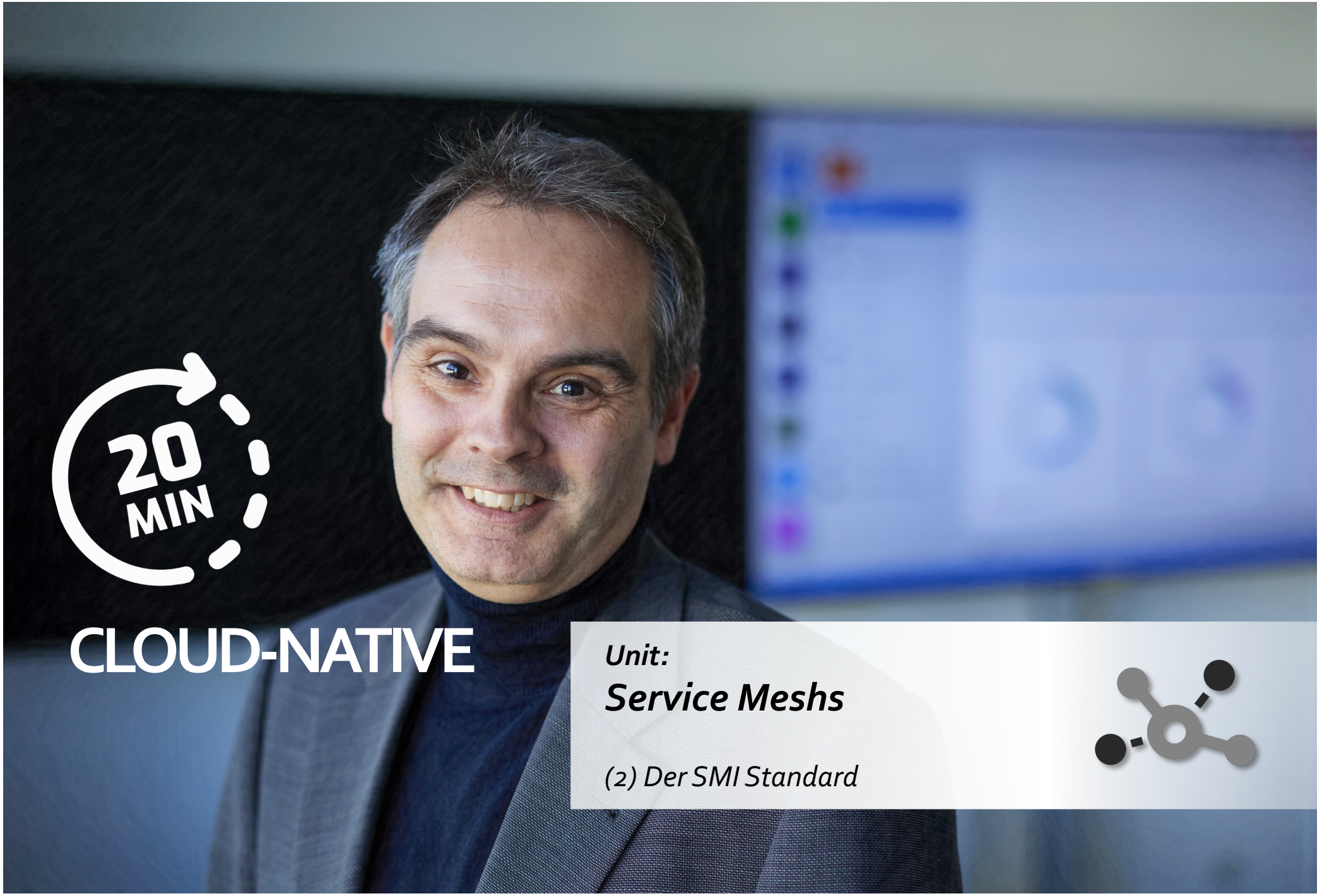


CLOUD NATIVE COMPUTING

CLOUD-NATIVE

Unit:

Service Meshs

(2) Der SMI Standard

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

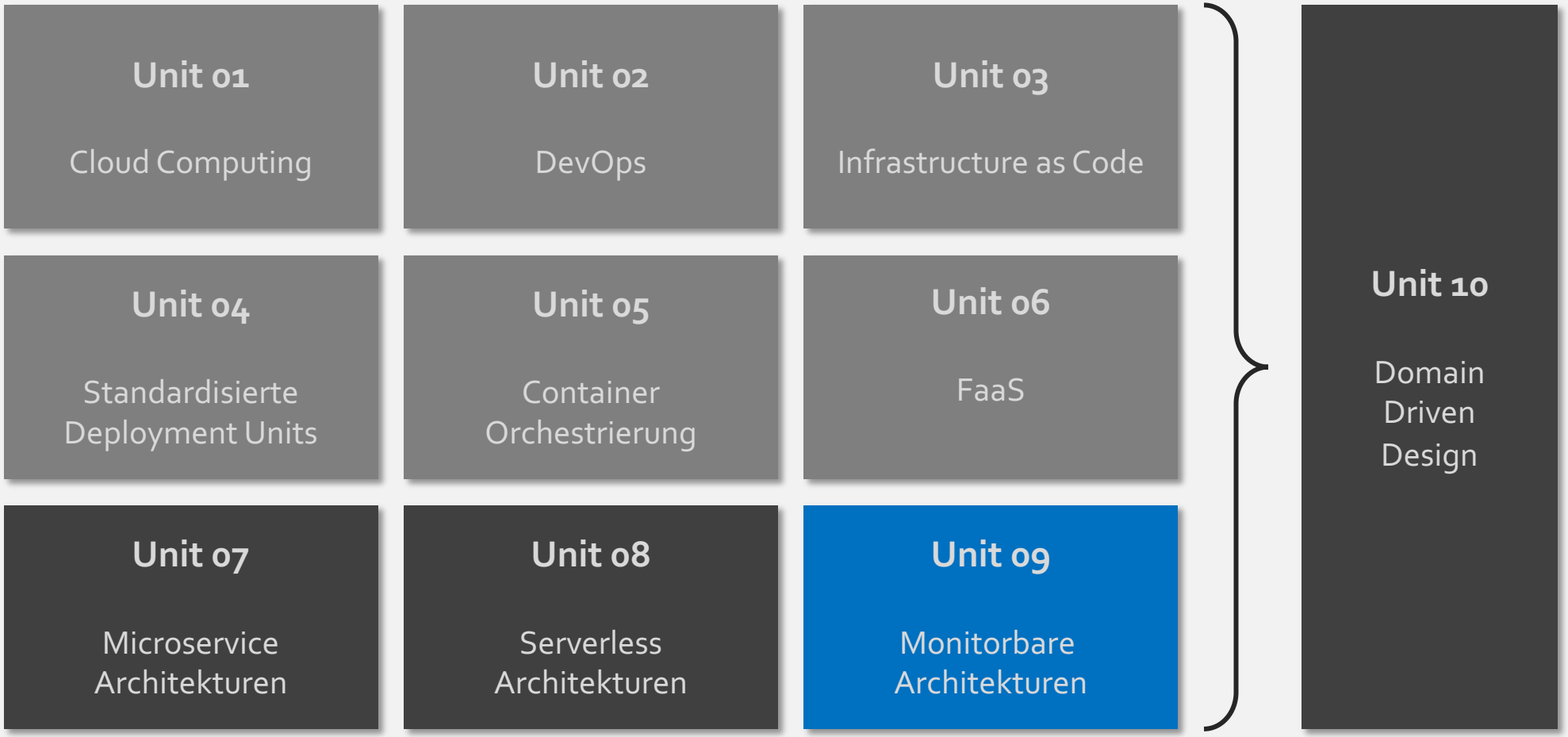
- Service Meshs
- Traffic-Management
- Resilienz
- Sicherheit
- Management und Analyse von Verkehrstopologien

13.4 Zusammenfassung

CLOUD NATIVE COMPUTING

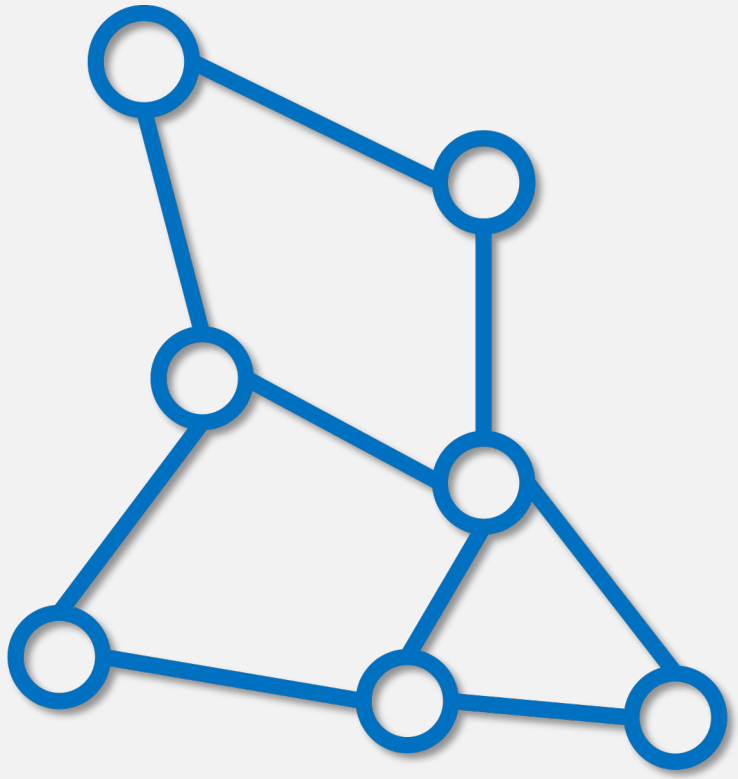
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



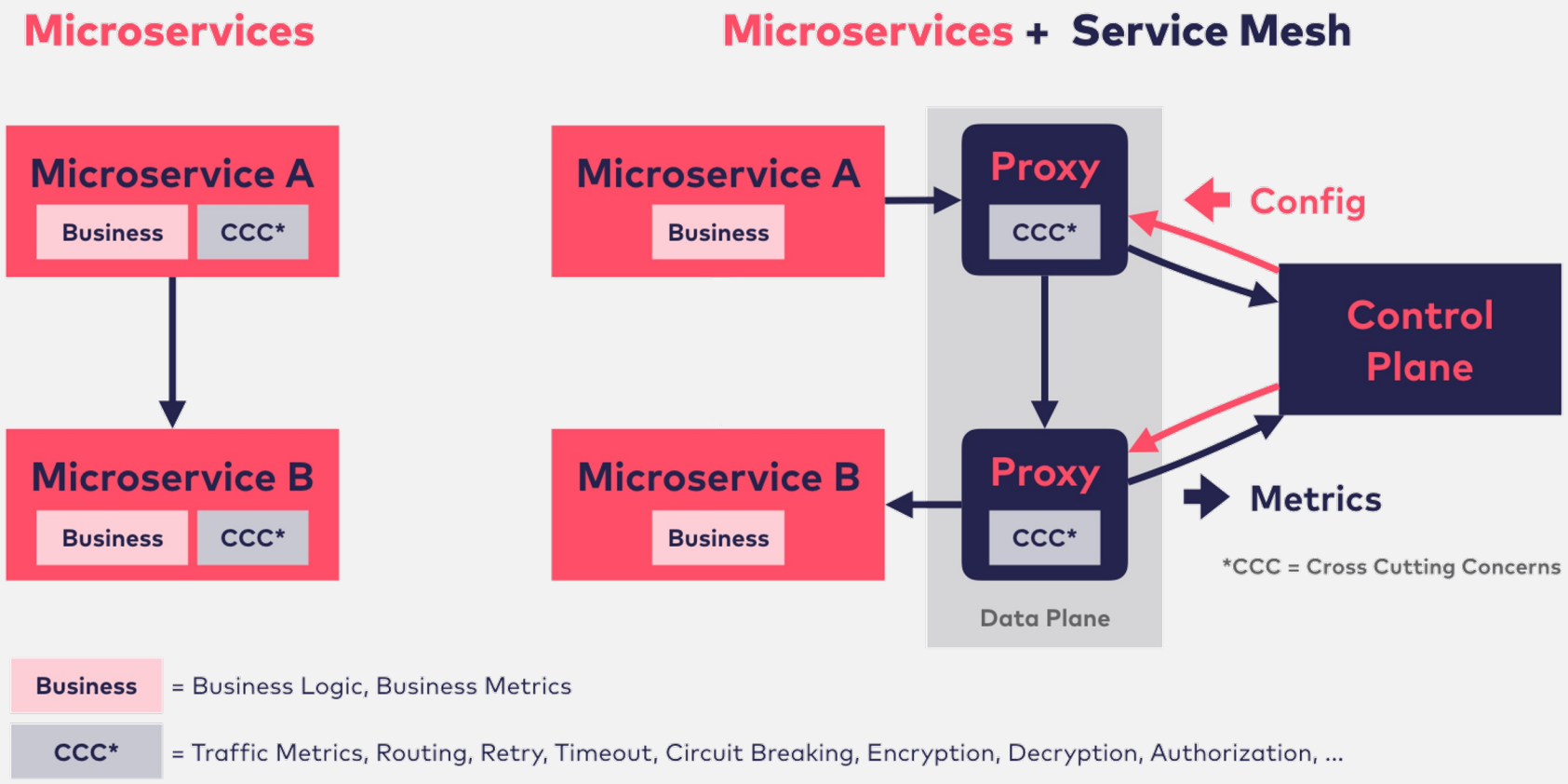
INHALTE

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



SERVICE MESH

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



Bildquelle: servicemesh.es

SERVICE MESH

Beispielprodukte

Kuma

Traefik

NGINX Service Mesh

Istio

Linkerd

Consul

CLOUD NATIVE COMPUTING

Auf servicemesh.es finden Sie eine Übersicht von vielen weiteren Service Meshs.

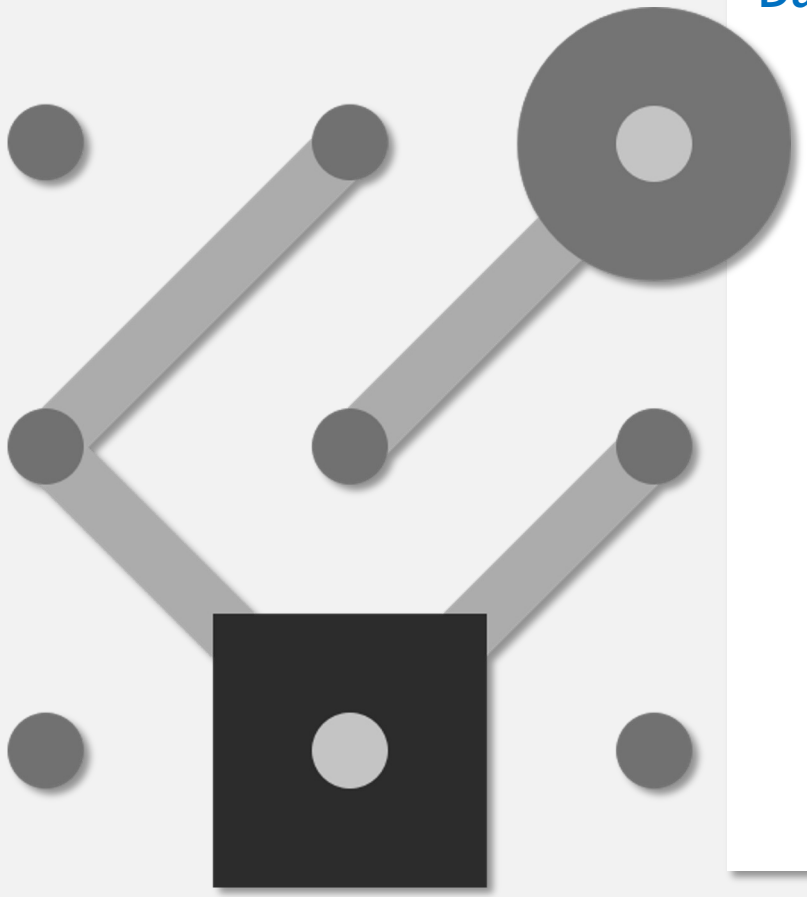
Es steht zu erwarten, dass sich dieser Bereich in den kommenden Jahren konsolidieren wird und stärker in Container Plattformen wie K8s integriert werden wird.

Mit dem SMI Standard (Service Mesh Interface) existiert ein erster Standardentwurf in diesem Bereich.

SERVICE MESH INTERFACE

Der SMI Standard

Das Service Mesh Interface (SMI) ist eine Spezifikation für Service Meshes, die auf Kubernetes ausgeführt werden. Es definiert einen gemeinsamen Standard, der von einer Vielzahl von Anbietern implementiert werden kann. Dies ermöglicht sowohl eine Standardisierung für Endbenutzer als auch Innovationen durch Anbieter von Service Mesh-Technologie. SMI ermöglicht Flexibilität und Interoperabilität und deckt die gängigsten Service-Mesh-Funktionen ab.



Das SMI ist:

Eine Standardschnittstelle für Kubernetes Service-Meshes und den grundlegenden Funktionsumfang von Service-Meshes definiert:

- ***Traffic Policy*** - Anwenden von Richtlinien wie Identität und Transportverschlüsselung zwischen Diensten
- ***Traffic Telemetry*** - Erfassen wichtiger Messdaten wie Fehlerrate und Latenz zwischen Diensten
- ***Traffic Management*** - Verschieben von Datenverkehr zwischen verschiedenen Diensten

Quelle: https://smi-spec.io

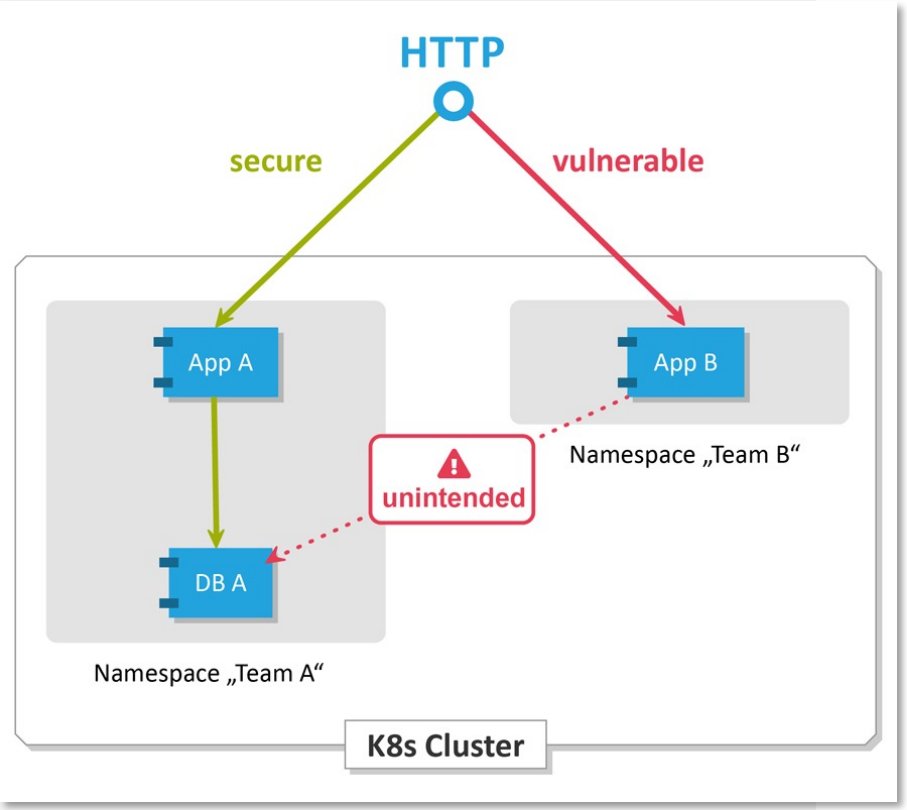
SERVICE MESH INTERFACE

Der SMI Standard

Traffic Policy
Anwenden von Richtlinien wie Identität und Transportverschlüsselung zwischen Diensten

Traffic Telemetry
Erfassen wichtiger Messdaten wie Fehlerrate und Latenz zwischen Diensten

Traffic Management
Verschieben von Datenverkehr zwischen verschiedenen Diensten



Quelle: https://smi-spec.io

DER SMI STANDARD

Traffic Access Control (I)

Mittels SMI können Zugriffsrichtlinien für Anwendungen definiert werden.

- Die **Zugriffskontrolle ist additiv**.
- Datenverkehr wird **standardmäßig verweigert**, es sei denn es werden Zugriffsregeln explizit ergänzt.

- Ein Traffic Target ordnet eine Reihe von Traffic Definitions (Regeln) einer Dienstidentität zu.
- Regeln sind Traffic Definitions, die definieren, wie zulässiger Verkehr für bestimmte **Protokolle** aussehen darf
  - Der Zugriff wird über referenzierte Traffic Specs (Protokolle) und eine Liste von **Quelldienstidentitäten** gesteuert.
  - Jeder Pod, der versucht, eine Verbindung herzustellen und nicht in der definierten **Liste der Quellen** enthalten ist, wird abgelehnt.
  - Der Zugriff wird basierend auf der Dienstidentität (Kubernetes-Serviceaccounts) gesteuert.

Quelle: https://smi-spec.io

DER SMI STANDARD

Traffic Access Control (II)

```yaml
kind: HTTPRouteGroup
metadata:
  name: the-routes
spec:
  matches:
  - name: metrics
    pathRegex: "/metrics"
    methods: ["GET"]
```

Routes

```yaml
kind: TrafficTarget
metadata:
  name: path-specific
  namespace: default
spec:
  sources:
  - kind: ServiceAccount
    name: prometheus
    namespace: monitoring
  destination:
    kind: ServiceAccount
    name: service-a
    namespace: my-app
  rules:
  - kind: HTTPRouteGroup
    name: the-routes
    matches: ["metrics"]
```

Zugriffsregel

Dieses Beispiel zeigt einen häufigen Anwendungsfall, den Zugriff auf Metriken so einzuschränken, dass diese nur von Prometheus abgefragt werden dürfen.

Quelle: https://smi-spec.io

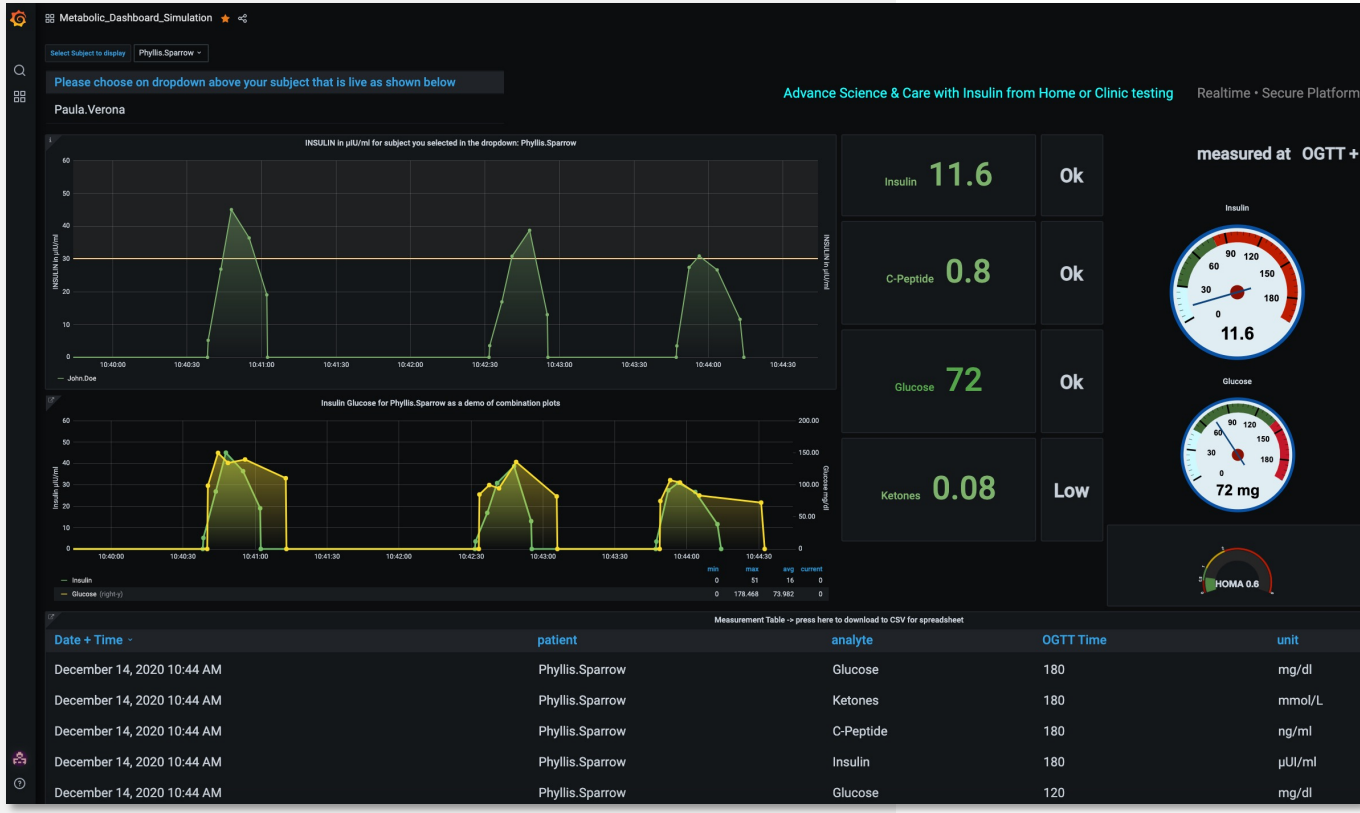
SERVICE MESH INTERFACE

Der SMI Standard

Traffic Policy
Anwenden von Richtlinien wie Identität und Transportverschlüsselung zwischen Diensten

Traffic Telemetry
Erfassen wichtiger Messdaten wie Fehlerrate und Latenz zwischen Diensten

Traffic Management
Verschieben von Datenverkehr zwischen verschiedenen Diensten



Quelle: https://smi-spec.io

DER SMI STANDARD

Traffic Metrics

Diese Teil-Spezifikation des SMI-Standards beschreibt einen gemeinsamen Integrationspunkt für Tools, die Metriken des HTTP-Verkehrs auswerten (bspw. HPA-Skalierung).

Metriken sind dabei immer einer Ressource zugeordnet. Dies können Pods sowie übergeordnete Konzepte wie Namespaces, Deployments oder Services sein. Metriken sind dabei entweder der Kubernetes-Ressource zugeordnet, die den gemessenen Datenverkehr generiert (request) oder bedient (response).

```
kind: TrafficMetrics
resource:
  name: foo-775b9cbd88-ntxsl
  namespace: foobar
  kind: Pod
edge:
  direction: from
  side: client
  resource:
   name: baz-577db7d977-lsk2q
   namespace: foobar
   kind: Pod
timestamp: 2019-04-08T22:25:55Z
window: 30s
metrics:
  - name: p99_response_latency
    unit: seconds
    value: 10m
  - name: p90_response_latency
    unit: seconds
    value: 10m
  - name: p50_response_latency
    unit: seconds
    value: 10m
  - name: success_count
    value: 100
  - name: failure_count
    value: 100
```

Derartige Metriken werden im laufenden Betrieb aus den Sidecar-Proxies erhoben.

Quelle: https://smi-spec.io

SERVICE MESH INTERFACE

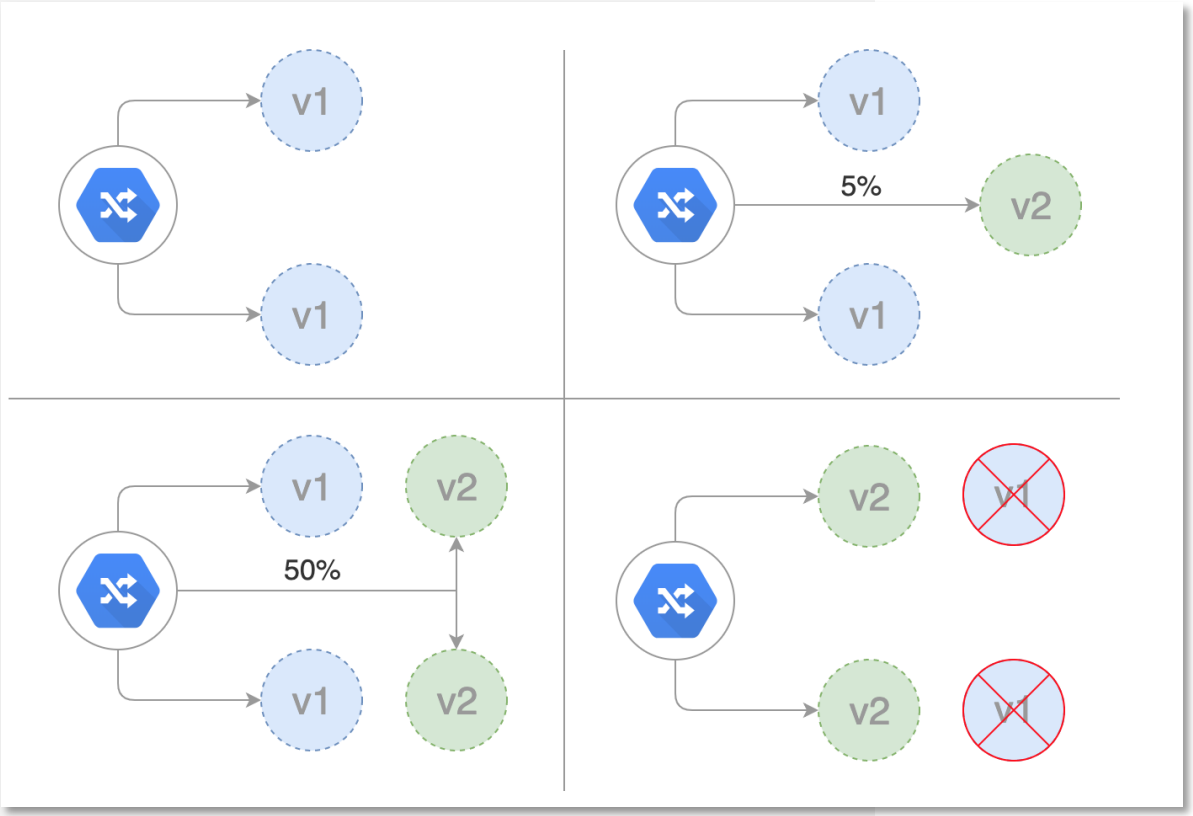
Der SMI Standard

CLOUD NATIVE COMPUTING

Traffic Policy
Anwenden von Richtlinien wie Identität und Transportverschlüsselung zwischen Diensten

Traffic Telemetry
Erfassen wichtiger Messdaten wie Fehlerrate und Latenz zwischen Diensten

Traffic Management
Verschieben von Datenverkehr zwischen verschiedenen Diensten

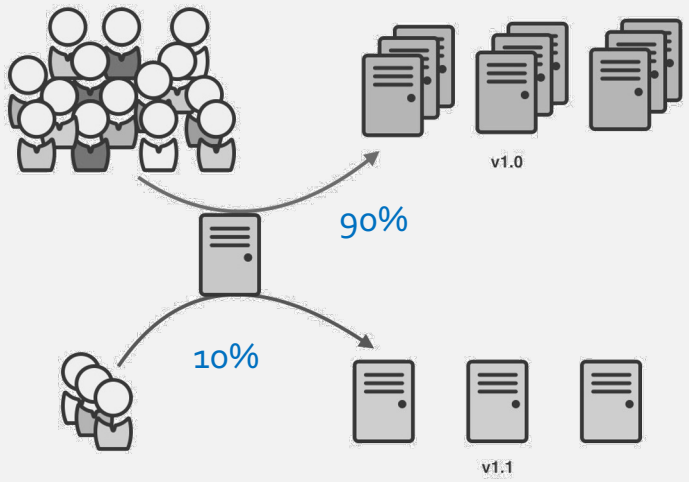


Quelle: https://smi-spec.io

DER SMI STANDARD

Traffic Management (Traffic Splits, Canary Releases)

- Mittels der Traffic Split-Ressource, kann der **prozentuale Anteil des Datenverkehrs** zwischen verschiedenen Services gesteuert werden
- Wird verwendet, um den ausgehenden **Datenverkehr** auf verschiedene Ziele **aufzuteilen**
- Oft genutzt, um **Canary Versionen** für neue Softwareversionen zu orchestrieren




```
kind: TrafficSplit
metadata:
  name: canary
spec:
  # The root service that clients use to connect
  service: website
  # Services inside the namespace
  backends:
  - service: website-v1.0
    weight: 90
  - service: website-v1.1
    weight: 10
```

Beispiel: Aufteilung des HTTP-Verkehrs zwischen zwei Versionen eines Dienstes. 90% des Traffics gehen an die Version v1.0. 10% des Traffics gehen an die Version v1.1 (Canary Release)

Quelle: https://smi-spec.io

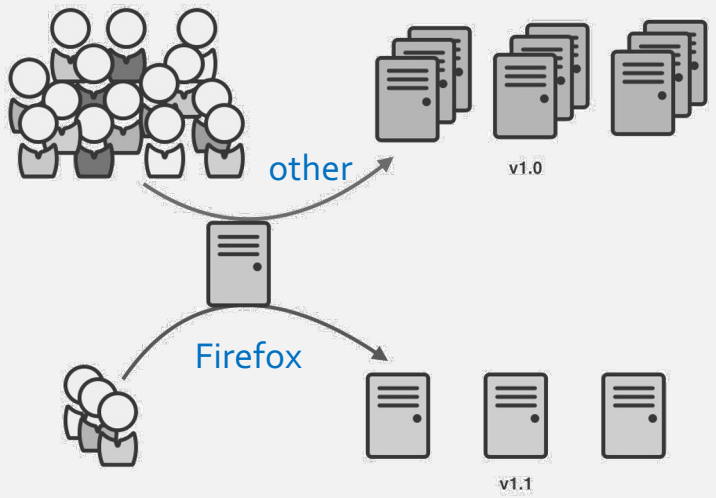
DER SMI STANDARD

Traffice Management (Traffic Split, A/B Testszenarien)

- Traffic Splits können auch genutzt werden, um **A/B-Testszenarien** umzusetzen
- Ein Traffic Split kann bspw. anhand von HTTP-Headerfilter vorgenommen werden
- **Bestimmter Benutzeranteil** wird dann an ein Test-Backend weitergeleitet
- Alle anderen Benutzer (**Kontrollgruppe**) werden weiter an das normale Backend weitergeleitet
- Hierzu können bspw. HTTP-Header-Filter mithilfe von HTTPRouteGroup definiert werden

```
kind: HTTPRouteGroup
metadata:
  name: ab-test
matches:
- name: firefox-users
  headers:
  - user-agent: ".*Firefox.*"
```

Route um Nutzer anhand ihres Browser (user-agent) selektieren zu können, hier werden alle Firefox Nutzer selektiert.



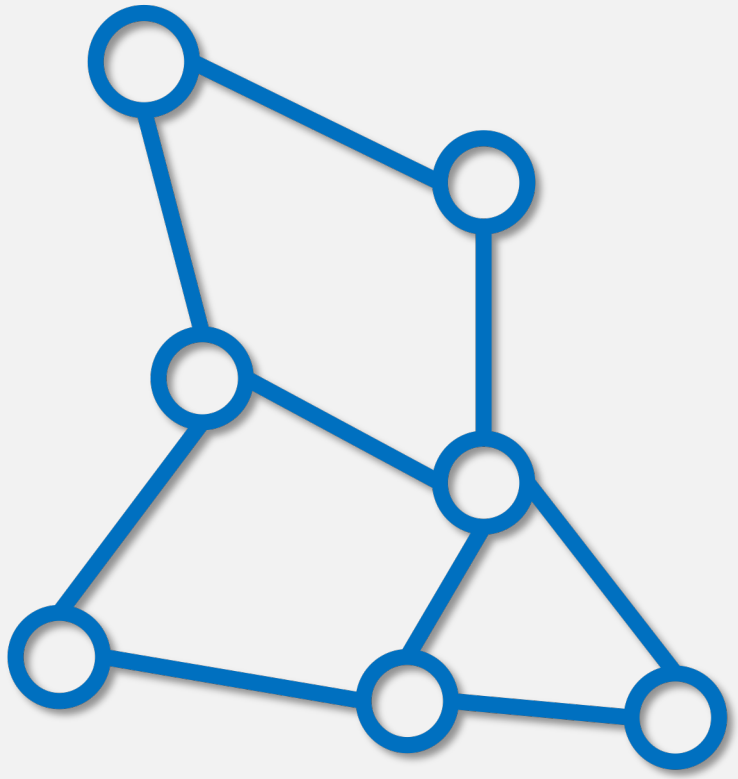

```
kind: TrafficSplit
metadata:
  name: ab-test
spec:
  service: website
  matches:
  - kind: HTTPRouteGroup
    name: ab-test
  backends:
  - service: website-v1.0
    weight: 0
  - service: website-v1.1
    weight: 100
```

Hier werden alle Firefox Nutzer auf die v1.1 des Website Services gerouted.

Quelle: https://smi-spec.io

AUSBLICK

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



CLOUD NATIVE COMPUTING

KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de