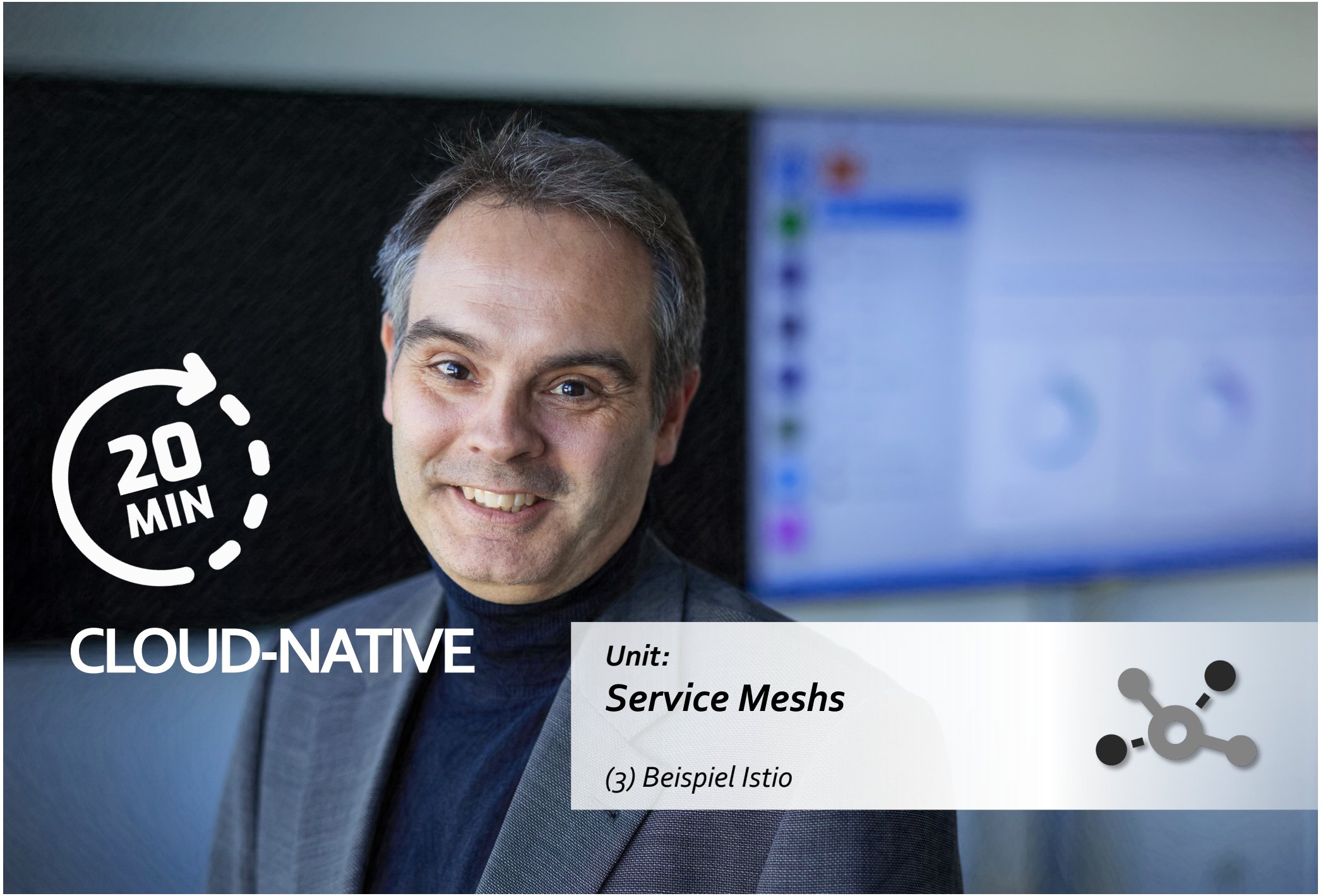


CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Service Meshs

(3) Beispiel Istio

creative commons

CLOUD NATIVE COMPUTING

Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.

KAPITEL 13

Automatisierte Instrumentierung mittels Service Meshs

13.1 Konsolidierung von Telemetriedaten

13.2 Instrumentierung von Systemen

- Logging
- Monitoring
- Tracing

13.3 Automatisierte Instrumentierung

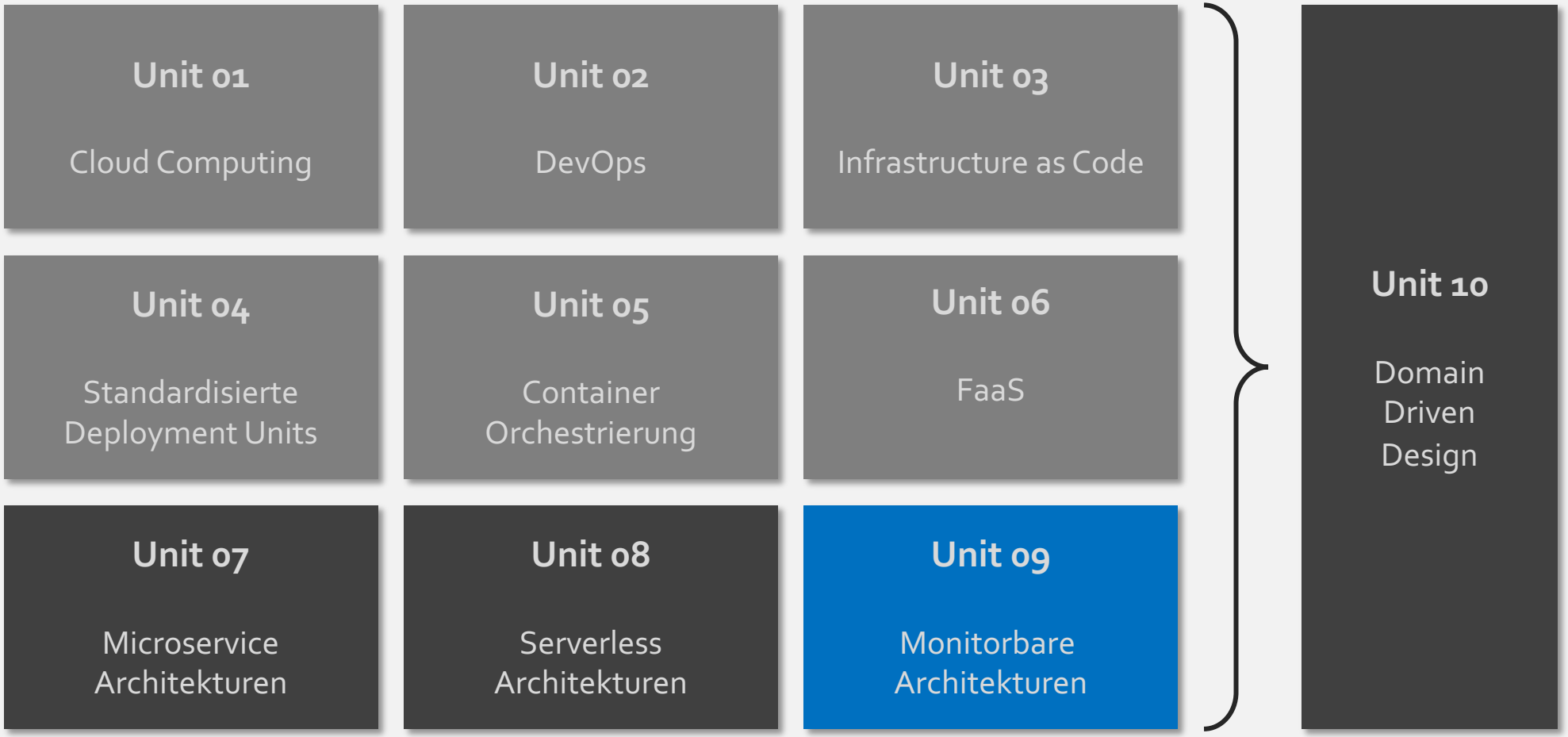
- Service Meshs
- Traffic-Management
- Resilienz
- Sicherheit
- Management und Analyse von Verkehrstopologien

13.4 Zusammenfassung

CLOUD NATIVE COMPUTING

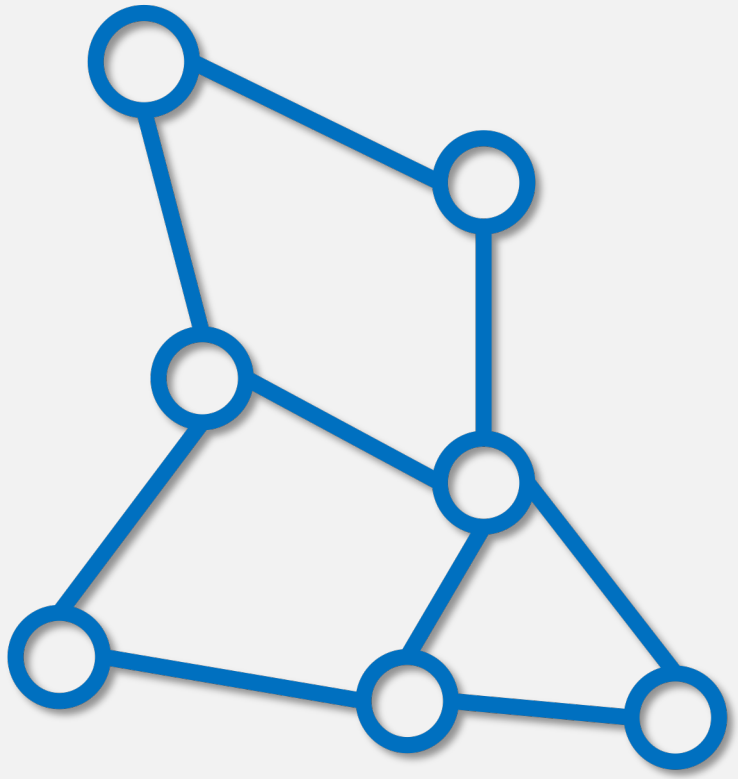
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

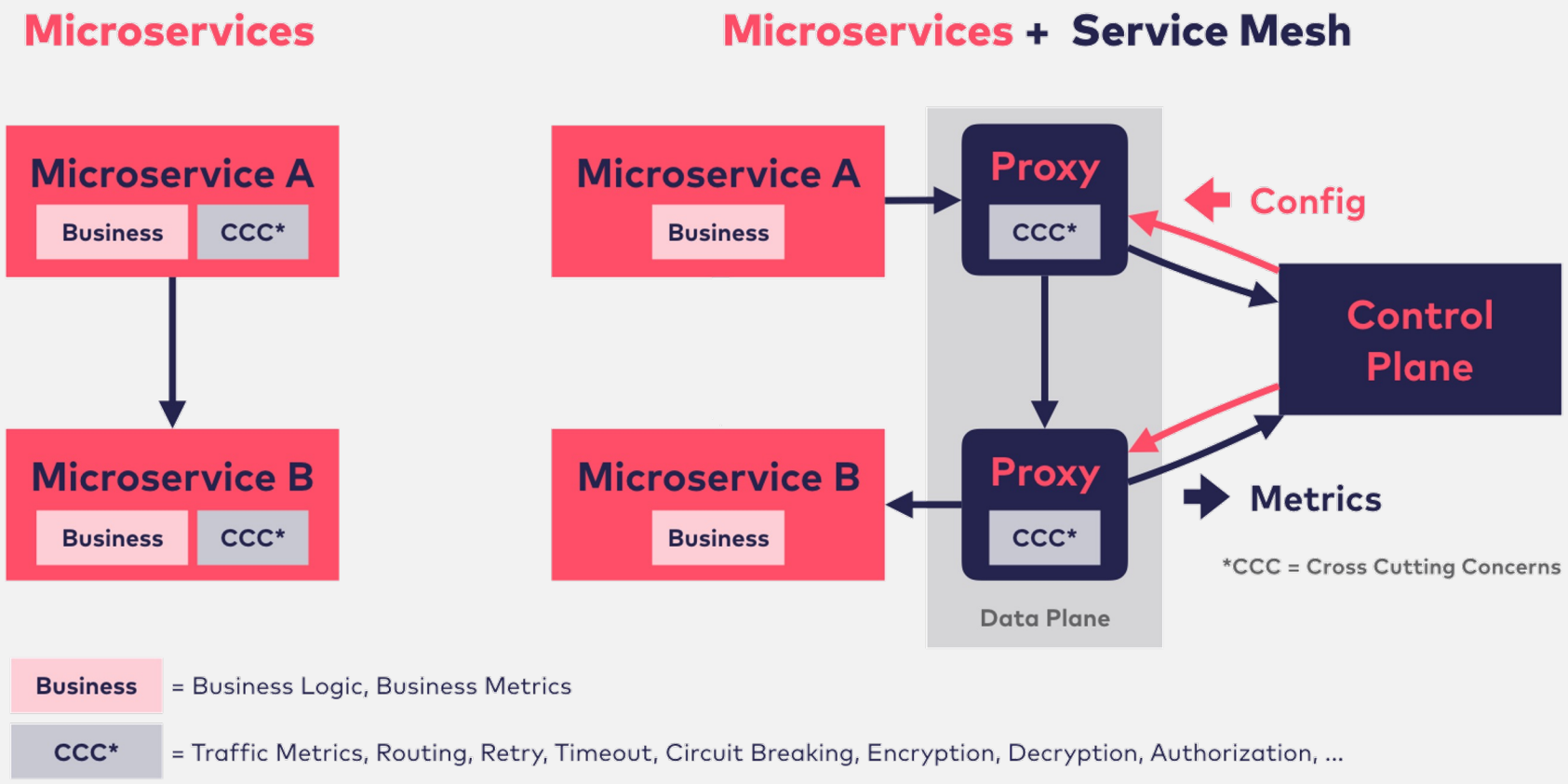
- **Was sind Service Meshs?**
- **SMI-Standard**
- **Funktionalitäten von Service Meshs am Beispiel von Istio**
  - Traffic Management
  - Resilienz
  - Sicherheit
  - Beobachtbarkeit



CLOUD NATIVE COMPUTING

SERVICE MESH

TLDR: Handhabung von Microservice Cross-Cutting-Concerns mittels des Sidecar-Patterns

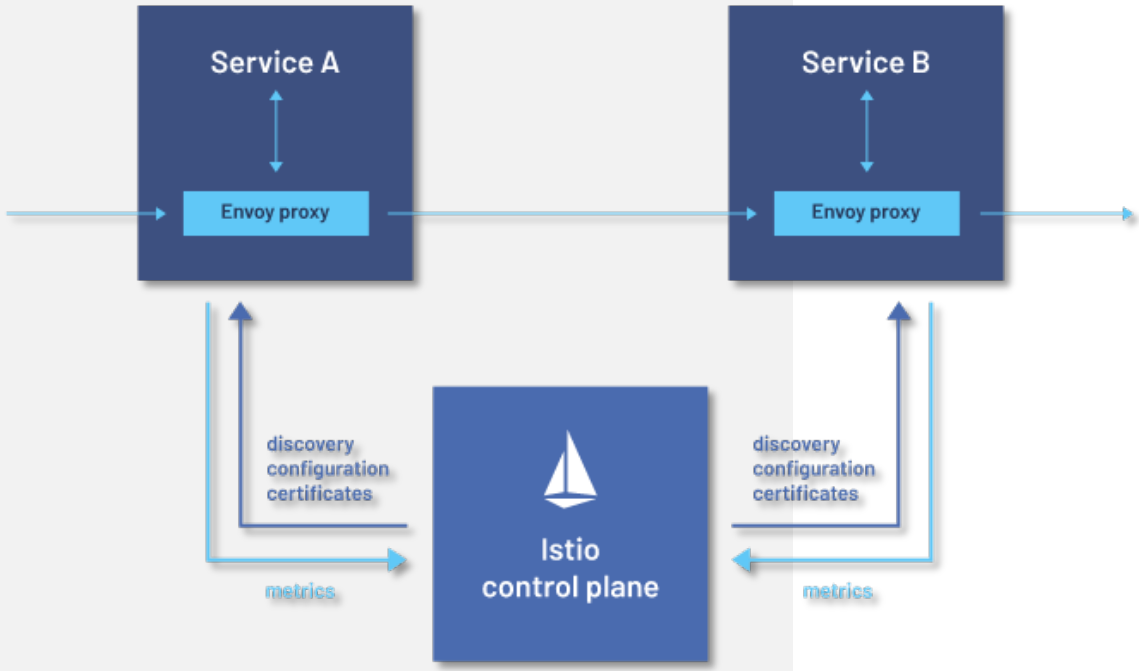


Bildquelle: servicemesh.es

ISTIO

Typvertreter für Service Meshs

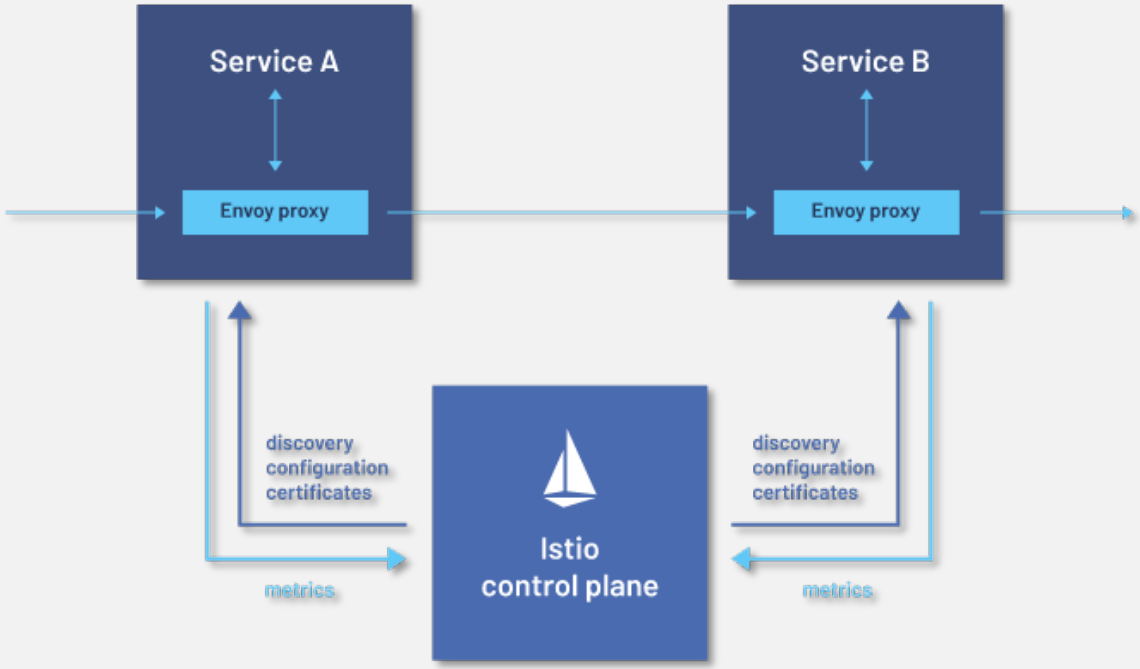
- Istio ist ein Open-Source Service Mesh, das entwickelt wurde, um den Betrieb von Microservices zu vereinfachen.
- Es wurde von Google ins Leben gerufen und wird von der Cloud Native Computing Foundation (CNCF) unterstützt.
- Istio bietet eine Reihe von Funktionen, die die Sicherheit, Zuverlässigkeit und Beobachtbarkeit von Microservices verbessern sollen.
- Es kann den Datenverkehr zwischen verschiedenen Microservices zu kontrollieren, und erlaubt es, Verkehrsregeln wie Lastverteilung und Failover zu konfigurieren.
- Es bietet Funktionen für das Monitoring von Anwendungen und kann beispielsweise Metriken für den Datenverkehr von Microservices sammeln.



ISTIO

Typvertreter für Service Meshs

- **Traffic Management**
- Resilienz
- Sicherheit
- Beobachtbarkeit



CLOUD NATIVE COMPUTING

ISTIO

Traffic Management

- Mit Traffic-Routing-Regeln lässt sich der Fluss des Datenverkehrs und der API-Aufrufe zwischen Services steuern
- Istio erleichtert das Einrichten wichtiger Aufgaben wie A/B-Tests, Canary-Rollouts und gestaffelte Rollouts mit prozentualer Traffic-Aufteilung.
- Außerdem bietet es Out-of-Box-Failure-Recovery-Funktionen, die dazu beitragen, Services robuster gegen Ausfälle von Upstream-Services oder des Netzwerks zu machen.

Regeln werden mithilfe von **Kubernetes Custom Resource Definitions (CRDs)** spezifiziert, die sich Kubernetes-üblich mittels YAML ausdrücken lassen.

Mittels diesen Ressourcen lässt sich ein Istio Service Mesh definieren:

- **Virtual Services**
- **Destination Rules**
- Gateways
- Service Entries
- Sidecars

Quelle: istio.io

ISTIO

Traffic Management (Virtual Services)

- Mit **Virtual Services** lässt sich in Istio konfigurieren, wie Requests an einen Service innerhalb eines Istio-Service-Meshs weitergeleitet werden.
- Jeder Virtual Service besteht aus Routing-Regeln, die der Reihe nach ausgewertet werden, so dass Istio jeden Request an den virtuellen Service mit einem bestimmten realen Service innerhalb des Meshs abgleichen kann.
- **Virtual Services entkoppeln so Services durch einen Routing Mittler.**
- Mittels Virtual Services hat man so umfangreiche Möglichkeiten das Routing von Datenverkehr in einem Mesh festzulegen.

Beispiel eines Nutzer-basierten Routings

Ein typischer Anwendungsfall ist das Umleiten von Datenverkehr an verschiedene Versionen eines Services, die als Serviceuntergruppen angegeben werden. Services senden Anfragen an den virtuellen Service-Host, als ob es sich um eine einzelne Entität handeln würde, und der Proxy leitet den Verkehr dann je nach den Regeln für den virtuellen Service an die verschiedenen Versionen weiter. Solche Regeln können bspw. sein

- "20 % der Aufrufe gehen an die neue Version"
- *"Aufrufe von diesen Benutzern gehen an Version 2".*

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
```

Kubernetes Service-Name

Definition von Subsets (nächste Folie)

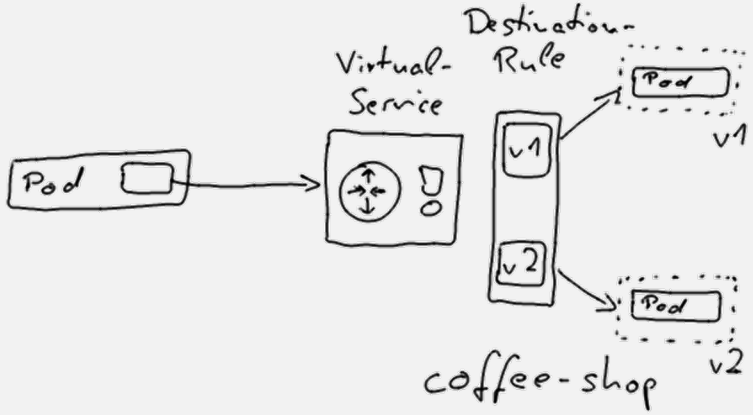
Routing-Regeln werden in sequentieller Reihenfolge von oben nach unten ausgewertet, wobei die erste Regel in der Definition des virtuellen Services die höchste Priorität erhält.

Quelle: istio.io

ISTIO

Traffic Management (Destination Rules)

- **Destination Rules** sind ein weiterer wichtiger Bestandteil der Traffic-Routing-Funktionalität von Istio.
- Virtuelle Services haben die Aufgabe Datenverkehr zu einem bestimmten Ziel zu leiten und dann dort Destination Rules am Ziel des Datenverkehrs anzuwenden.
- Destination Rules gelten also für den "Endanflug" des Datenverkehrs.
- Man verwendet Destination Rules, meist, um benannte Service-Untergruppen anzugeben, z. B. die Gruppierung aller Instanzen eines bestimmten Services nach Version.



Istio unterstützt in Destination Rules u.a. die folgenden Lastverteilungsmodelle:

- **ROUND ROBIN (default):** Alle Serviceinstanzen erhalten der Reihe nach Anfragen.
- **RANDOM:** Anfragen werden nach dem Zufallsprinzip an Instanzen im Pool weitergeleitet.
- **WEIGHTED:** Anfragen werden nach einem bestimmten Prozentsatz an Instanzen im Pool weitergeleitet.
- **LEASTS REQUESTS:** Anfragen werden an Instanzen mit der geringsten Anzahl von Anfragen weitergeleitet.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: my-destination-rule
spec:
  host: my-svc
  trafficPolicy:
    loadBalancer:
      simple: RANDOM
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
    trafficPolicy:
      loadBalancer:
        simple: ROUND_ROBIN
  - name: v3
    labels:
      version: v3
```

Subset Definition

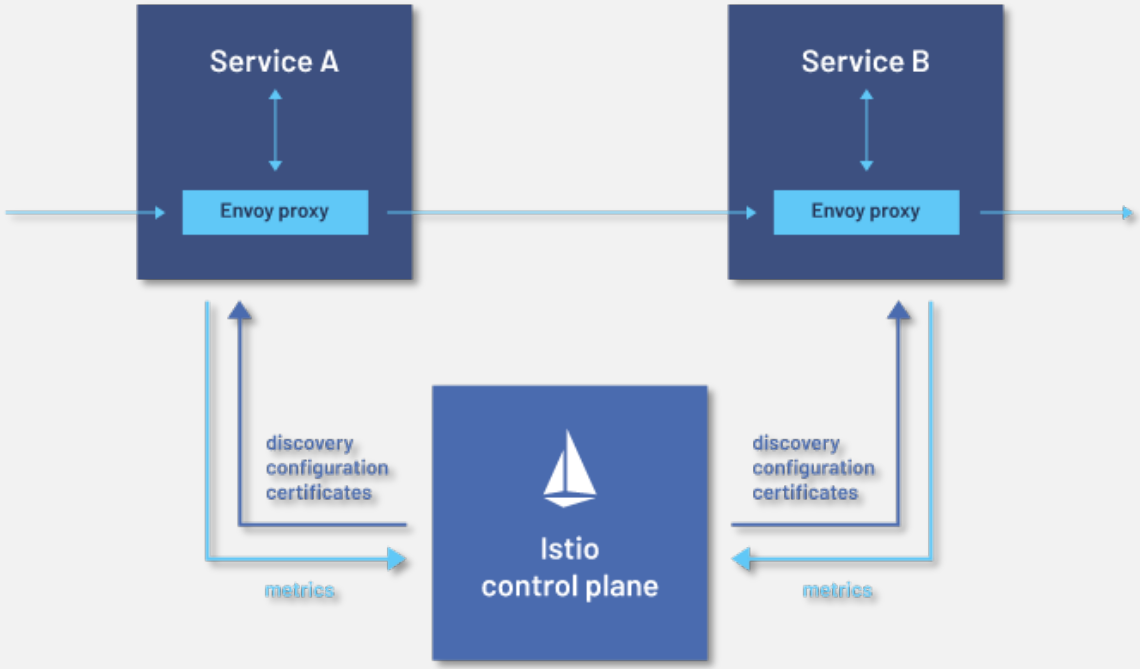
Beispiel: *Diese Destination Rule konfiguriert drei verschiedene Untergruppen „v1", „v2", „v3" für den Service „my-svc" mit unterschiedlichen Load-Balancing Stratgien.*

Quelle: istio.io

ISTIO

Typvertreter für Service Meshs

- Traffic Management
- **Resilienz**
- Sicherheit
- Beobachtbarkeit

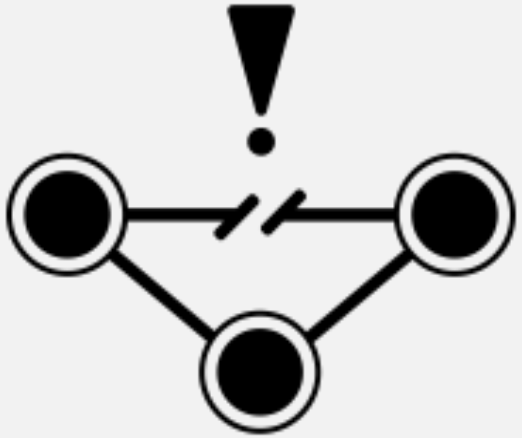


CLOUD NATIVE COMPUTING

ISTIO

Zuverlässigkeitsmechanismen

- Unter Resilienz versteht man die Widerstandsfähigkeit von Systemen gegenüber Teilsystemausfällen.
- Resilienz lässt sich in Systemen unter anderem durch Wahl geeigneter Einstellungen von folgenden Mechanismen erreichen:
  - **Timeouts**
  - **Wiederholungen**
  - **Circuit Breaker**
- Die gewählten Einstellungen lassen sich mittels **Fault Injection** testen und optimieren.



ISTIO

Resilienz (Timeouts)

- **Ein Timeout ist die Zeitspanne, die ein Proxy auf Antworten von einem bestimmten Service warten sollte**, um sicherzustellen, dass Requests innerhalb eines vorhersehbaren Zeitrahmens erfolgreich sind oder fehlschlagen.
- Für einige Anwendungen und Service ist der Standard-Timeout des Betriebssystems jedoch möglicherweise nicht geeignet.
- Ein zu langes Timeout könnte beispielsweise zu einer übermäßigen Latenz in Fehlersituationen führen.
- Ein zu kurzes Timeout könnte dazu führen, dass Requests unnötig fehlschlagen, an denen mehrere Dienste beteiligt sind.
- Um optimale Timeout-Einstellungen zu verwenden, lassen sich mit Istio Timeouts auf einfache Weise auf Ebene virtueller Services dynamisch pro Service anpassen.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    timeout: 10s
```

Für diesen virtuellen Service wird bspw. eine Timeout für Requests an die v1-Untermenge des „ratings" Service festlegt.

Quelle: istio.io

ISTIO

Resilienz (Retries)

- **Wiederholungen** geben an, wie oft ein Proxy maximal versucht, eine Verbindung zu einem Service herzustellen, wenn der erste **Aufruf fehlschlägt**.
- Wiederholungen können die Verfügbarkeit von Services und damit die Widerstandsfähigkeit des Gesamtsystems gegenüber Fehlerzuständen verbessern.
- Wiederholungen stellen sicher, dass Requests nicht dauerhaft aufgrund von vorübergehenden Problemen wie einem überlasteten Service oder Netzwerk fehlschlagen.
- Das Standard-Wiederholungsverhalten für Requests sind bspw. zwei Wiederholungsversuche (kann aber variabel und automatisch angepasst werden).
- Wie bei Timeouts kann auch das Wiederholungsverhalten Service-spezifisch konfiguriert werden.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
    retries:
      attempts: 3
      perTryTimeout: 2s
```

Dieses Beispiel konfiguriert maximal 3 Wiederholungsversuche, um nach einem anfänglichen Anruffehler eine Verbindung zu dem Service-Subset „v1“ herzustellen, jeweils mit einem Timeout von 2 Sekunden.

Quelle: istio.io

ISTIO

Resilienz (Circuit Breaker)

- **Circuit Breaker sind ein weiterer Mechanismus für die Erstellung von resilienten Microservice-basierten Anwendungen.**
- In einem Circuit Breaker werden **Requestlimits** für Services festgelegt, z. B. die Anzahl der gleichzeitigen Verbindungen oder wie oft Aufrufe an einen Service fehlgeschlagen sind.
- Sobald dieses Limit erreicht ist, wird der Circuit Breaker "ausgelöst" und stoppt weitere Verbindungen.
- Die Verwendung eines Circuit-Breaker-Musters ermöglicht einen schnellen Ausfall von Services (fail early!!!), anstatt dass Clients versuchen, eine Verbindung zu überlasteten oder ausfallenden Services herzustellen.
- Wie bei Timeouts und Wiederholungen können auch Circuit-Breaker Settings Service-spezifisch konfiguriert werden.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
    trafficPolicy:
      connectionPool:
        tcp:
          maxConnections: 100
```

Dieses Beispiel begrenzt die Anzahl der gleichzeitigen Verbindungen für die v1 Version des Review-Service auf 100.

Quelle: istio.io

ISTIO

Resilienz (Fault Injection)

- Fault Injection erzeugt Fehler künstlich in einem System, um sicherzustellen, dass es Fehlerbedingungen standhalten und sich davon erholen kann, um die Verfügbarkeit kritischer Services sicherzustellen.
- Auf diese Weise kann man Anwendungs-spezifische Fehler, wie z. B. HTTP-Fehlercodes, injizieren, um relevantere Ergebnisse zu erhalten.
- Istio kann zwei Arten von Fehlern injizieren:
  - **Delays:** Verzögerungen sind Timing-Fehler. Sie imitieren eine erhöhte Netzwerklatenz oder einen überlasteten Upstream-Service.
  - **Aborts:** Abbrüche sind Crash-Fehler. Sie imitieren Ausfälle in Upstream-Services. Abbrüche treten normalerweise in Form von HTTP-Fehlercodes oder TCP-Verbindungsfehlern auf.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - fault:
      delay:
        percentage:
          value: 0.1
        fixedDelay: 5s
    route:
    - destination:
        host: ratings
        subset: v1
```

Dieser virtuelle Service führt bei 1 von 1000 Requests zu einer Verzögerung von 5 Sekunden.

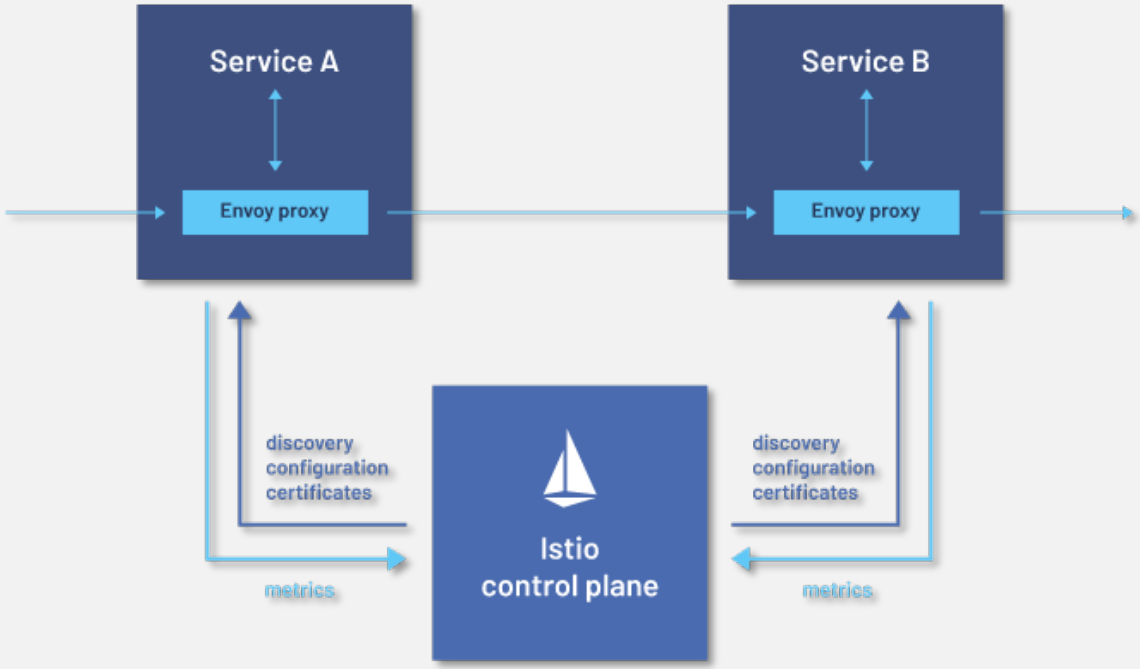
Fault Injection ist quasi Chaos-Engineering auf der Data Plane.

Quelle: istio.io

ISTIO

Typvertreter für Service Meshs

- Traffic Management
- Resilienz
- **Sicherheit**
- Beobachtbarkeit



CLOUD NATIVE COMPUTING

ISTIO

Security (Leitgedanken)

Die Zerlegung einer monolithischen Anwendung in atomare Services bietet verschiedene Vorteile, darunter eine bessere Agilität, Skalierbarkeit und eine Wiederverwendbarkeit von Services. Allerdings entstehen auch besondere Sicherheitsanforderungen:

- Um sich gegen **Man-in-the-Middle-Angriffe** zu schützen, benötigt man eine Verschlüsselung des Datenverkehrs.
- Um eine flexible **Service-Zugriffskontrolle** zu ermöglichen, benötigt man feinkörnige Zugriffsrichtlinien.
- Um festzustellen, wer was zu welchem Zeitpunkt getan hat, benötigt man **Auditing-Tools**.

Die Sicherheitsfunktionen von Istio bieten insbesondere eine transparente TLS-Verschlüsselung und Tools für Authentifizierung, Autorisierung und Audit (AAA). Die Leitgedanken dabei sind:

- **Security by default:** Keine Änderungen am Anwendungscode und der Infrastruktur erforderlich werden
- **Defense in Depth:** Integration mit bestehenden Sicherheitssystemen zur Bereitstellung mehrerer Verteidigungsebenen
- **Zero-Trust Networking:** Aufbau von Sicherheit auf vertrauensunwürdigen Netzwerken

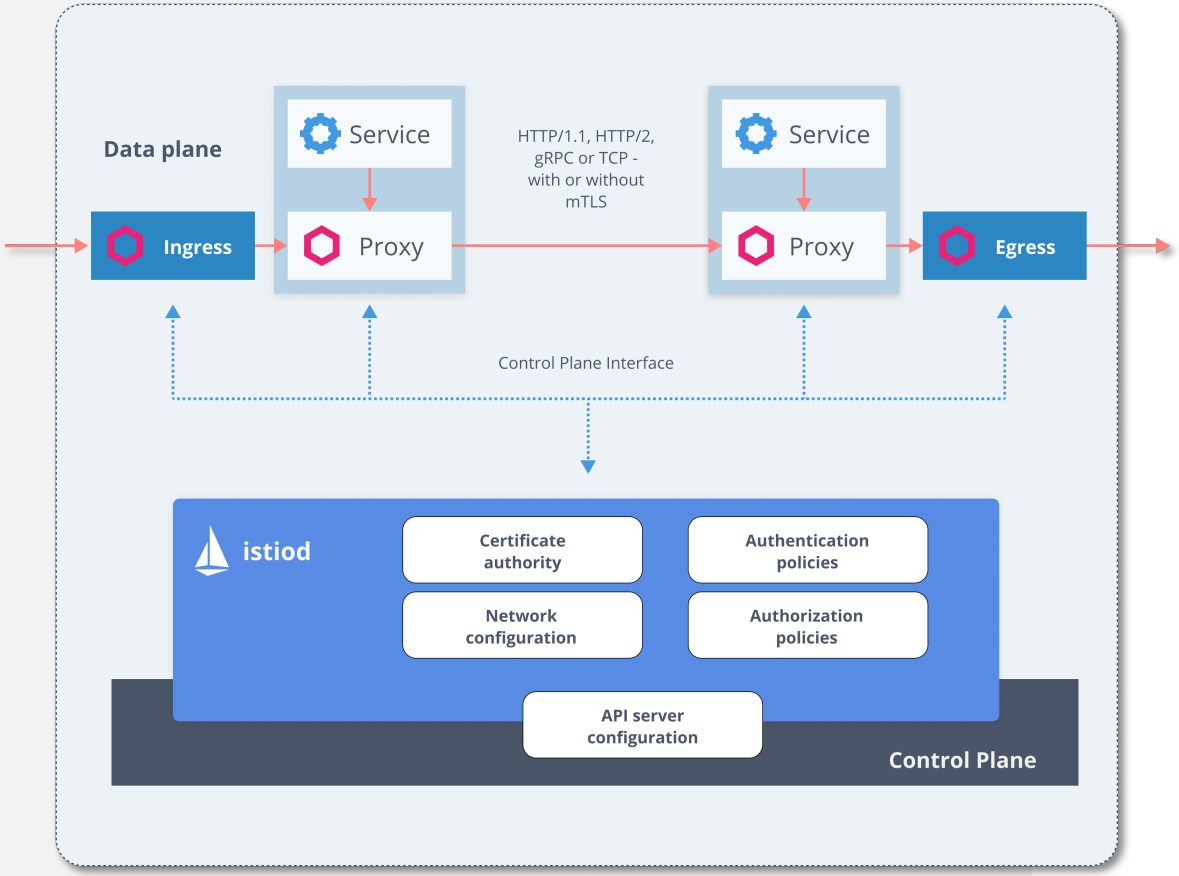
Quelle: istio.io

ISTIO

Security

Die Sicherheit wird in Istio durch mehrere Komponenten sichergestellt:

- **Eine Zertifizierungsstelle (CA) für die Schlüssel- und Zertifikatsverwaltung (Mutual TLS)**
- Der **Konfigurations-API-Server** zur Verteilung von Authentication Policies, Authorization Policies und Secure Naming Informationen an die Proxys.
- **Sidecar-** und **Perimeter-Proxys** arbeiten als **Policy Enforcement Points (PEPs),** um die Kommunikation zwischen Clients und Servern zu sichern.
- Istio-spezifische Envoy-Proxy-Erweiterungen zur Verwaltung von Telemetrie und Auditing.

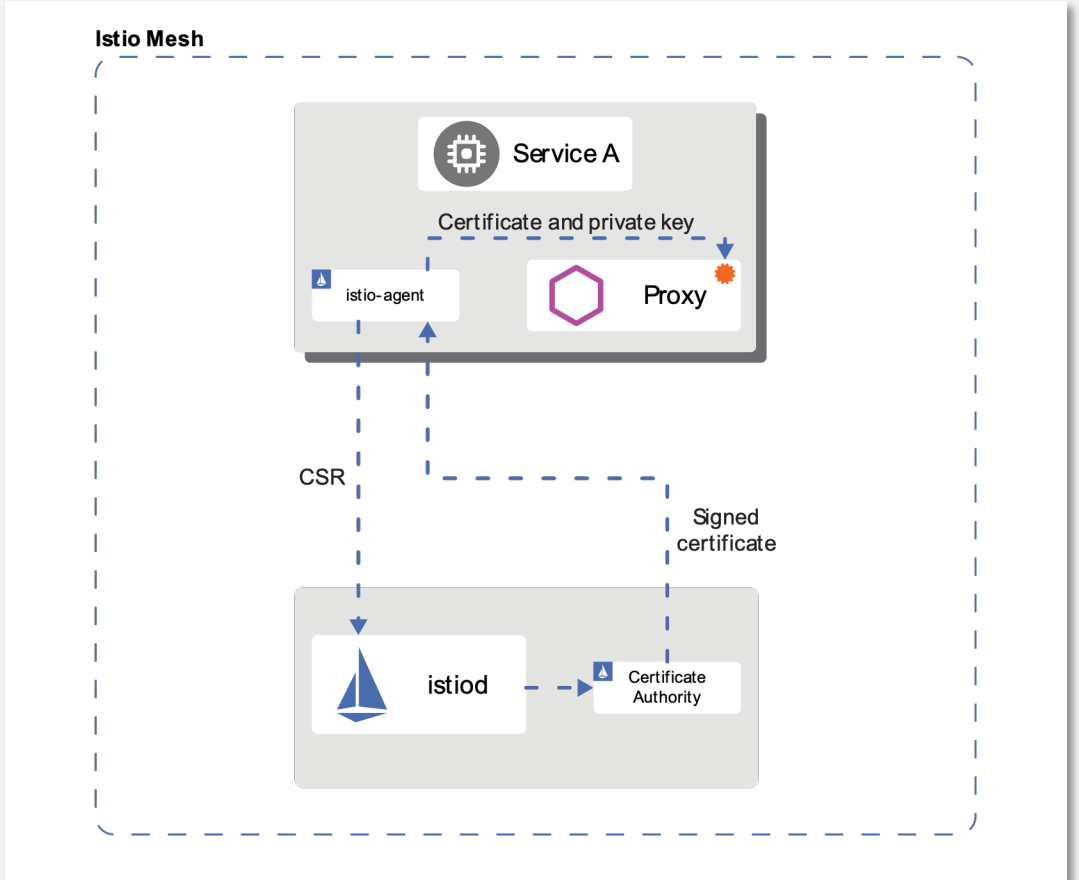


Quelle: istio.io

ISTIO

Security (Identity und Certificate Management)

- Die Identität ist ein grundlegendes Konzept jeder Sicherheitsinfrastruktur.
- Zu Beginn einer Workload-zu-Workload-Kommunikation müssen beide Parteien sich gegenseitigen authentifizieren.
- Auf der Client-Seite wird die Identität des Services anhand der sicheren Namensinformationen überprüft, um festzustellen, ob es sich um einen autorisierten Nutzer des Services handelt.
- Auf der Serverseite kann basierend auf den Autorisierungsrichtlinien bestimmt werden, auf welche Informationen der Client zugreifen darf.



Istio stellt mit X.509-Zertifikaten Identitäten für jeden Workload sicher bereit. Istio-Agenten, die neben jedem Proxy laufen, arbeiten mit Istio zusammen, um die Schlüssel- und Zertifikatsrotation zu automatisieren.

Quelle: istio.io

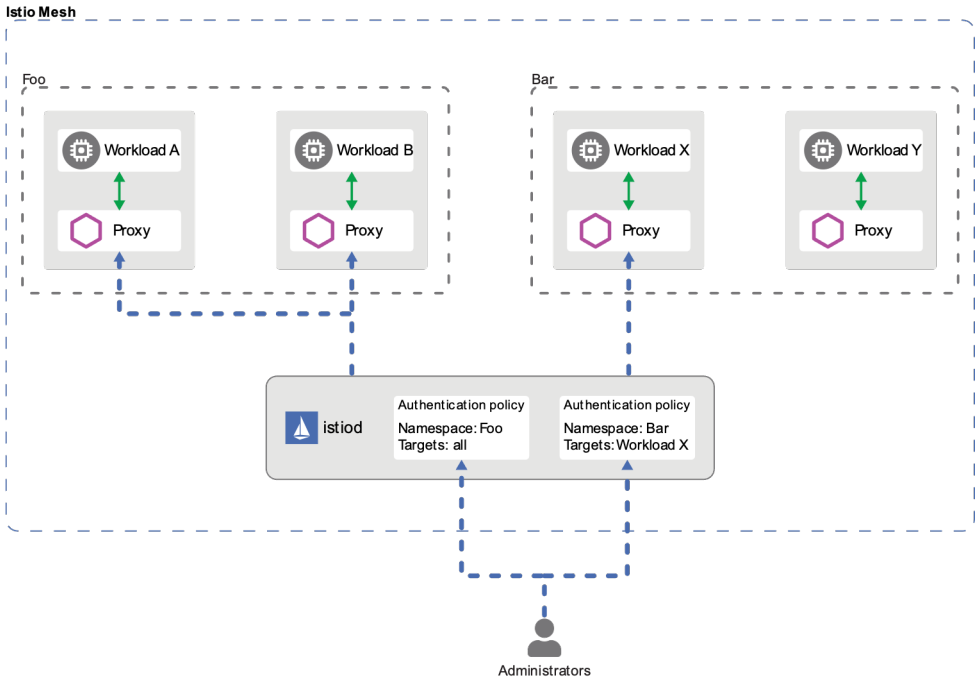
ISTIO

Security (Authentication Architektur)

Istio bietet zwei Arten der Authentifizierung:

- **Peer-Authentifizierung** wird für die Service-to-Service Kommunikation verwendet, um den Client zu verifizieren, der die Verbindung herstellt. Istio bietet Mutual TLS als Full-Stack-Lösung für die Transport-Authentifizierung, die ohne Änderungen am Service-Code aktiviert werden kann.

- **Request-Authentifizierung** wird für die Endbenutzer-Authentifizierung verwendet, um den an die Anfrage angehängten Berechtigungsnachweis zu verifizieren. Istio ermöglicht u.a. Authentifizierung mit JSON Web Token (JWT)-Validierung und die Verwendung benutzerdefinierter Authentifizierungsanbieters (OAUTH2) wie bspw. AuthO, Firebase Auth, Google Auth, Github, usw.



Die Authentifizierung für Services kann mithilfe von Peer- und Requests-Authentifizierungsrichtlinien festgelegt werden. Diese Authentication Policies werden im Istio-Konfigurationsspeicher gespeichert. Der Istio-Controller überwacht den Konfigurationsspeicher. Istio sendet diese Policies asynchron an die Ziel-Endpunkte. Sobald der Proxy eine Policy empfängt, wird diese sofort auf diesem Pod wirksam.

Quelle: istio.io

ISTIO

Security (Authentication Policies)

Peer-Authentifizierungsrichtlinien werden insbesondere genutzt, um Mutual TLS-Modus in einem Mesh durchzusetzen. Die folgenden Modi werden unterstützt:

- **PERMISSIVE:** Workloads akzeptieren sowohl mutual TLS als auch Klartextverkehr
- **STRICT:** Workloads akzeptieren nur gegenseitigen TLS-Verkehr.
- **DISABLE:** Wechselseitiges TLS ist deaktiviert. Nicht empfohlen, es sei denn, es wird eine eigene Sicherheitslösung bereitgestellt.

Mit Workload-spezifischen Peer-Authentifizierungsrichtlinien lassen sich auch verschiedene Mutual TLS-Modi für verschiedene Ports angeben.

```
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
metadata:
  name: "example-peer-policy"
  namespace: "foo"
spec:
  selector:
    matchLabels:
      app: reviews
  mtls:
    mode: STRICT
```

Beispiel:
Diese Peer-Authentifizierungsrichtlinie erfordert, dass alle Workloads im Namespace foo Mutual TLS verwenden.

```
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
metadata:
  name: "example-workload-policy"
  namespace: "foo"
spec:
  selector:
     matchLabels:
       app: example-app
  portLevelMtls:
    80:
      mode: DISABLE
```

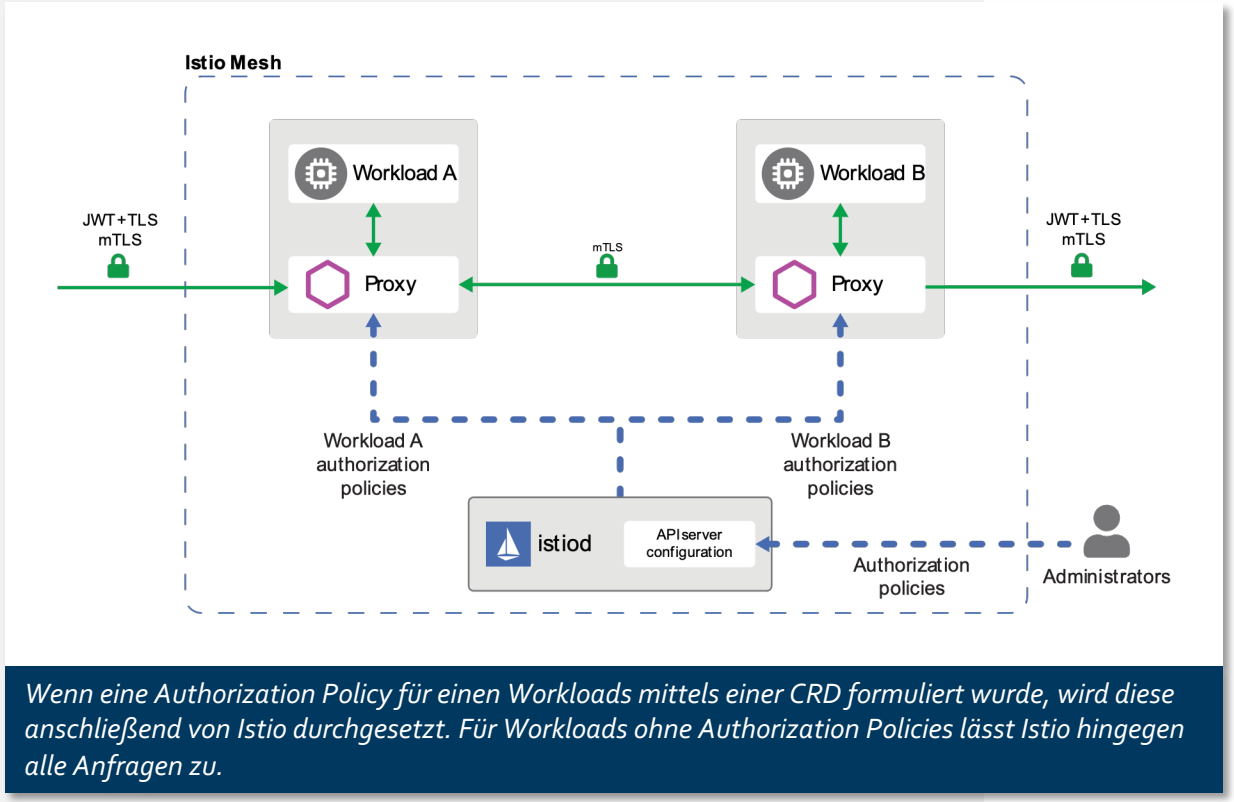
Beispiel:
Diese Richtlinie deaktiviert bspw. Mutual TLS an Port 80 für den Workload app:example-app und verwendet die Mutual TLS-Einstellungen der Namespace-weiten Peer-Authentifizierungsrichtlinie für alle anderen Ports

Quelle: istio.io

ISTIO

Security (Authorization Architektur)

- Die Autorisierungsfunktionen von Istio bieten eine **Mesh-**, **Namespace-** und **Workload-**weite **Zugriffskontrolle** für Workloads in einem Mesh.
- **Jeder Proxy führt hierzu eine Autorisierungs-Engine aus, die Requests zur Laufzeit autorisiert.**
- Wenn ein Request beim Proxy eingeht, wertet die Autorisierungs-Engine den Anfragekontext anhand der aktuellen Authorization Policy aus und gibt das Autorisierungsergebnis (ALLOW oder DENY) zurück.



Wenn eine Authorization Policy für einen Workloads mittels einer CRD formuliert wurde, wird diese anschließend von Istio durchgesetzt. Für Workloads ohne Authorization Policies lässt Istio hingegen alle Anfragen zu.

Quelle: istio.io

ISTIO

Security (Authorization Policies)

- **Authorization Policies können als ALLOW- oder DENY-Regeln formuliert werden.**
- DENY-Regeln haben Vorrang vor den ALLOW-Regeln.
- Beziehen sich mehrere Authorization Policies auf denselben Workload, wendet Istio diese additiv an.
- Mittels Selektoren kann die Anwendung von Authorization Policies auf bestimmte Workloads eingeschränkt werden.
- Selektoren enthalten hierzu (Kubernetes-üblich) eine Liste von {Schlüssel: Wert}-Paaren, wobei der Schlüssel der Name des Labels ist.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin-deny
 namespace: foo
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 action: DENY
 rules:
 - from:
   - source:
       notNamespaces: ["foo"]
```

Beispiel:
Diese DENY-Policy, verweigert Requests, wenn die Quelle nicht aus dem Namensraum foo stammt.

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-read
  namespace: default
spec:
  selector:
    matchLabels:
      app: products
  action: ALLOW
  rules:
  - to:
    - operation:
        methods: ["GET", "HEAD"]
```

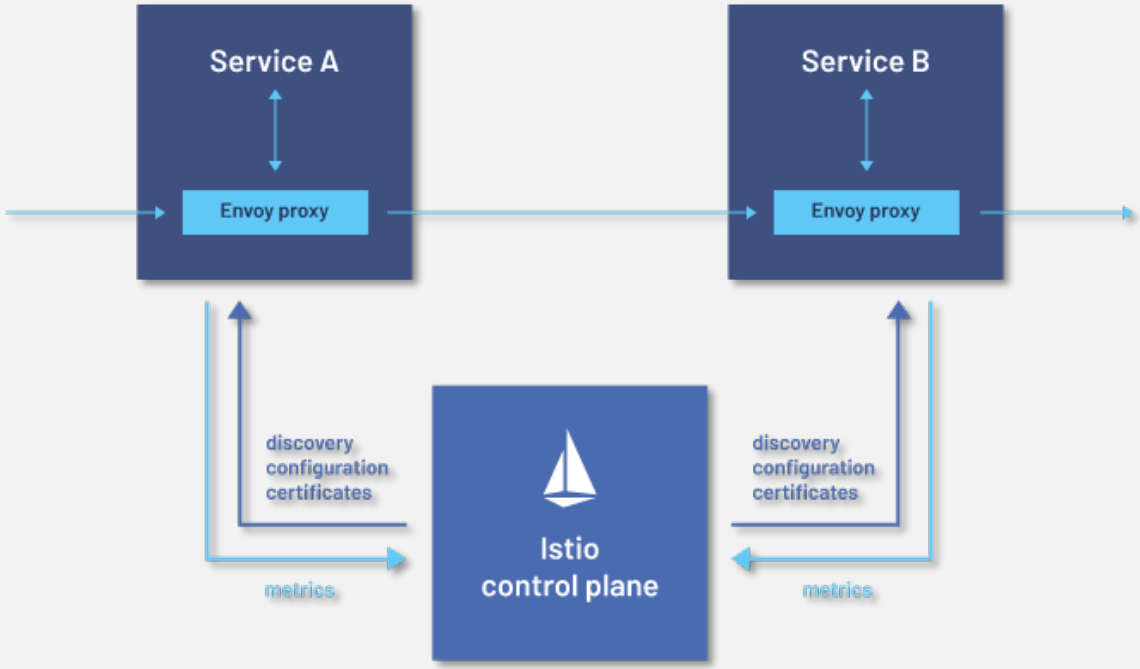
Beispiel:
Diese ALLOW-Policy erlaubt den "GET"- und "HEAD"-Zugriff auf den Workload mit dem Label "app: products" im Default-Namespace.

Quelle: istio.io

ISTIO

Typvertreter für Service Meshs

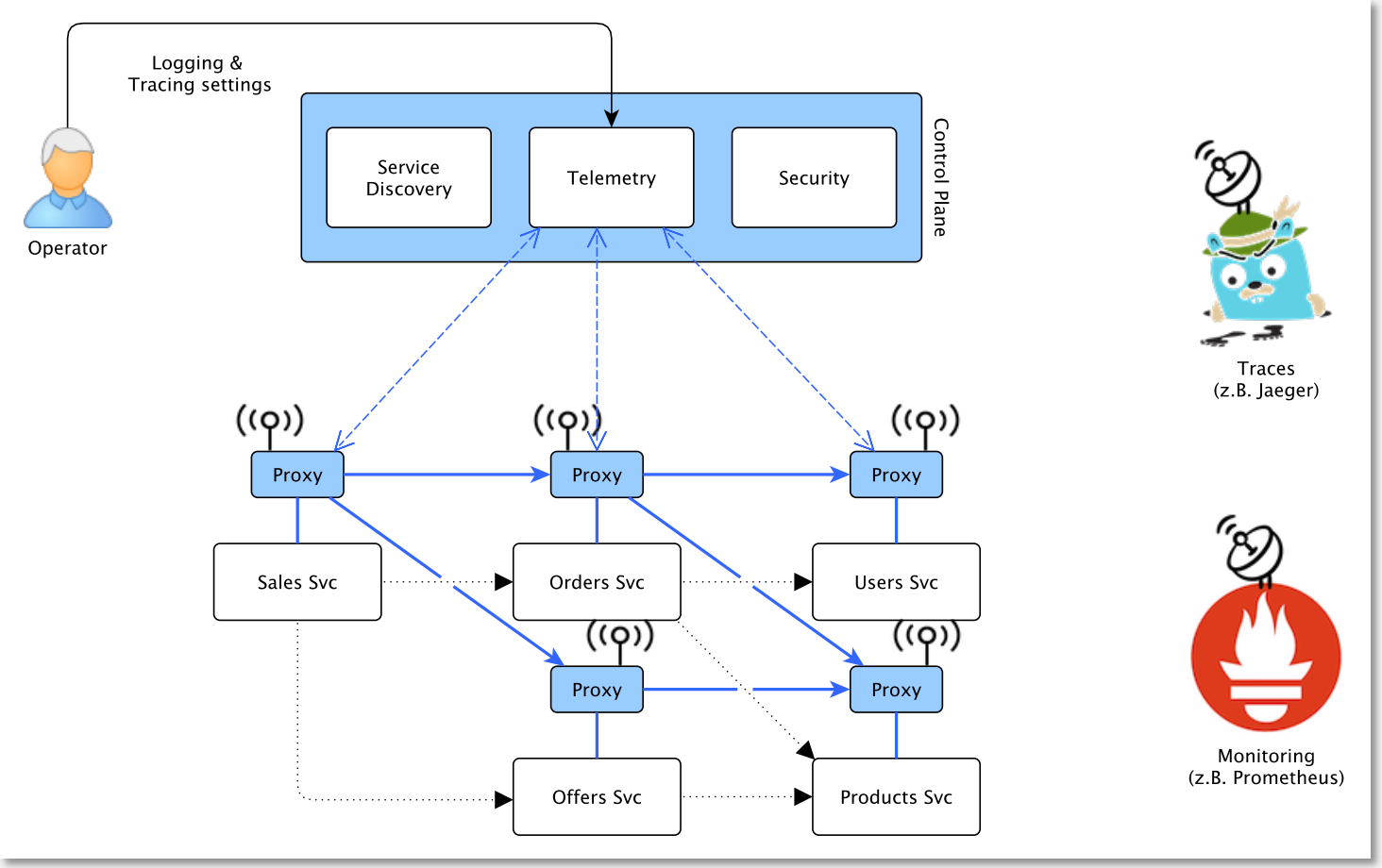
- Traffic Management
- Resilienz
- Sicherheit
- **Beobachtbarkeit**



CLOUD NATIVE COMPUTING

SERVICE MESH

Telemetrie und Beobachtbarkeit



Telemetriedaten können transparent an den Proxies des Service Meshs gesammelt warden, um zum einen Latenz-, Zustands- und Volumen-basiertes Load Balancing und Traffic-Management zu ermöglichen.

Dies ermöglicht zum einen Resilienz aber natürlich auch Beiträge zur Observability.

Merke:
Telemetrie umfasst

- *Tracing*
- *Monitoring*
- *Logging*

ISTIO

Observability

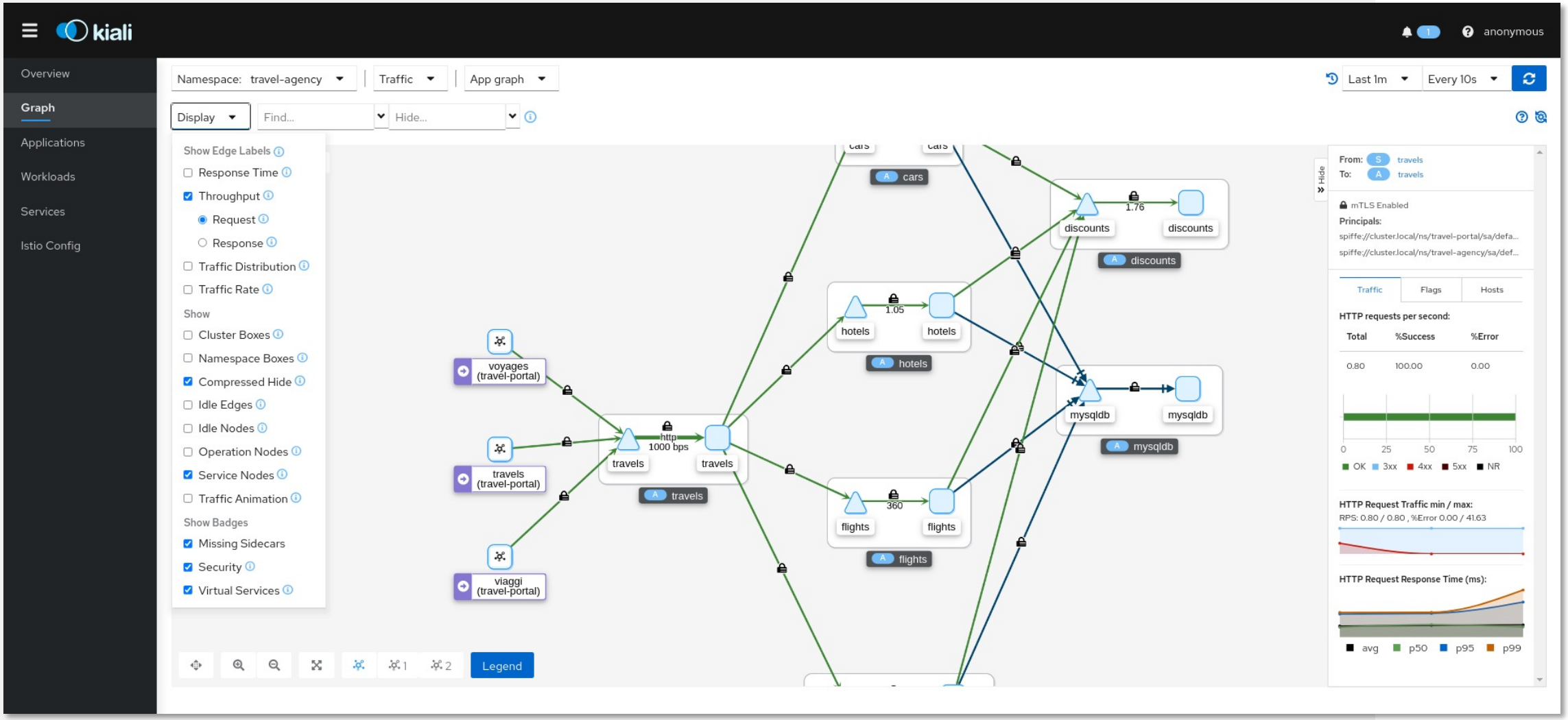
- Istio generiert detaillierte Telemetriedaten für die gesamte Service-Kommunikation innerhalb eines Meshes.
- Diese Daten ermöglichen die Beobachtung des Serviceverhaltens und ermöglichen im DevOps Feedback-Cycle, Fehler zu beheben und Anwendungen zu optimieren.

- **Metriken:**
  Istio generiert eine Reihe von Service-Metriken (Latenz, Traffic, Fehler und Sättigung).
- **Distributed Traces:**
  Istio generiert verteilte Trace-Spans für jeden Service, die den Betreibern ein detailliertes Verständnis der Request-Flüsse und Service-Abhängigkeiten innerhalb eines Meshes ermöglichen.
- **Access Logs:**
  Wenn Datenverkehr in einen Service innerhalb eines Mesh fließt, kann Istio eine vollständige Aufzeichnung jedes Requests generieren, einschließlich Quell- und Ziel-Metadaten. Diese Informationen ermöglichen es das Serviceverhalten bis auf die Ebene der einzelnen Workload-Instanzen zu überprüfen.

Quelle: istio.io

KIALI

Ermöglicht eine Form der Black-Box Observability und Visualisierung von Verkehrstopologien



Bildquelle: kiali.io

KIALI

Darstellung von Verkehrstopologien

Funktionalitäten

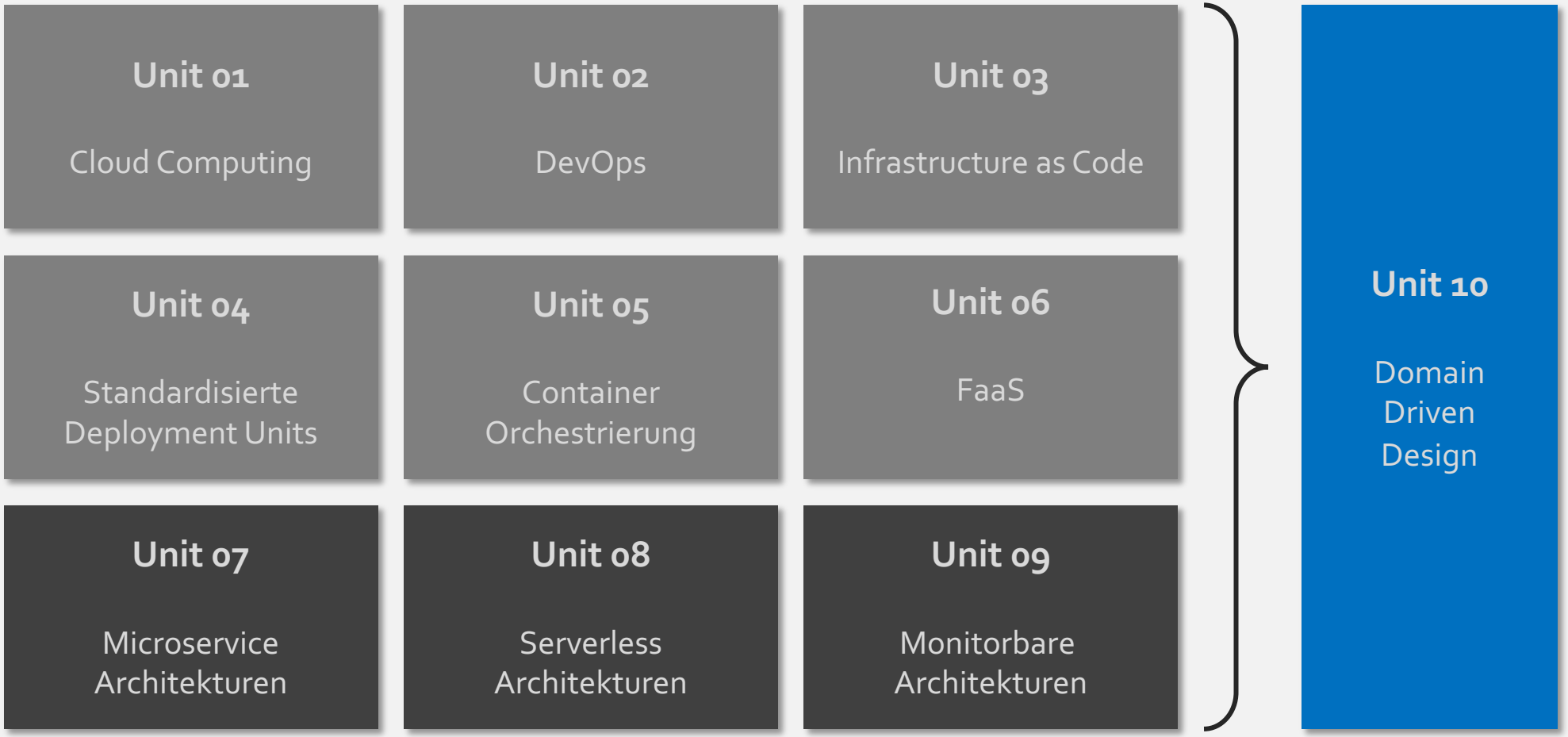
- **Topologie-Visualisierung**
  Interaktive Darstellung der Service-Topologie, die es ermöglicht Beziehungen zwischen den einzelnen Services zu verstehen
- **Leistungsüberwachung**
  Echtzeit-Darstellung von Metriken zu Datenverkehr, Latenz, Fehler und Ressourcennutzung
- **Fehlerbehebung**
  Visualisierung von Fehlern und Ausnahmen, die in der Anwendung auftreten können, sowie eine Möglichkeit zur Verfolgung von Anfragen (Distributed Tracing)
- **Konnektivitätsprüfung**
  Überprüfung der Konnektivität zwischen Services, einschließlich der Überprüfung von Service-Endpunkten.
- **Konfigurationsmanagement**
  Möglichkeit zur Verwaltung der Service-Mesh-Konfiguration, einschließlich der Konfiguration des Loadbalancings, Sicherheitseinstellungen und Routing-Regeln

CLOUD NATIVE COMPUTING

Kiali ist eine Open-Source-Plattform zur Visualisierung und Überwachung von Service-Mesh-Architekturen.

AUSBLICK

Überblick über Units und Themen dieses Moduls



KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de