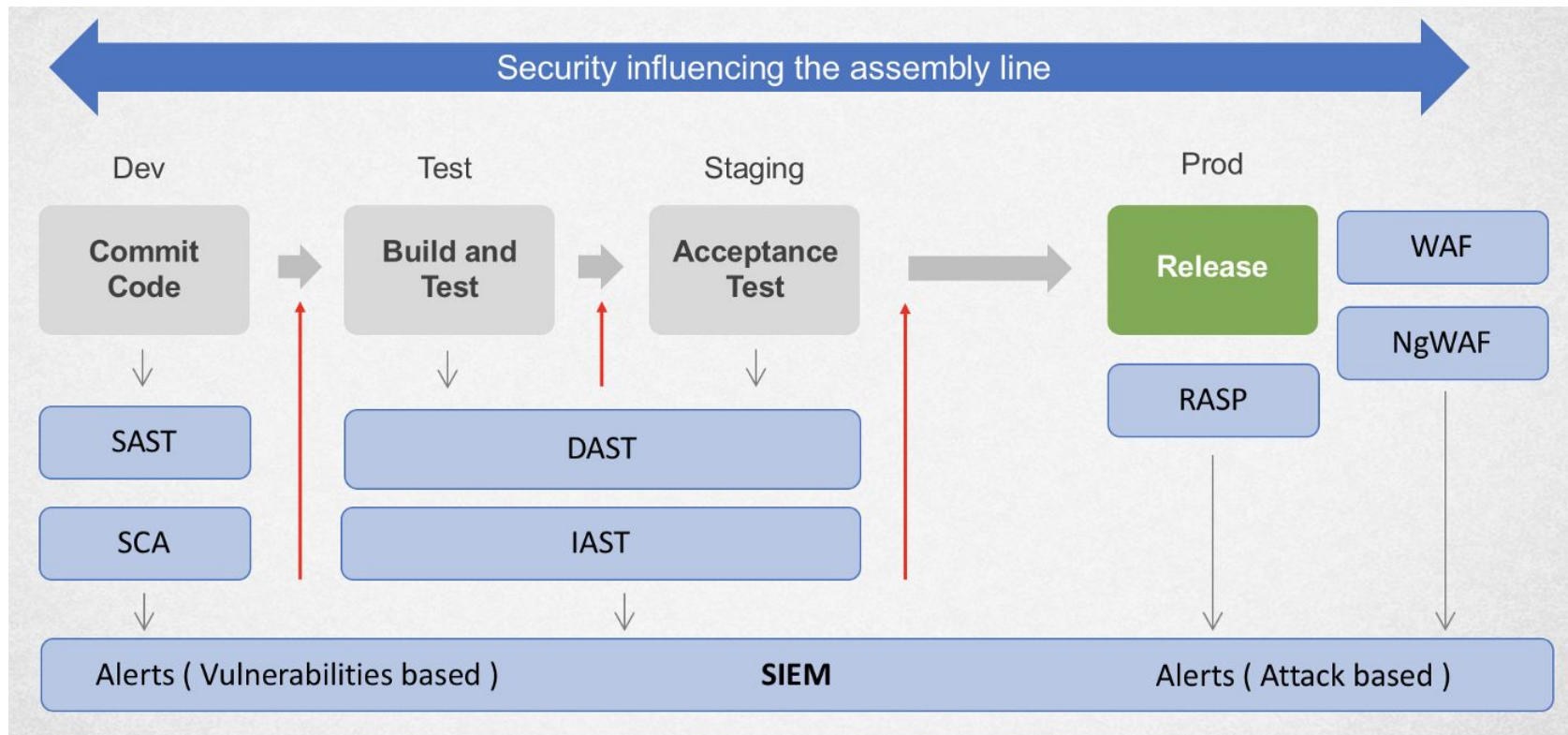
SOURCE: [http://redhatgov.io/workshops/secure\\_software\\_factory/lab02/](http://redhatgov.io/workshops/secure_software_factory/lab02/)  
<https://cloud.gov/docs/compliance/ato-process/>

# Trusted Software Supply Chain / Secure Software Development Life Cycle (SSDLC)



SOURCE:

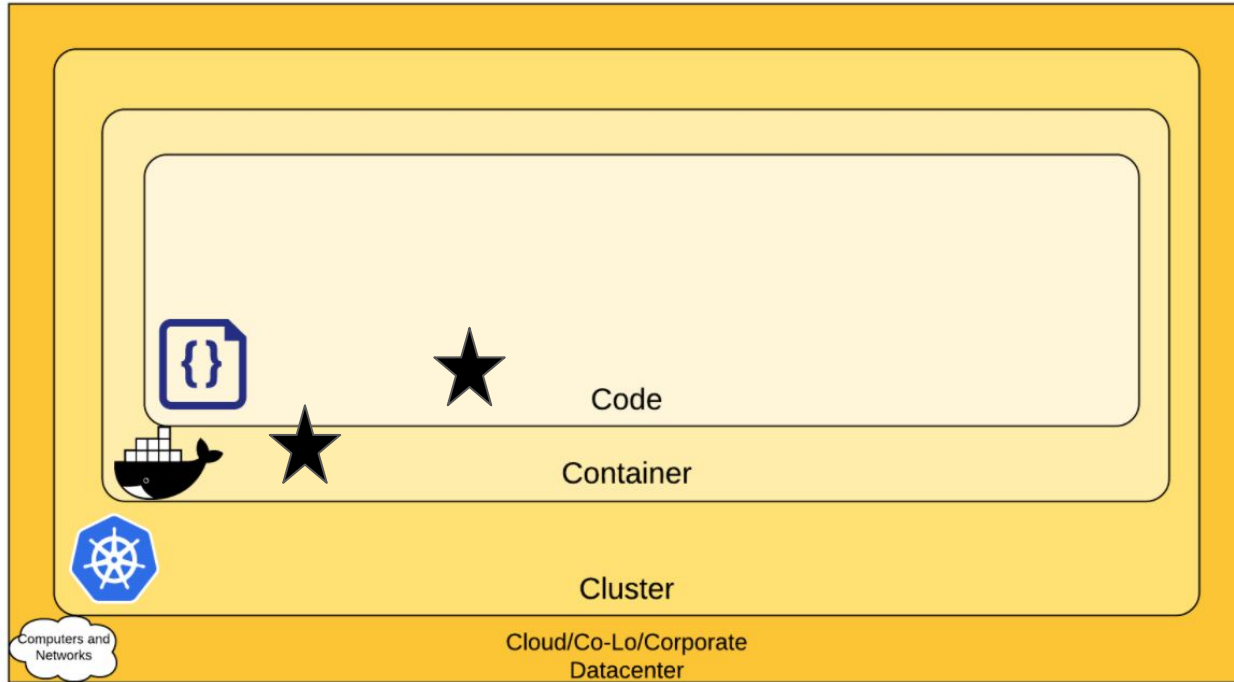
<https://holisticsecurity.io/assets/blog20200210/20200210-security-along-container-based-sdlc-v2.png>



SOURCE: <https://blog.shiftright.io/understand-offense-to-inform-defense-ccb06f69ac19>



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



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.

SOURCE: <https://kubernetes.io/docs/concepts/security/overview/>

# Helpful Cloud-native Security Checklists/Tools/Guidelines



## Security Challenges for Cloud Native Architectures

Addressing Visibility and Protection for Cloud Workloads

### 1. Develop

IaC scanning, early detection of compliance and security violations with DevOps integrations

- ❑ Scan IaC templates in the IDE
- ❑ Scan Dockerfiles for vulnerabilities
- ❑ Scan Kubernetes® app manifests for insecure configurations
- ❑ Scan source code repositories for package vulnerabilities
- ❑ Trigger scans when developers make pull requests
- ❑ Develop tests based on threat modeling to identify hot spots

### 2. Distribute

Container image scanning, secure registries, and security testing

- ❑ Scan container images for vulnerabilities and malware
- ❑ Detect and alert on secrets leakage in container images
- ❑ Sign images and build metadata in the CI/CD pipeline
- ❑ Scan Kubernetes app manifests to identify compromising configurations
- ❑ Perform comprehensive static and dynamic app security testing (SAST, DAST)
- ❑ Maintain dedicated test environments to validate security tests
- ❑ Maintain private registries for development artifacts
- ❑ Maintain pre-production registries for production deployment artifacts
- ❑ Ensure the use of signed images throughout the process
- ❑ Ensure images are tagged with validated SHA digests
- ❑ Encrypt container images for confidentiality

### 3. Deploy

Pre-flight checks and security policies

- ❑ Validate image SHA digest and signatures
- ❑ Enforce image runtime policies
- ❑ Enforce container runtime policies
- ❑ Ensure standards-based policy evaluation (e.g., NIST 800-190)
- ❑ Ensure host compliance
- ❑ Adopt observability frameworks for metrics and logs
- ❑ Develop policies to evaluate container runtime behavior

### 4. Run

Automatic scaling and evolution to meet the needs of a highly dynamic environment

- ❑ Establish container host protection
- ❑ Use a trusted and secure boot sequence
- ❑ Use stripped-down OS
- ❑ Employ Kubernetes API security
- ❑ Enforce pod security policies
- ❑ Perform audit log analysis
- ❑ Encrypt secrets
- ❑ Enforce container runtime security (process, file, and network)
- ❑ Use Zero Trust and microsegmentation
- ❑ Check for trusted images
- ❑ Employ a service mesh

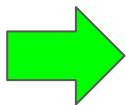
For more resources on cloud native security best practices, visit [Cyberpedia](#) and the [Prisma Cloud library](#).

**Prisma® Cloud** secures infrastructure, applications, data and entitlements across the world's largest clouds, all from a single unified solution. With a combination of cloud service provider APIs and a unified agent framework, users gain unmatched visibility and protection.

**Prisma Cloud** integrates with any continuous integration and continuous delivery (CI/CD) workflow to secure cloud infrastructure and applications early in development. Scan infrastructure-as-code (IaC) templates, container images, serverless functions and more while gaining powerful, full-stack runtime protection. This is unified security for DevOps and security teams.

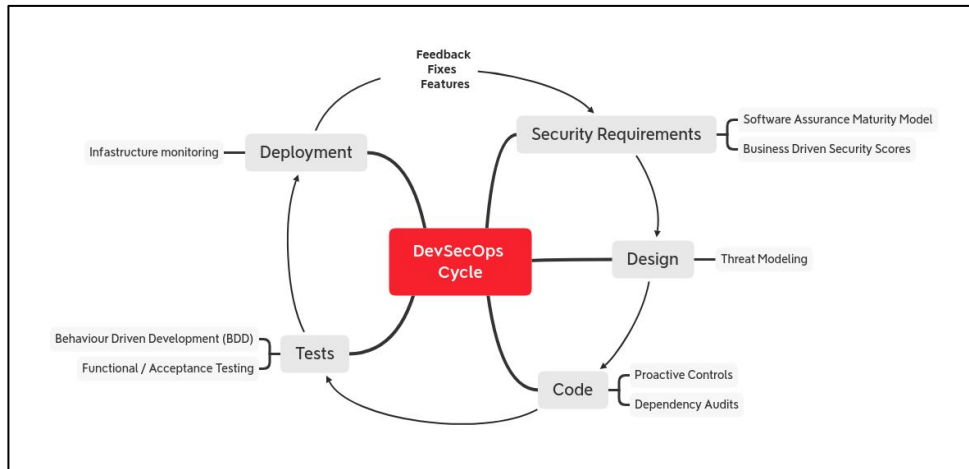
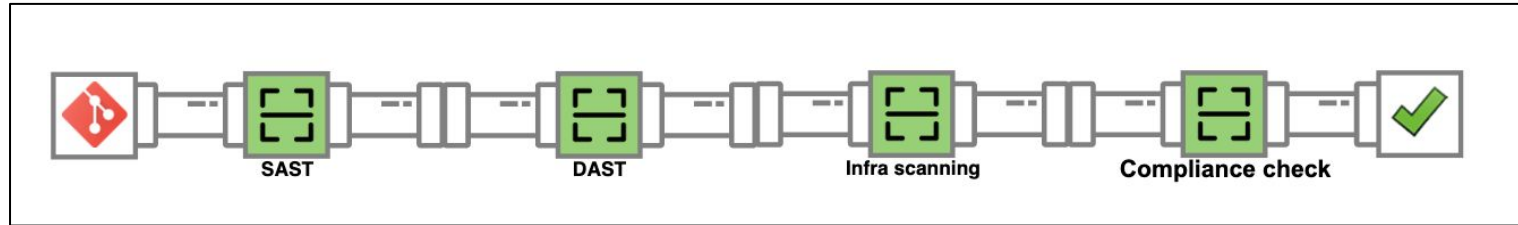
SOURCE:

[https://www.paloaltonetworks.com/apps/pan/public/downloadResource?pagePath=/content/pan/en\\_US/resources/datasheets/prisma-cloud-native-security-checklist-ds](https://www.paloaltonetworks.com/apps/pan/public/downloadResource?pagePath=/content/pan/en_US/resources/datasheets/prisma-cloud-native-security-checklist-ds)



**This Cloud-Native Security checklist is my personal favourite.**

# OWASP DevSecOps Guideline + OWASP Software Assurance Maturity Model (SAMM)



1. Take care of secrets and credentials in git repositories
2. SAST (Static Application Security Test)
3. IAST (Interactive Application Security Testing)
4. DAST (Dynamic Application Security Test)
5. Infrastructure scanning
6. Compliance check

<https://github.com/OWASP/DevSecOpsGuideline;>  
[https://owasp samm.org/model/;](https://owasp samm.org/model/)  
<https://owasp.org/www-project-devsecops-guideline/>

# OWASP Software Assurance Maturity Model (SAMM)



[ABOUT SAMM](#) [THE MODEL](#) [GUIDANCE](#) [COMMUNITY](#) [USER DAY](#) [CONTACT](#)

## THE MODEL

### SAMM model overview

Governance	Design	Implementation	Verification	Operations
Strategy and Metrics	Threat Assessment	Secure Build	Architecture Assessment	Incident Management
Policy and Compliance	Security Requirements	Secure Deployment	Requirements-driven Testing	Environment Management
Education and Guidance	Security Architecture	Defect Management	Security Testing	Operational Management

OWASP Software Assurance Maturity Model (SAMM) is to be the prime maturity model for software assurance that provides an effective and measurable way for all types of organizations to analyze and improve their software security posture.

OWASP SAMM supports the complete software lifecycle, including development and acquisition, and is technology and process agnostic. It is intentionally built to be evolvable and risk-driven in nature.

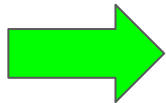
<https://owaspsamm.org/model/>

# Helpful Cloud-native Security Checklists/Tools/Guidelines

1. **Securing the Code:** For developers such as myself, and all other developers, start with something more developer-centric/friendly/palatable and/or starters such as the <https://12factor.net/> to slowly introduce security topics (Perhaps start from the second factor of the Twelve-Factor app: “Explicitly declare and isolate dependencies” to introduce Software Composition Analysis (SCA) tool such as OWASP Dependency Check)
2. **For Developers/Security Architects:**
  - a. Principles of secure development & deployment - <https://github.com/ukncsc/secure-development-and-deployment>
  - b. <https://github.com/brant-ruan/awesome-cloud-native-security>
3. **Hardening/securing the K8S Cluster**
  - a. [https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR\\_KUBERNETESHARDENINGGUIDANCE.PDF](https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR_KUBERNETESHARDENINGGUIDANCE.PDF)
  - b. <https://github.com/Vinum-Security/kubernetes-security-checklist>
4. **Securing the Cloud Open Source Tools for Cloud Security**
  - a. <https://github.com/toniblyx/my-arsenal-of-aws-security-tools>

# Helpful Cloud-native Security Checklists/Tools/Guidelines

- <https://github.com/gunjan5/cloud-native-security>
- <https://resources.github.com/downloads/DevSecOps-on-AWS-ATO-Brief-for-ReInvent-share.pdf>
- <https://github.com/mitre/mitre-saf/tree/master/public>
- <https://dodcio.defense.gov/Portals/0/Documents/Library/DoDEnterpriseDevSecOpsStrategyGuide.pdf>



These lists are not meant to be comprehensive but this is the distilled list.

# The Update Framework: A framework for securing software update systems



The Update Framework (TUF) helps developers maintain the security of software update systems, providing protection even against attackers that compromise the repository or signing keys. TUF provides a flexible framework and specification that developers can adopt into any software update system. TUF is hosted by the Linux Foundation as part of the Cloud Native Computing Foundation (CNCF) and is used in production by various tech companies and open source organizations.

A variant of TUF called Uptane is widely used to secure over-the-air updates in automobiles.

Uptane is an open and secure software update system design which protects software delivered over-the-air to the computerized units of automobiles. The framework can thwart attacks from malicious actors who can compromise servers and networks used to sign and deliver updates. Hence, it is designed to be resilient even to the best efforts of nation state attackers. There are multiple different free open source and closed source implementations available. Uptane is integrated into Automotive Grade Linux, an open source system currently used by many large OEMs, and has also been adopted by a number of U.S. and international manufacturers. Within the next few years, about one-third of new cars on U.S. roads will include Uptane.

Currently considered the de facto secure standard for software updates on automobiles.

<https://theupdateframework.com/>  
<https://uptane.github.io/>

# Uptane: Securing Software Updates for Automobiles

## Uptane

Securing Software Updates for Automobiles

[Home](#)[The Basics](#)[Design Documents](#)[Deployment Best Practices](#)[Learn More](#)[Press](#)

Uptane is an open and secure software update system design which protects software delivered over-the-air to the computerized units of automobiles. The framework can thwart attacks from malicious actors who can compromise servers and networks used to sign and deliver updates. Hence, it is designed to be resilient even to the best efforts of nation state attackers. There are multiple different free open source and closed source implementations available. Uptane is integrated into [Automotive Grade Linux](#), an open source system currently used by many large OEMs, and has also been adopted by a number of U.S. and international manufacturers. Within the next few years, about one-third of new cars on U.S. roads will include Uptane.

Currently considered the *de facto* secure standard for software updates on automobiles, in July 2018, formal standardization of Uptane began under a non-profit consortium called the [Uptane Alliance](#). [Uptane Standard for Design and Implementation](#) version 1.0, which presents procedures for secure design and implementation of the framework, was released on July 31, 2019, under the auspices of the [IEEE/ISTO Federation](#). The initiative now continues as a Linux Foundation Joint Development Foundation project, with versions [1.1.0](#) and [1.2.0](#) released in 2021. Recommended deployment strategies are under active development, and are published and regularly updated [on this site](#). All Uptane materials, including [technical papers](#), [security audits](#), and [a public reference implementation](#) are also freely available for all to use without a fee.

Uptane is a Joint Development Foundation project of the Linux Foundation, operating under the formal title of Joint Development Foundation Projects, LLC, Uptane Series. For questions about the documentation or implementation, please contact [Prof. Justin Cappos](#) and other members of the [Secure Systems Lab](#) at [New York University](#). Contributors and maintainers are governed by the [CNCF Community Code of Conduct](#).

Uptane is supported by U.S. Department of Homeland Security grants D15PC00239 and D15PC00302. The views and conclusions contained herein are the authors' and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Department of Homeland Security (DHS) or the U.S. government.

<https://uptane.github.io/>



# Python reference implementation of The Update Framework (TUF)

1. Install TUF
  - a. `python -m pip install securesystemslib[colors,crypto,pynacl] tuf`
2. Create a basic repository and client
  - a. `repo.py --init`
3. Add an update to the repository.
  - a. `echo 'Test file' > testfile`
  - b. `repo.py --add testfile`
4. Serve the repo
  - a. `python3 -m http.server 8001`
5. Obtain and verify the testfile update on a client.
  - a. `client.py --repo http://localhost:8001 testfile`

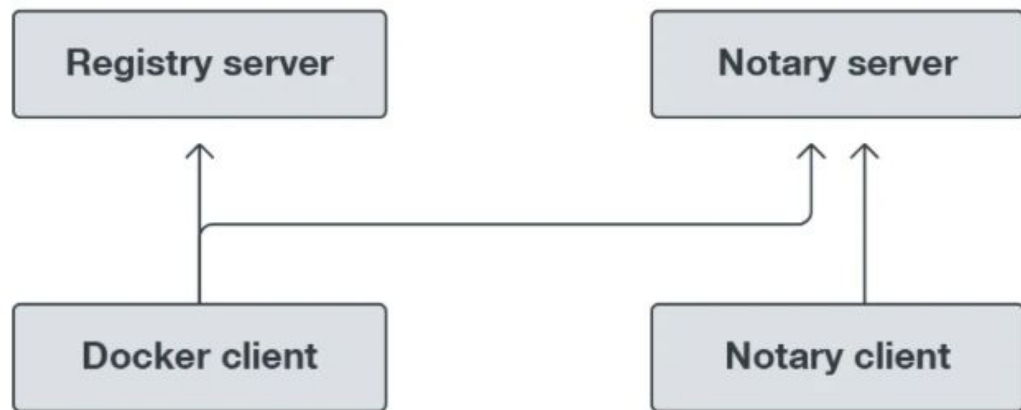
**The client can request the package testfile from the repository. TUF will download and verify metadata from the repository as necessary to determine what the trustworthy hashes and length of testfile are, then download the target testfile from the repository and keep it only if it matches that trustworthy metadata.**

<https://github.com/theupdateframework/python-tuf/blob/develop/docs/QUICKSTART.md>

<https://github.com/theupdateframework/python-tuf/blob/develop/docs/CLI.md>

# Docker Content Trust: What It Is and How It Secures Container Images

Can your container image be trusted? Learn how Docker Content Trust (DCT) employs digital signatures for container image verification and manages trusted collections of content.



The Docker Notary tool allows publishers to digitally sign their collections while users get to verify the integrity of the content they pull. Through The Update Framework (TUF), Notary users can provide trust over arbitrary collections of data and manage the operations necessary to ensure freshness of content.

<https://www.trendmicro.com/vinfo/sg/security/news/virtualization-and-cloud/docker-content-trust-what-it-is-and-how-it-secures-container-images>

**At the end of the day, the developer is still the key - no amount of framework, technologies, platforms can replace vigilance and awareness.**

“Zero trust is a concept, not an action.” - KEN WESTIN, SECURITY RESEARCHER

*“Zero trust is not a technology, it’s not something you buy, it’s a **strategy**.”*  
- Gregory Touhill

# References

- <https://www.cncf.io/online-programs/cncf-member-webinar-security-in-the-world-of-service-meshes/>
- [https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing\\_Multi\\_cloud\\_Portable\\_Tier\\_Microservices\\_Applications\\_A\\_live\\_demo\\_on\\_cloud\\_native\\_application\\_security\\_platforms.pdf](https://github.com/OWASP/www-chapter-singapore/raw/master/assets/presos/Securing_Multi_cloud_Portable_Tier_Microservices_Applications_A_live_demo_on_cloud_native_application_security_platforms.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/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/Securing\\_your\\_APIs\\_-\\_OWASP\\_API\\_Top\\_10\\_2019\\_Real-life\\_Case.pdf](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/Microservices%20Security%2C%20Container%20Runtime%20Security%2C%20MITRE%20ATT%26CK%2%AE%20%20for%20Kubernetes%20\(K8S\)%20and%20Service%20Mesh%20for%20Security.pdf](https://owasp.org/www-chapter-singapore/assets/presos/Microservices%20Security%2C%20Container%20Runtime%20Security%2C%20MITRE%20ATT%26CK%2%AE%20%20for%20Kubernetes%20(K8S)%20and%20Service%20Mesh%20for%20Security.pdf)
- <https://www.wired.com/story/what-is-zero-trust/>

# Reach Out

<https://www.linkedin.com/in/awnathan>  
[nathan.mk.aw@gmail.com](mailto:nathan.mk.aw@gmail.com)  
<https://nathanawmk.github.io/>

# Backup

# Pixie - Instant Kubernetes observability with Pixie

## Debug faster with code-level insights

PIXIE labs



Pixie is an open source observability tool for Kubernetes applications. Use Pixie to view the high-level state of your cluster (service maps, cluster resources, application traffic) and also drill-down into more detailed views (pod state, flame graphs, individual full-body application requests).

### Why Pixie?

Three features enable Pixie's magical developer experience:

**Auto-telemetry:** Pixie uses eBPF (Extended Berkeley Packet Filter, a kernel technology) to automatically collect telemetry data such as full-body requests, resource and network metrics, application profiles, and more. See the full list of data sources [here](#).

**In-Cluster Edge Compute:** Pixie collects, stores and queries all telemetry data locally in the cluster. Pixie uses less than 5% of cluster CPU, and in most cases less than 2%.

**Scriptability:** PxL, Pixie's flexible Pythonic query language, can be used across Pixie's UI, CLI, and client APIs.

SOURCE:

<https://newrelic.com/platform/kubernetes-pixie>

<https://github.com/pixie-io/pixie>

# GNAP

<https://github.com/ietf-wg-gnap/gnap-core-protocol>

<https://github.com/nathanawmk/oauth.xyz-java>



# Micro-segmentation Considerations

[https://www.illumio.com/sites/default/files/Illumio\\_Infographic\\_How\\_to\\_Choose\\_Your\\_Segmentation\\_Strategy\\_2019\\_03.pdf](https://www.illumio.com/sites/default/files/Illumio_Infographic_How_to_Choose_Your_Segmentation_Strategy_2019_03.pdf)