Istio in Practice - Day 1

Introductions

Each person say:

- Hello & where they are from
- How much k8s/Istio experience they have
- The main thing they want to learn
- Favourite something computer related
- Favourite something not computer related

Training

Learn and engage directly alongside our team, with courses for all stages of your Kubernetes journey

Workshops

Enhancing Kubernetes knowledge with our workshops. Whichever stage of your cloud native journey you are on we can help you do it better

Consulting

Consulting and engineering to make the most of Kubernetes and move you to production quickly

Subscription

Reference architecture, online training and SLA support 24x7 for your production Kubernetes deployment

Agenda

- Introduction
- Traffic Management
- Security
- Observability

What is a Service Mesh?

An architectural pattern that provides common network services as a feature of the infrastructure

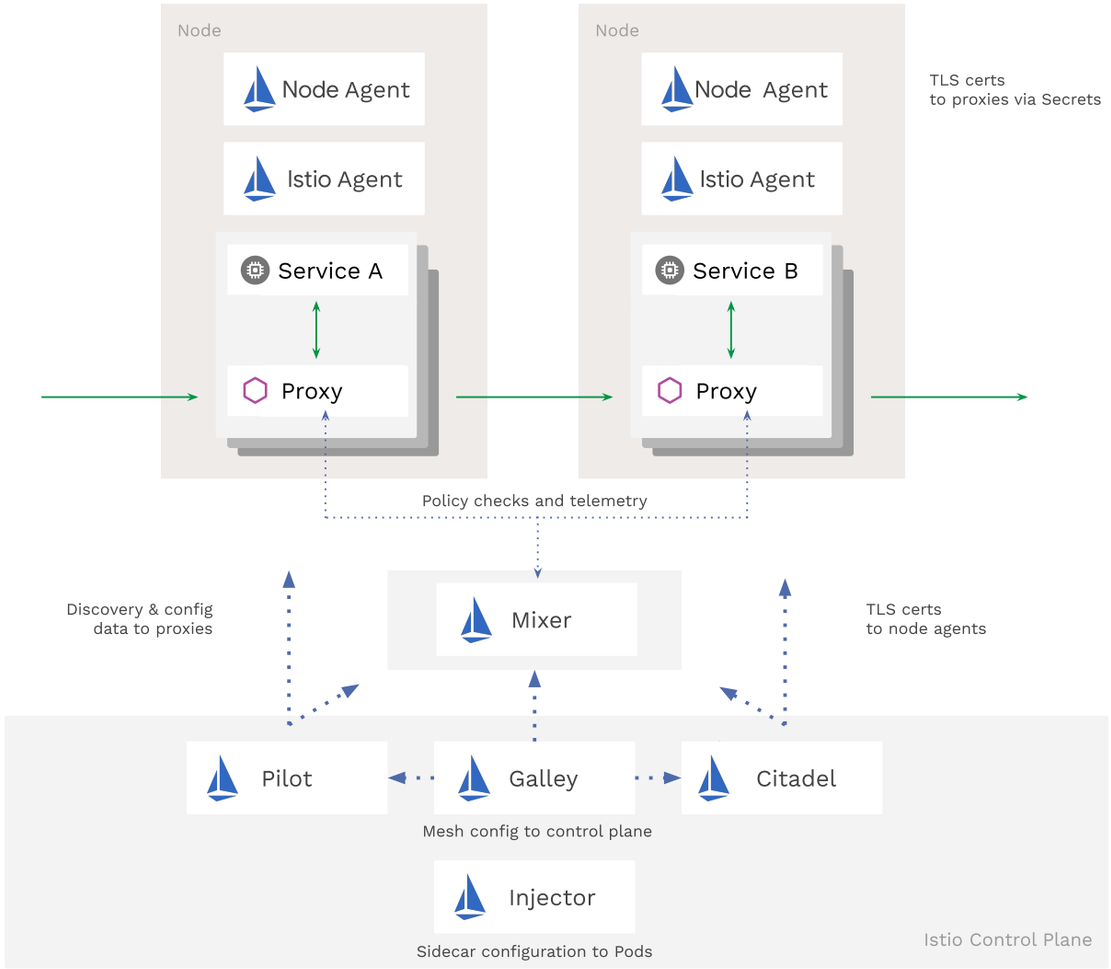
Why Do We Need a Service Mesh?

The scale and complexity of microservice architectures and distributed systems benefits greatly from a dedicated layer to orchestrate cross cutting concerns

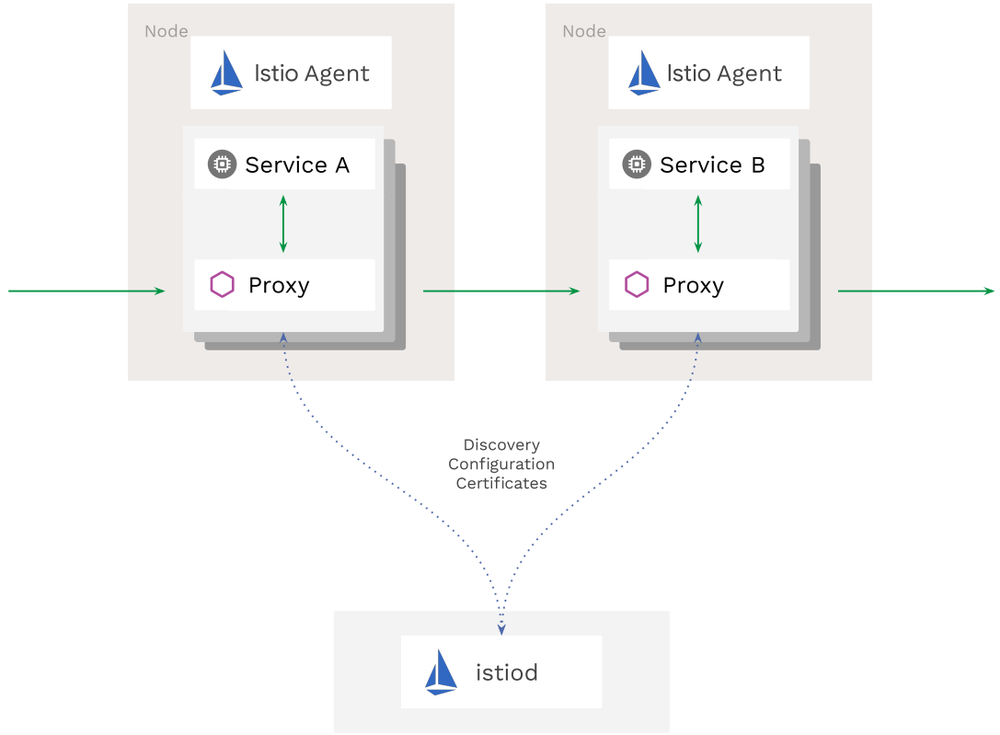
- Facilitates service-to-service communication between microservices
- Abstracts the network and allows operators to think in terms of services rather than specific IPs and ports
- Functionality includes discovery, load balancing, failure recovery, policy enforcement, monitoring and telemetry

Service Mesh Implementations

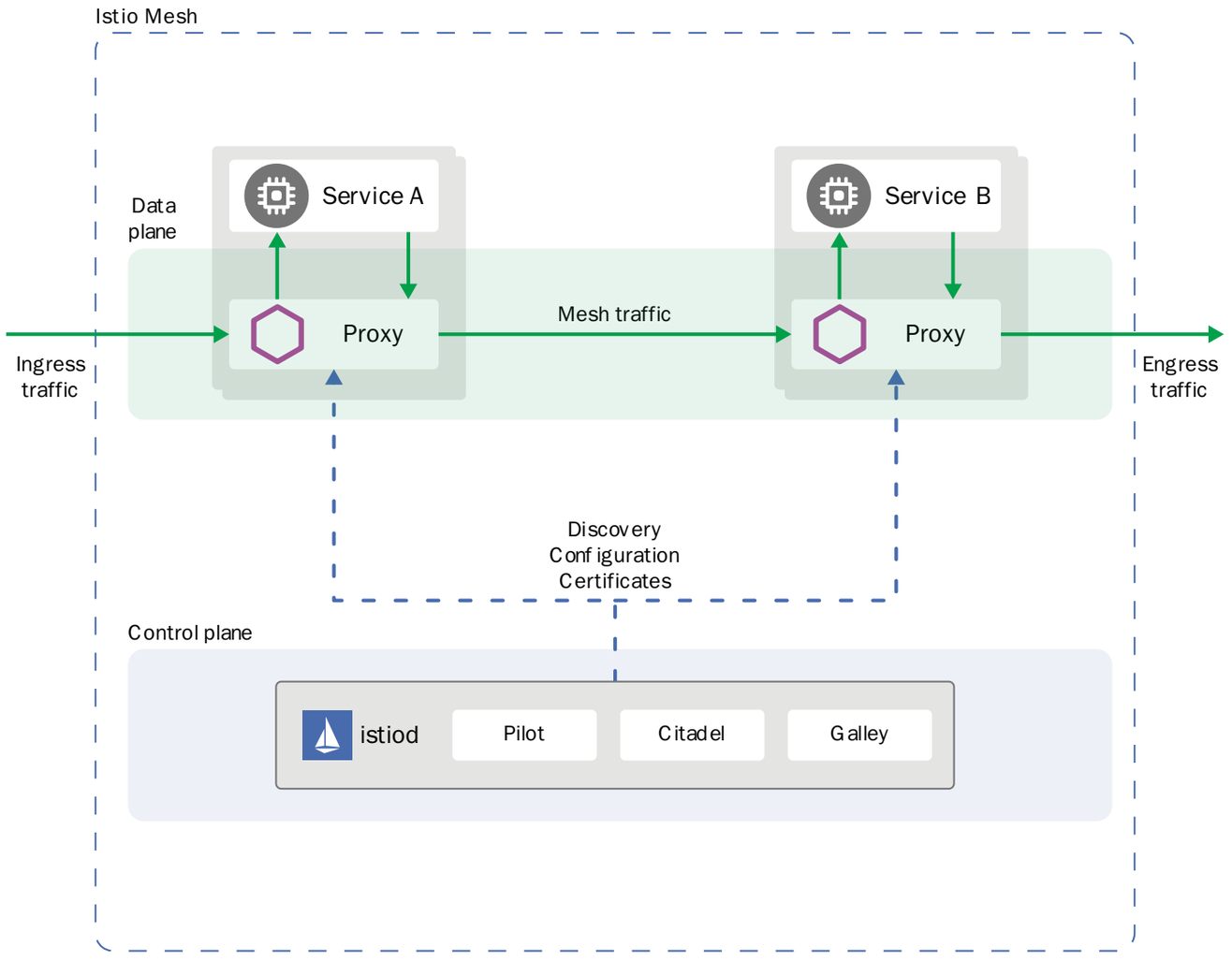
Istio Architecture v1.4.X



Istio Architecture v1.5.X+



Istio Architecture v1.5.X+



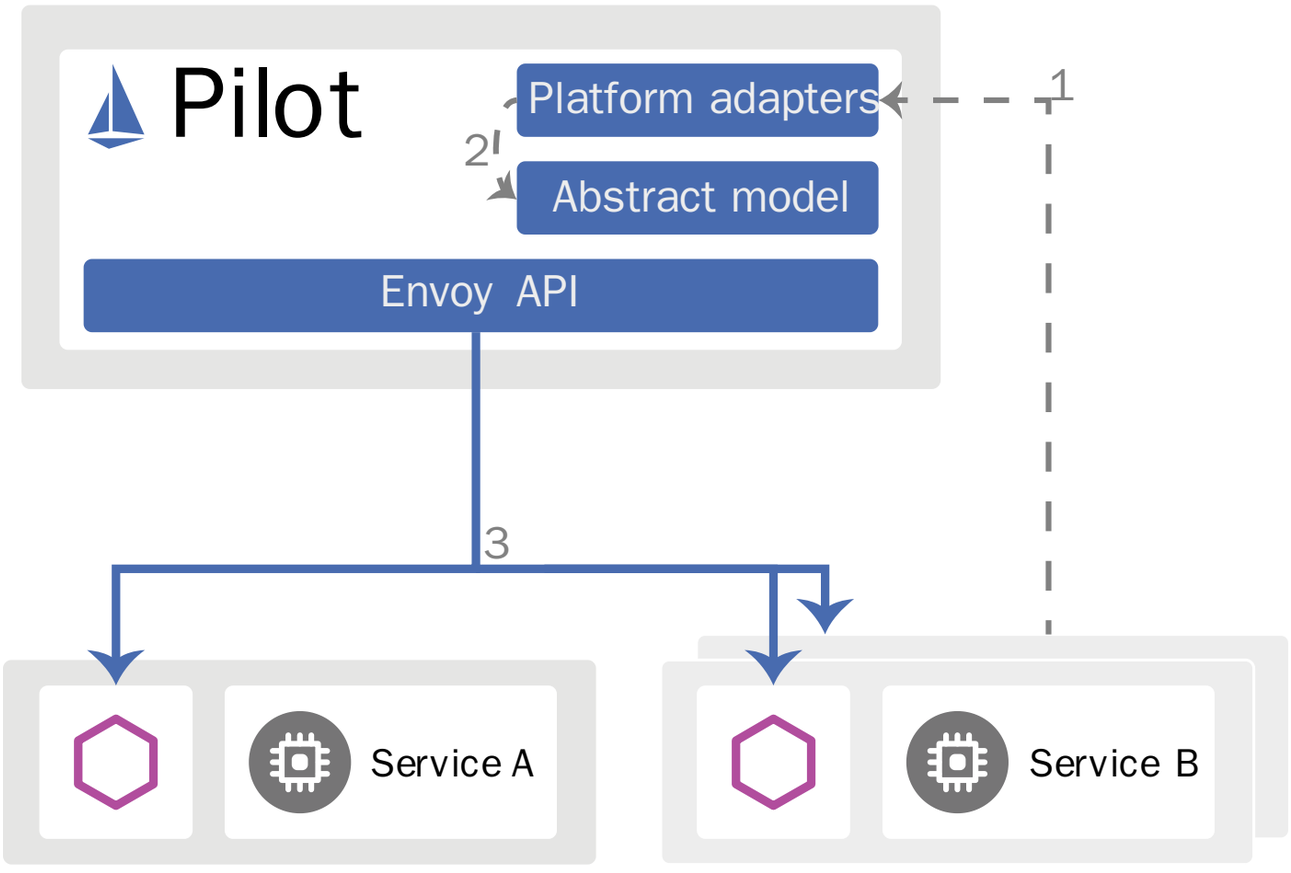
https://istio.io/docs/ops/deployment/architecture/arch.svg

Control Plane / `istiod`

- Pilot
- Citadel
- Galley
- Sidecar Injection

Pilot

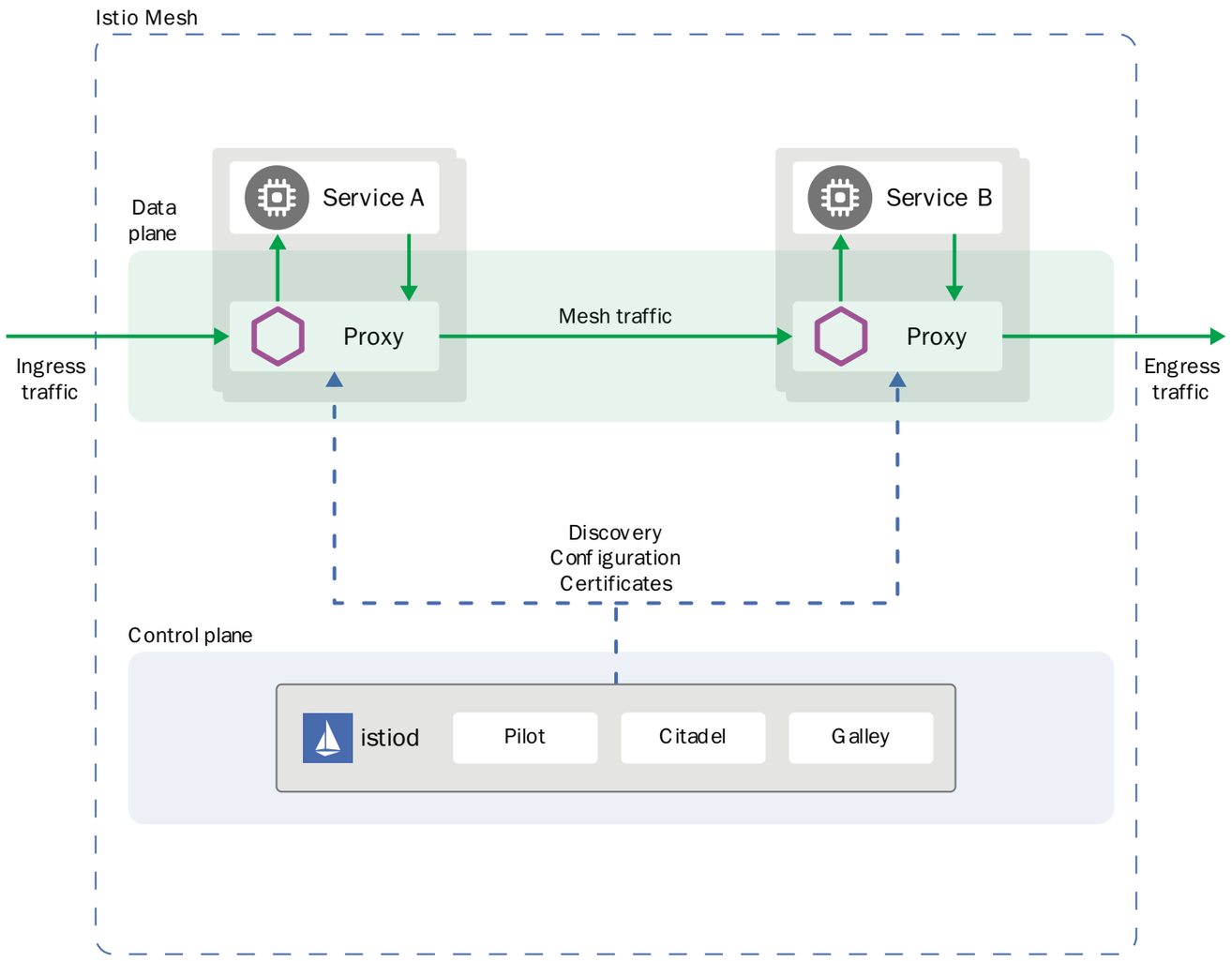
Serves xDS APIs to configure service discovery, traffic management capabilities and security for the Envoy sidecars



https://istio.io/latest/docs/ops/deployment/architecture/discovery.svg

Citadel

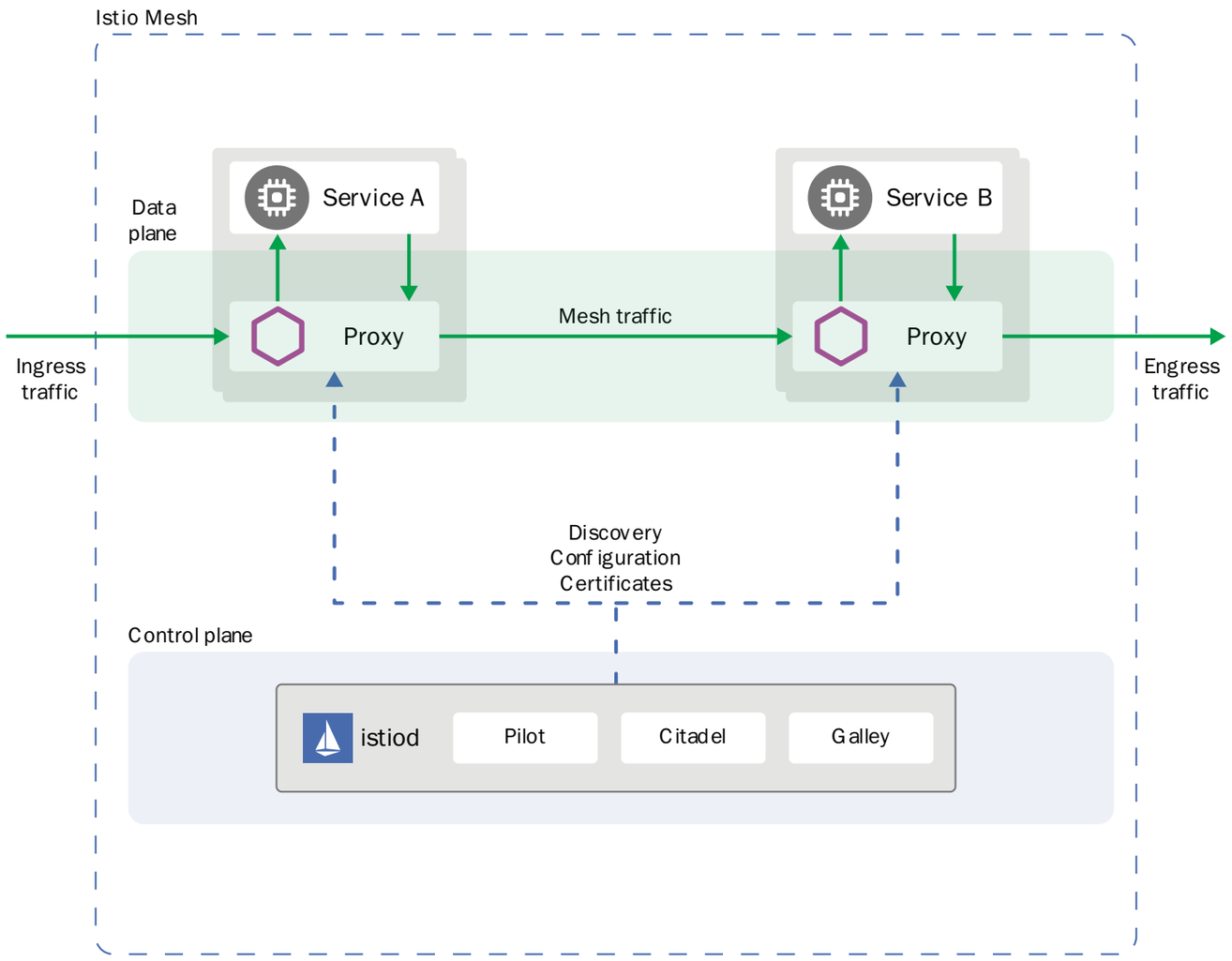
Enables strong service-to-service and end-user authentication with built-in identity and credential management



https://istio.io/latest/docs/ops/deployment/architecture/arch.svg

Galley

Converts platform specific CRDs into component configuration which can be consumed (e.g. by Pilot) over the Mesh Configuration Protocol



https://istio.io/latest/docs/ops/deployment/architecture/arch.svg

Sidecar Injection

Manual

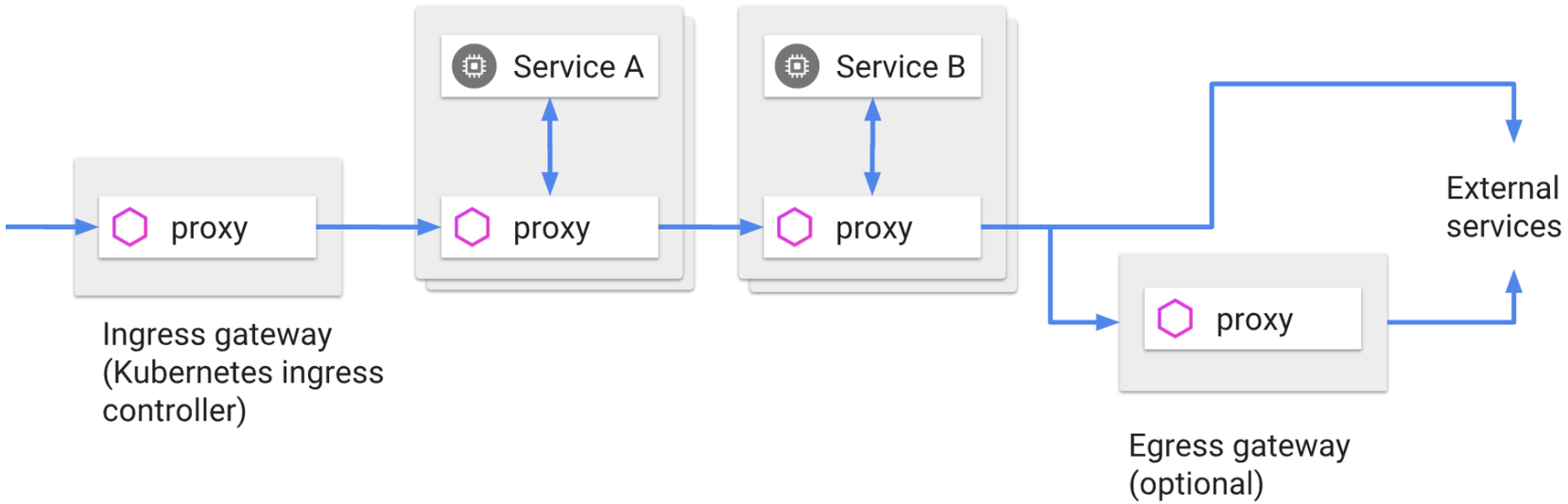
```
istioctl kube-inject -f deployment.yaml | \
    kubectl apply -f -
```

Automatic

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: istio-sidecar-injector
webhooks:
- rules:
  - apiGroups:
    - ""
    apiVersions:
    - v1
    operations:
    - CREATE
    resources:
    - pods
    scope: '*'
...
```

Data Plane

Envoy



Workshop: Deploy GKE Cluster and Install Istio

IstioOperator API

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: default
```

```
istioctl install -f istio-operator.yaml
```

Traffic Management

Istio's traffic routing rules are used to control the flow of traffic and API calls between services.

Istio simplifies configuration of service-level properties like circuit breakers, timeouts, and retries, and makes it easy to set up important tasks like A/B testing, canary rollouts, and staged rollouts with percentage-based traffic splits.

`VirtualServices` and `DestinationRules` are key resources for configuring Istio's traffic routing functionality.

VirtualService

A `VirtualService` is used to configure how requests are routed to a service.

```yaml
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
```

The `hosts` field

The `hosts` field lists the destination hosts to which traffic routing rules are applied.

```
hosts:
- reviews # short name expanded to FQDN
```

Routing rules

The `http` section contains an ordered list of route rules for HTTP traffic.

- `match` conditions and actions for routing HTTP/1.1, HTTP2, and gRPC traffic sent to the destination(s) specified in the hosts field.
- A routing rule consists of the destination of the traffic and zero or more match conditions, depending on the use case.

Match condition

In this case, the routing should apply to all requests from the user `jason`, so the `end-user` header, and `exact` field are used to select the appropriate requests.

```
- match:
   - headers:
       end-user:
         exact: jason
```

Destination

The `route` section's `destination` field specifies the actual destination for traffic that matches this condition.

```
route:
- destination:
    host: reviews
    subset: v2
```

DestinationRule

`VirtualServices` define how traffic is routed to a given destination.

`DestinationRules` define policies that apply to traffic intended for a service after routing has occurred.

A `DestinationRule` is applied after `VirtualService` routing rules are evaluated, so they apply to the traffic's "real" destination.

my-destination-rule.yaml

```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
```

Load balancing options

By default, Istio uses a round-robin load balancing policy but can also apply the following models:

- Random: Requests are forwarded at random to instances in the pool.
- Weighted: Requests are forwarded to instances in the pool according to a specific percentage.
- Least requests: Requests are forwarded to instances with the least number of requests.

Workshop - VirtualServices and DestinationRules

ServiceEntry

ServiceEntry

In order to direct traffic within your mesh, Istio needs to know where all your endpoints are and which services they belong to.

`ServiceEntry`

A `ServiceEntry` is used to add an entry into Istio's internal service registry.

After a `ServiceEntry` is added, the Envoy proxies can send traffic to the specified host as if it was a service in the mesh.

ServiceEntry

Configuring service entries allows traffic to be managed for services running outside of the mesh, including the following tasks:

- Redirect and forward traffic for external destinations, such as APIs consumed from the web, or traffic to services in legacy infrastructure.
- Define retry, timeout, and fault injection policies for external destinations.
- Logically add services from a different cluster to the mesh to configure a multicluster Istio mesh on Kubernetes.

The following example `MESH_EXTERNAL` service entry adds the `ext-svc` external dependency to Istio's service registry:

`svc-entry.yaml`

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: svc-entry
spec:
  hosts:
  - ext-svc.example.com      # FQDN of external resource or wildcard pref
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

Multi cluster services

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: httpbin-bar
spec:
  hosts:
  # must be of form name.namespace.global
  - httpbin.bar.global
  # Treat remote cluster services as part of the service mesh
  # as all clusters in the service mesh share the same root of trust.
  location: MESH_INTERNAL
ports:
- name: http1
  number: 8000
  protocol: http
resolution: DNS
addresses:
# the IP address to which httpbin.bar.global will resolve to
# must be unique for each remote service, within a given cluster.
# This address need not be routable. Traffic for this IP will bes
# captured by the sidecar and routed appropriately.
- 240.0.0.2
endpoints:
# This is the routable address of the ingress gateway in cluster2
# that sits in front of sleep.foo service. Traffic from the sidecar
# will be routed to this address.
- address: ${CLUSTER2_GW_ADDR}
  ports:
    http1: 15443
```

Gateway

Istio can control inbound and outbound traffic to the mesh by defining ingress and egress Gateways.

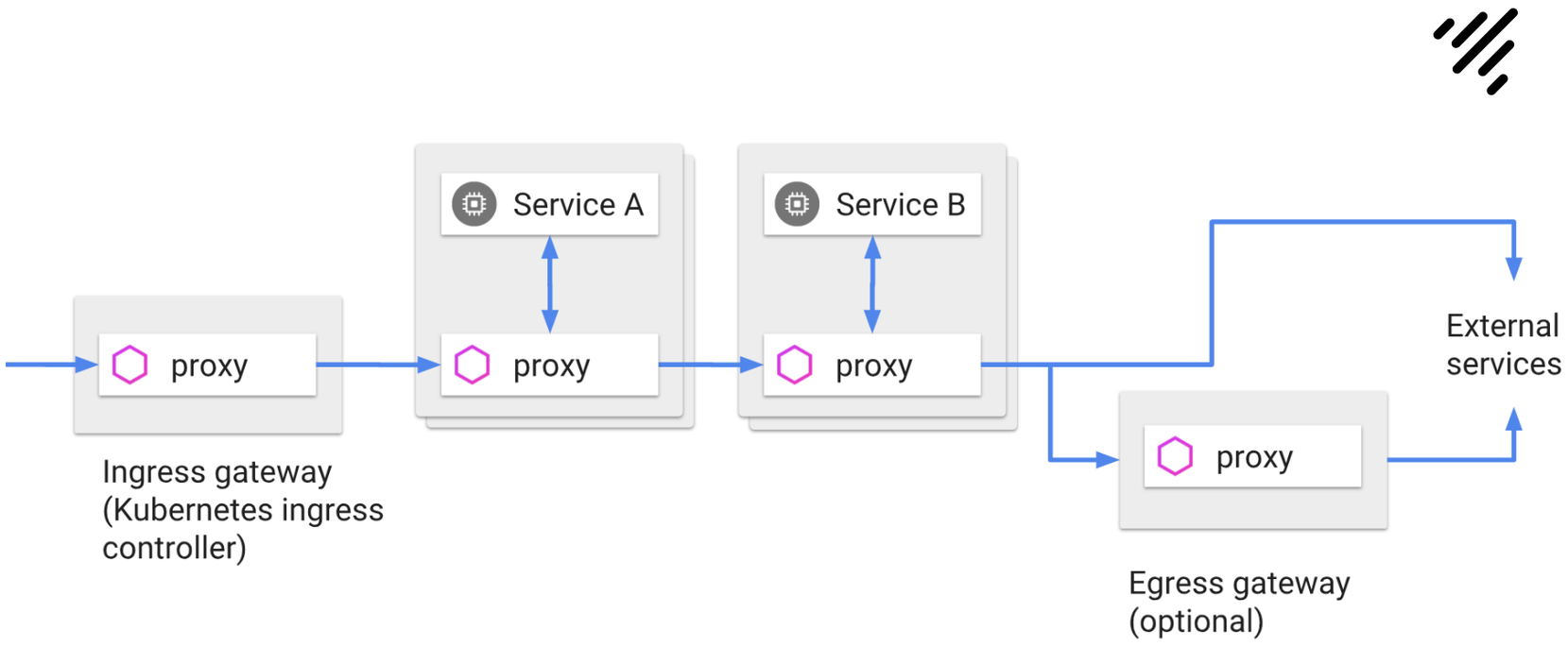
Gateway configurations are applied to standalone Envoy proxies that are running at the edge of the mesh, rather than sidecar Envoy proxies running alongside the service workloads.

Ingress Gateway

An ingress Gateway describes a load balancer operating at the edge of the mesh that receives incoming HTTP/TCP connections.

Egress Gateway

An egress gateway configures a dedicated exit node for the traffic leaving the mesh, limiting which services can or should access external networks, or to enable secure control of egress traffic to add security to the mesh.



https://blogs.sap.com/wp-content/uploads/2019/02/Screenshot-2019-02-08-at-16.02.10.png

httpbin-gateway.yaml

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: httpbin-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "httpbin.example.com"
```

Ingress and Egress Gateway installation

```
istioctl manifest install -f <(cat <<EOF
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  profile: default
  components:
    ingressGateways:
    - enabled: true # default is true for default profile
    egressGateways:
    - enabled: true
EOF
)
```

The `Gateway` must be bound to a `VirtualService` and specify routing as show below:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: httpbin
spec:
  hosts:
  - "httpbin.example.com"
  gateways:
  - httpbin-gateway
  http:
- match:
  - uri:
      prefix: /status
  - uri:
      prefix: /delay
  route:
  - destination:
      port:
        number: 8000
      host: httpbin
```

Workshop - ServiceEntry and Gateway

Security

Introduction

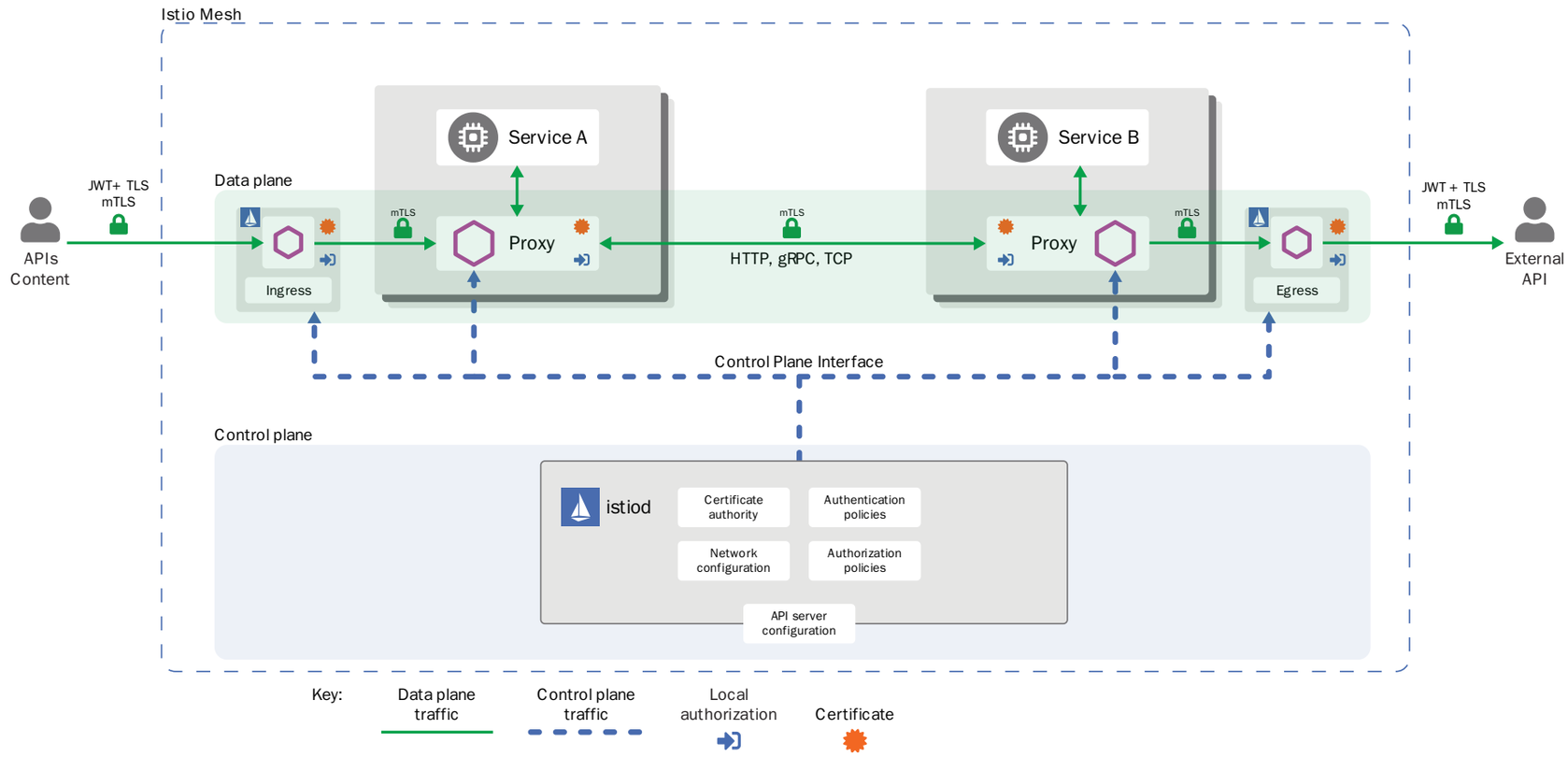
Istio implements a comprehensive array of security features to address common concerns:

- To defend against man-in-the-middle attacks, we need traffic encryption.
- To provide flexible service access control, we need mutual TLS and fine-grained access policies.
- To audit who did what at what time, we need auditing tools.

The goals of Istio security are:

- Security by default: no changes needed for application code and infrastructure
- Defence in depth: integrate with existing security systems to provide multiple layers of defence
- Zero-trust network: build security solutions on untrusted networks

High-level architecture



https://istio.io/docs/concepts/security/arch-sec.svg

Istio identity

Identity is a fundamental concept of any security infrastructure.

In the Istio identity model, Istio uses the first-class service identity to determine the identity of a service.

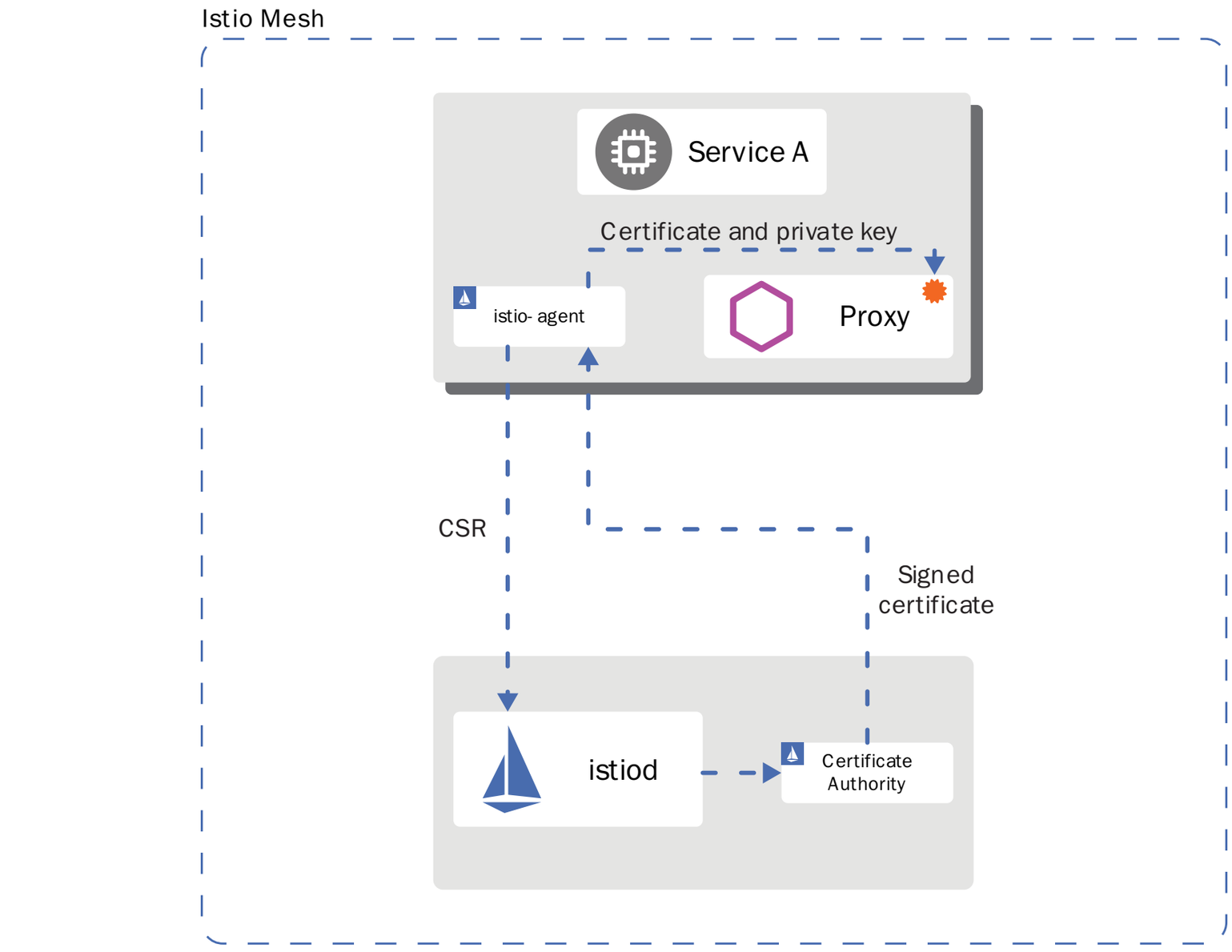
This gives great flexibility and granularity to represent a human user, an individual service, or a group of services.

On platforms that do not have such identity available, Istio can use other identities that can group service instances, such as service names.

PKI

The Istio PKI securely provisions strong identities to every workload with X.509 certificates.

To automate key and certificate rotation at scale, the PKI runs an Istio agent alongside each Envoy proxy for certificate and key provisioning.



https://istio.io/docs/concepts/security/id-prov.svg

Mutual TLS

Istio tunnels service-to-service communication through the client side and server side Envoy proxies.

For a client to call a server with mutual TLS authentication:

- Istio re-routes the outbound traffic from a client to the client's local sidecar Envoy.
- The client side Envoy starts a mutual TLS handshake with the server side Envoy.
- The client side Envoy and the server side Envoy establish a mutual TLS connection, and Istio forwards the traffic from the client side Envoy to the server side Envoy.
- After authorisation, the server side Envoy forwards the traffic to the server service through local TCP connections.

Permissive mode

Istio mutual TLS has a permissive mode, which allows a service to accept both plaintext traffic and mutual TLS traffic at the same time.

The server's installed Istio sidecar takes mutual TLS traffic immediately without breaking existing plaintext traffic.

Secure naming

In Istio, each workload is automatically assigned with an identity.

Server identities are encoded in certificates, but service names are retrieved through the discovery service or DNS.

The secure naming information maps the server identities to the service names.

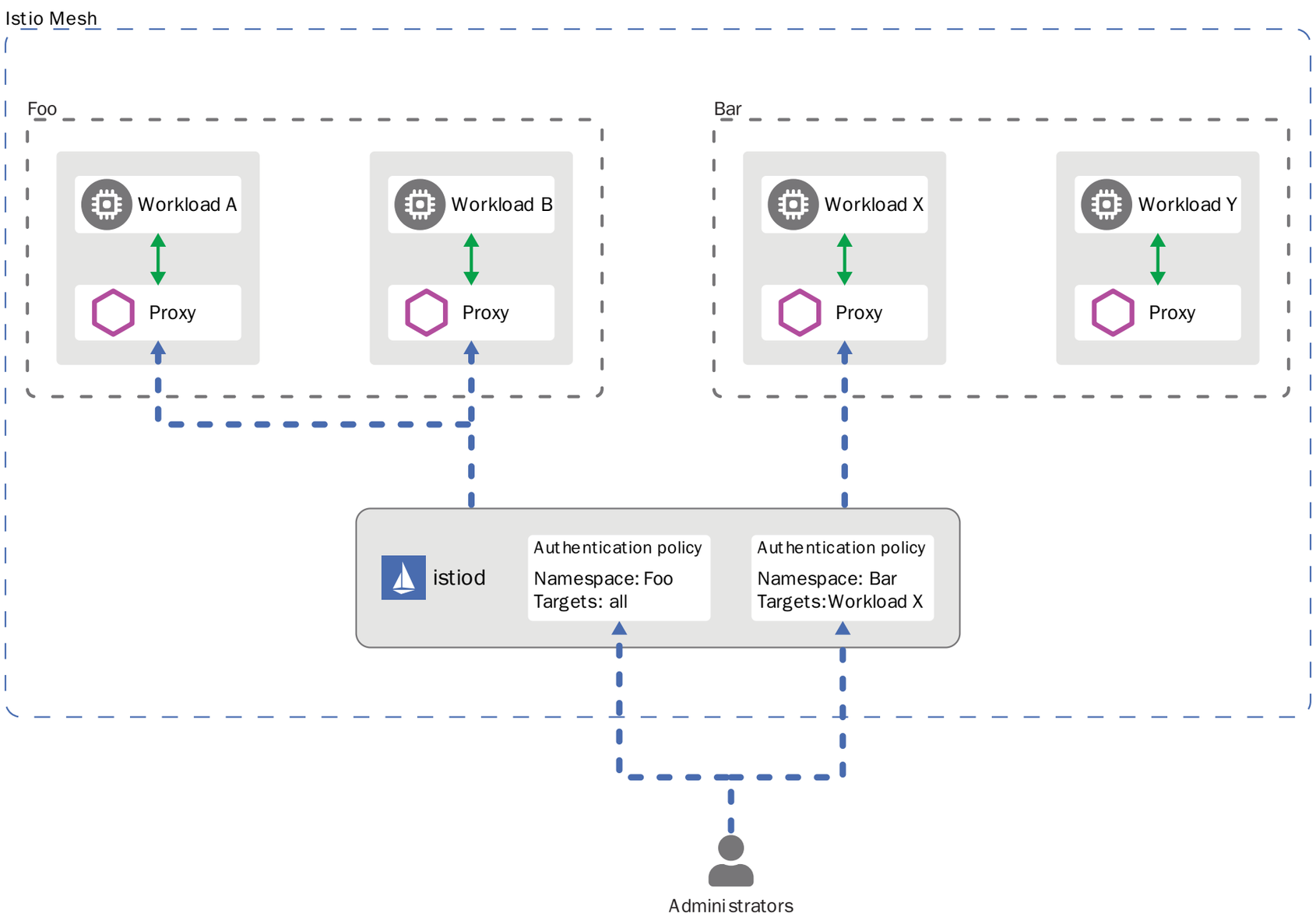
A mapping of identity `A` to service name `B` means "`A` is authorised to run service `B`".

The control plane watches the apiserver, generates the secure naming mappings, and distributes them securely to the Envoy proxies.

Secure naming offers the following benefits:

- Decouples the identity system from the service naming system.
- The certificate offers the identity of the server, instead of the service name the server is running for.
- It works well with the Kubernetes resource model.

Authentication architecture



https://istio.io/docs/concepts/security/authn.svg

Authentication

Istio provides two types of authentication:

- Peer authentication
- Request authentication

`PeerAuthentication` and `RequestAuthentication` policies use `selector` fields to specify the label of the workloads to which the policy applies.

The following snippet shows the selector field of a policy that applies to workloads with the `app:product-page` label:

```
selector:
   matchLabels:
     app: product-page
```

If you don't provide a value for the `selector` field, Istio matches the policy to all workloads in the storage scope of the policy.

Therefore, the `selector` fields help you specify the scope of the policies:

- Mesh-wide policy: A policy specified for the root namespace without or with an empty selector field.
- Namespace-wide policy: A policy specified for a non-root namespace without or with an empty selector field.
- Workload-specific policy: a policy defined in the regular namespace, with non-empty selector field.

Peer authentication

Used for service-to-service authentication to verify the client making the connection.

Istio offers mutual TLS as a full stack solution for transport authentication.

Authentication policies apply to requests that a service receives.

The following peer authentication policy specifies that transport authentication for the `reviews` service must use mutual TLS:

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

Peer authentication

Peer authentication policies specify the mutual TLS mode Istio enforces on target workloads. The following modes are supported:

- `PERMISSIVE`: Workloads accept both mutual TLS and plain text traffic. This mode is most useful during migrations when workloads without sidecar cannot use mutual TLS. Once workloads are migrated with sidecar injection, you should switch the mode to STRICT.
- `STRICT`: Workloads only accept mutual TLS traffic.
- `DISABLE`: Mutual TLS is disabled. From a security perspective, you shouldn't use this mode unless you provide your own security solution.

Peer authentication

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

Request authentication

Used for end-user authentication to verify the credential attached to the request.

The following request authentication policy requires an end-user JWT for the ingress gateway:

```
apiVersion: "security.istio.io/v1beta1"
kind: "RequestAuthentication"
metadata:
  name: "jwt-example"
  namespace: istio-system
spec:
  selector:
    matchLabels:
      istio: ingressgateway
  jwtRules:
  - issuer: "testing@secure.istio.io"
    jwksUri: "https://raw.githubusercontent.com/istio/istio/release-1.5/
```

Workshop - Authentication

Workshop - OIDC Authentication

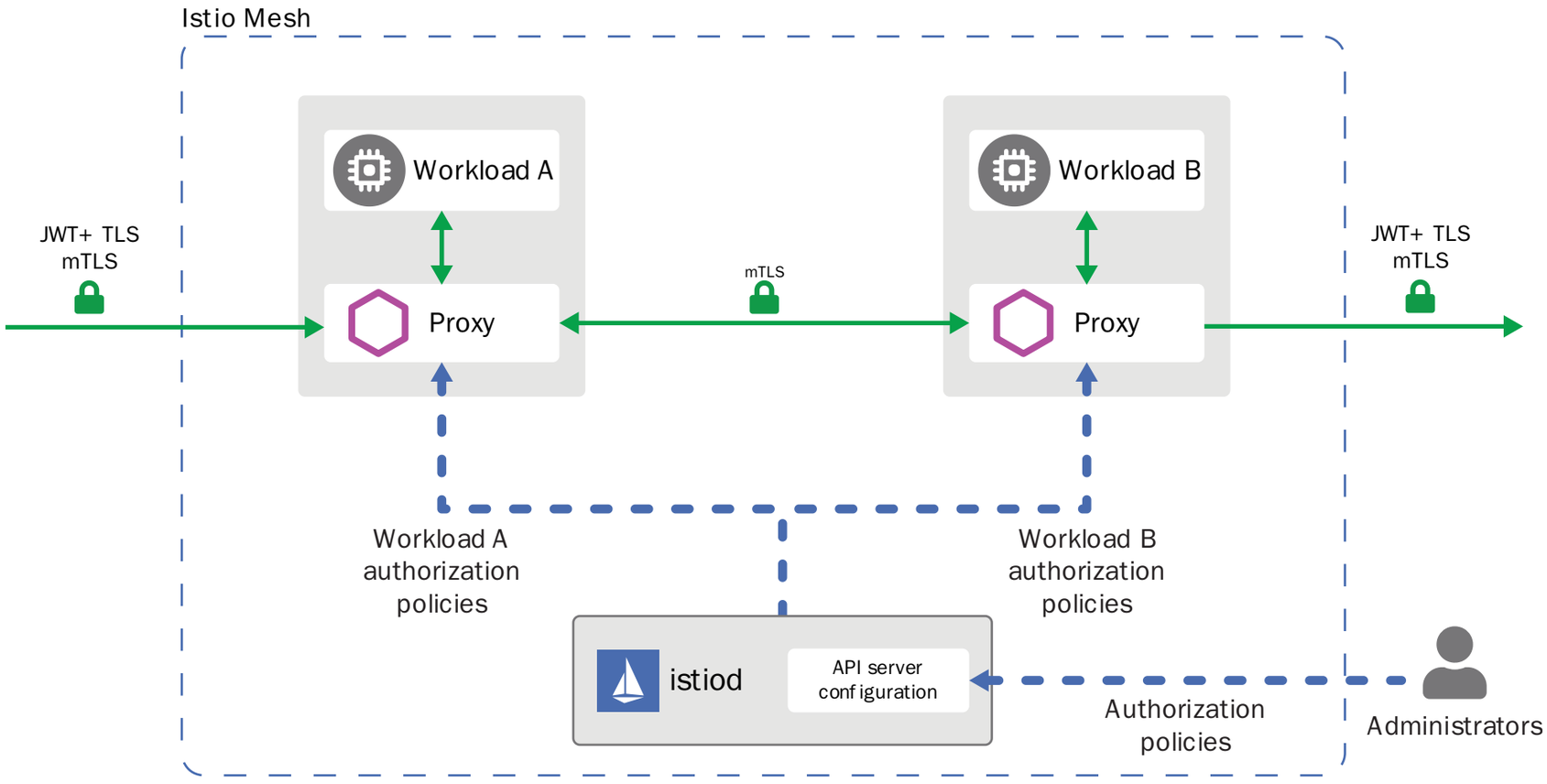
Authorisation

Istio's authorisation features provides mesh-level, namespace-level, and workload-level access control on workloads in an Istio Mesh.

This includes:

- Workload-to-workload and end-user-to-workload authorisation.
- A single `AuthorizationPolicy` CRD.
- Flexible semantics, operators can define custom conditions on Istio attributes.
- High performance, as Istio authorisation is enforced natively on Envoy.
- High compatibility, supports HTTP, HTTPS and HTTP2 natively, as well as any plain TCP protocols.

Authorisation architecture



https://istio.io/docs/concepts/security/authz.svg

There is no need to explicitly enable Istio's authorisation feature, only the `AuthorizationPolicy` needs applying on workloads to enforce access control.

If no `AuthorizationPolicy` applies to a workload, no access control will be enforced, In other words, all requests will be allowed.

If any `AuthorizationPolicy` applies to a workload, access to that workload is denied by default, unless explicitly allowed by a rule declared in the policy.

`AuthorizationPolicy`

To configure an Istio authorization policy, an `AuthorizationPolicy` resource is created.

An authorization policy includes a selector, an action, and a list of rules.

- The `selector` specifies the target that the policy
- The `action` field specifies whether to allow or deny the request
- The `rules` specify when to trigger the action
- The `from` field in the `rules` specifies the sources of the request
- The `to` field in the `rules` specifies the operations of the request
- The `when` field specifies the conditions needed to apply the rule

Allow-all and deny-all

The example below shows an allow-all policy which allows full access to all workloads in the default namespace.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: allow-all
  namespace: default
spec:
  rules:
  - {}
```

The example below shows a deny-all policy which denies access to all workloads in the admin namespace.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: admin
spec:
  {}
```

Authenticated and unauthenticated identity

To make workloads publicly accessible, the `source` section is left empty.

This allows sources from all (both authenticated and unauthenticated) users and workloads.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin
 namespace: foo
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 rules:
 - to:
   - operation:
       methods: ["GET", "POST"]
```

To allow only authenticated users, set `principals` to `"*"` instead, for example:

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin
 namespace: foo
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
rules:
- from:
  - source:
      principals: ["*"]
  to:
  - operation:
      methods: ["GET", "POST"]
```

Policy Target

Policy scope (target) is determined by metadata/namespace and an optional selector.

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

Full example

```yaml
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
 name: httpbin
 namespace: foo
spec:
 selector:
   matchLabels:
     app: httpbin
     version: v1
 rules:
 - from:
   - source:
       principals: ["cluster.local/ns/default/sa/sleep"]
   - source:
       namespaces: ["dev"]
   to:
   - operation:
       methods: ["GET"]
       paths: ["/info*"]
   when:
    - key: request.auth.claims[groups]
      values: ["group1"]
```

Workshop - Istio authorisation

Observability

Istio generates detailed telemetry for all service communications within a mesh.

Through Istio, operators gain a thorough understanding of how monitored services are interacting, both with other services and with the Istio components themselves.

Istio generates the following types of telemetry in order to provide overall service mesh observability:

- *Metrics* - Istio generates a set of service metrics based on latency, traffic, errors, and saturation as well as detailed metrics for the mesh control plane.

Istio generates the following types of telemetry in order to provide overall service mesh observability:

- *Distributed Traces* - Istio generates distributed trace spans for each service.

Istio generates the following types of telemetry in order to provide overall service mesh observability:

- *Access Logs* - As traffic flows into a service within a mesh, Istio can generate a full record of each request, including source and destination metadata.

Metrics

Metrics provide a way of monitoring and understanding behaviour across the mesh, generating metrics for all service traffic in, out, and within an Istio service mesh.

In addition to monitoring the behaviour of services within a mesh, it is also important to monitor the behaviour of the mesh itself.

Istio components export metrics on their own internal behaviours to provide insight on the health and function of the mesh control plane.

Proxy-level metrics

Istio metrics collection begins with the sidecar proxies (Envoy).

Each proxy generates a rich set of metrics about all traffic passing through the proxy (both inbound and outbound).

As a result, understanding the connection between mesh services and Envoy resources is required for monitoring the Envoy metrics.

Istio enables operators to select which of the Envoy metrics are generated and collected at each workload instance.

This enables targeted debugging of networking behaviour, while reducing the overall cost of monitoring across the mesh.

Example proxy-level Metrics:

```
envoy_cluster_internal_upstream_rq{response_code_class="2xx",cluster_nam

envoy_cluster_upstream_rq_completed{cluster_name="xds-grpc"} 7164

envoy_cluster_ssl_connection_error{cluster_name="xds-grpc"} 0

envoy_cluster_lb_subsets_removed{cluster_name="xds-grpc"} 0

envoy_cluster_internal_upstream_rq{response_code="503",cluster_name="xds
```

Service-level metrics

In addition to the proxy-level metrics, Istio provides a set of service-oriented metrics for monitoring service communications.

These metrics cover the four basic service monitoring needs: latency, traffic, errors, and saturation.

Example service-level metric:

```
istio_requests_total{
  connection_security_policy="mutual_tls",
  destination_app="details",
  destination_principal="cluster.local/ns/default/sa/default",
  destination_service="details.default.svc.cluster.local",
  destination_service_name="details",
  destination_service_namespace="default",
  destination_version="v1",
  destination_workload="details-v1",
  destination_workload_namespace="default",
  reporter="destination",
  request_protocol="http",
  response_code="200",
  response_flags="-",
  source_app="productpage",
  source_principal="cluster.local/ns/default/sa/default",
  source_version="v1",
  source_workload="productpage-v1",
  source_workload_namespace="default"
} 214
```

```
istioctl dashboard prometheus
```

Prometheus Alerts Graph Status Help

Enable query history

istio_requests_total

Load time: 14ms
Resolution: 14s
Total time series: 15

Execute - insert metric at cursor -

Graph Console

Element

istio_requests_total{connection_mtls="false",destination_app="details",destination_namespace="default",destination_principal="unknown",destination_service="details.default.svc.cluster.local",destination_service_name="details",destination_service_namespace="default",destination_
v1",destination_workload_namespace="default",instance="172.17.0.18:42422",job="istio-mesh",reporter="client",request_protocol="http",response_code="200",source_app="productpage",source_namespace="default",source_principal="unknown",source_version="v1",source_workl

istio_requests_total{connection_mtls="false",destination_app="details",destination_namespace="default",destination_principal="unknown",destination_service="details.default.svc.cluster.local",destination_service_name="details",destination_service_namespace="default",destination_
v1",destination_workload_namespace="default",instance="172.17.0.18:42422",job="istio-mesh",reporter="server",request_protocol="http",response_code="200",source_app="productpage",source_namespace="default",source_principal="unknown",source_version="v1",source_work

istio_requests_total{connection_mtls="false",destination_app="policy",destination_namespace="istio-system",destination_principal="unknown",destination_service="istio-policy.istio-system.svc.cluster.local",destination_service_name="istio-policy",destination_service_namespace="ist
system",destination_version="unknown",destination_workload="istio-policy",destination_workload_namespace="istio-system",instance="172.17.0.18:42422",job="istio-
mesh",reporter="server",request_protocol="grpc",response_code="200",source_app="details",source_namespace="default",source_principal="unknown",source_version="v1",source_workload="details-v1",source_workload_namespace="default"}

istio_requests_total{connection_mtls="false",destination_app="policy",destination_namespace="istio-system",destination_principal="unknown",destination_service="istio-policy.istio-system.svc.cluster.local",destination_service_name="istio-policy",destination_service_namespace="ist
system",destination_version="unknown",destination_workload="istio-policy",destination_workload_namespace="istio-system",instance="172.17.0.18:42422",job="istio-
mesh",reporter="server",request_protocol="grpc",response_code="200",source_app="productpage",source_namespace="default",source_principal="unknown",source_version="v1",source_workload="productpage-v1",source_workload_namespace="default"}

istio_requests_total{connection_mtls="false",destination_app="policy",destination_namespace="istio-system",destination_principal="unknown",destination_service="istio-policy.istio-system.svc.cluster.local",destination_service_name="istio-policy",destination_service_namespace="ist
system",destination_version="unknown",destination_workload="istio-policy",destination_workload_namespace="istio-system",instance="172.17.0.18:42422",job="istio-
mesh",reporter="server",request_protocol="grpc",response_code="200",source_app="reviews",source_namespace="default",source_principal="unknown",source_version="v1",source_workload="reviews-v1",source_workload_namespace="default"}

istio_requests_total{connection_mtls="false",destination_app="policy",destination_namespace="istio-system",destination_principal="unknown",destination_service="istio-policy.istio-system.svc.cluster.local",destination_service_name="istio-policy",destination_service_namespace="ist
system",destination_version="unknown",destination_workload="istio-policy",destination_workload_namespace="istio-system",instance="172.17.0.18:42422",job="istio-mesh",reporter="server",request_protocol="grpc",response_code="200",source_app="unknown",source_namespac
system",source_principal="unknown",source_version="unknown",source_workload="istio-ingressgateway",source_workload_namespace="istio-system"}

istio_requests_total{connection_mtls="false",destination_app="productpage",destination_namespace="default",destination_principal="unknown",destination_service="productpage.default.svc.cluster.local",destination_service_name="productpage",destination_service_namespace="d
v1",destination_workload_namespace="default",instance="172.17.0.18:42422",job="istio-mesh",reporter="client",request_protocol="http",response_code="200",source_app="unknown",source_namespace="istio-system",source_principal="unknown",source_version="unknown",sourc
ingressgateway",source_workload_namespace="istio-system"}

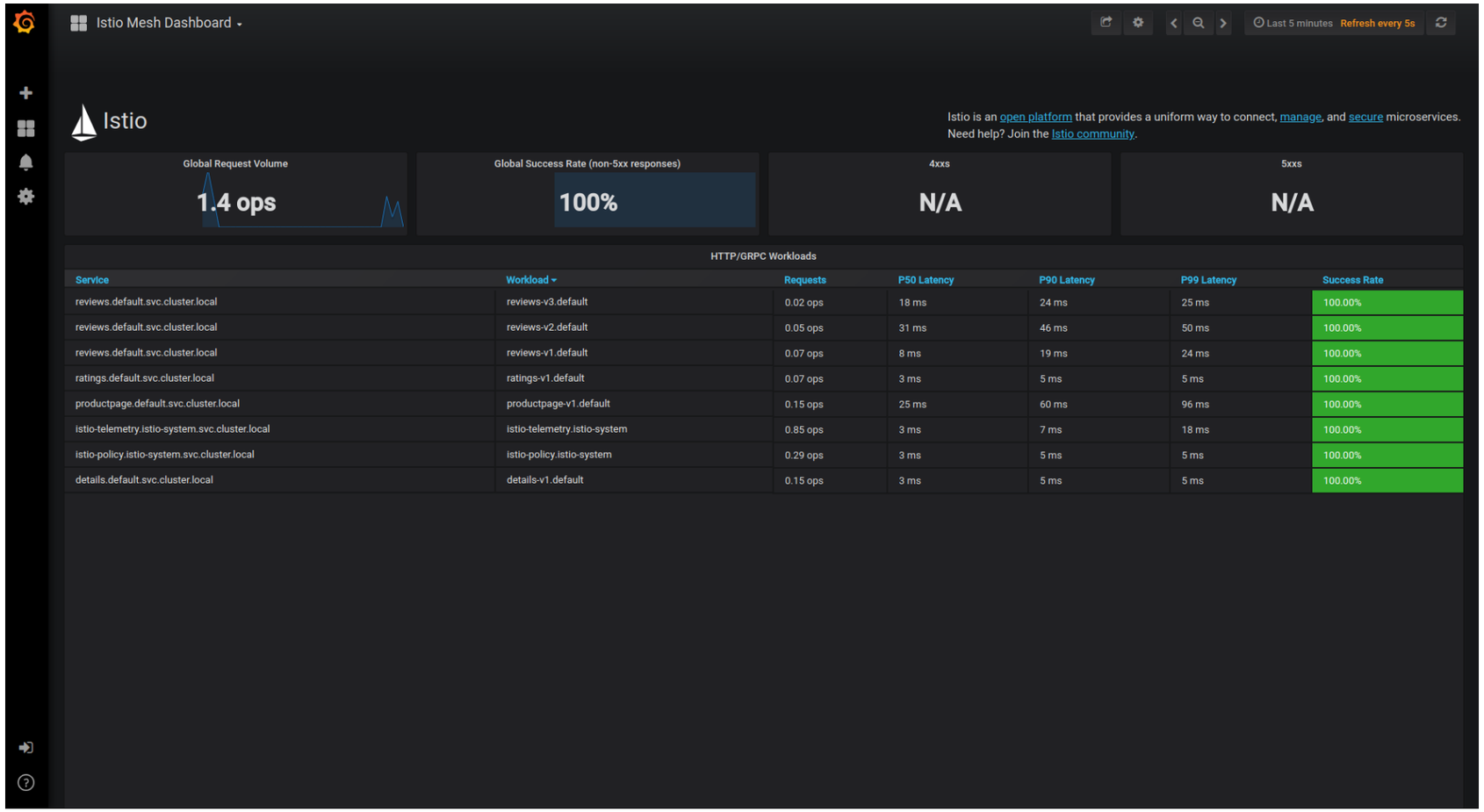
Control plane metrics

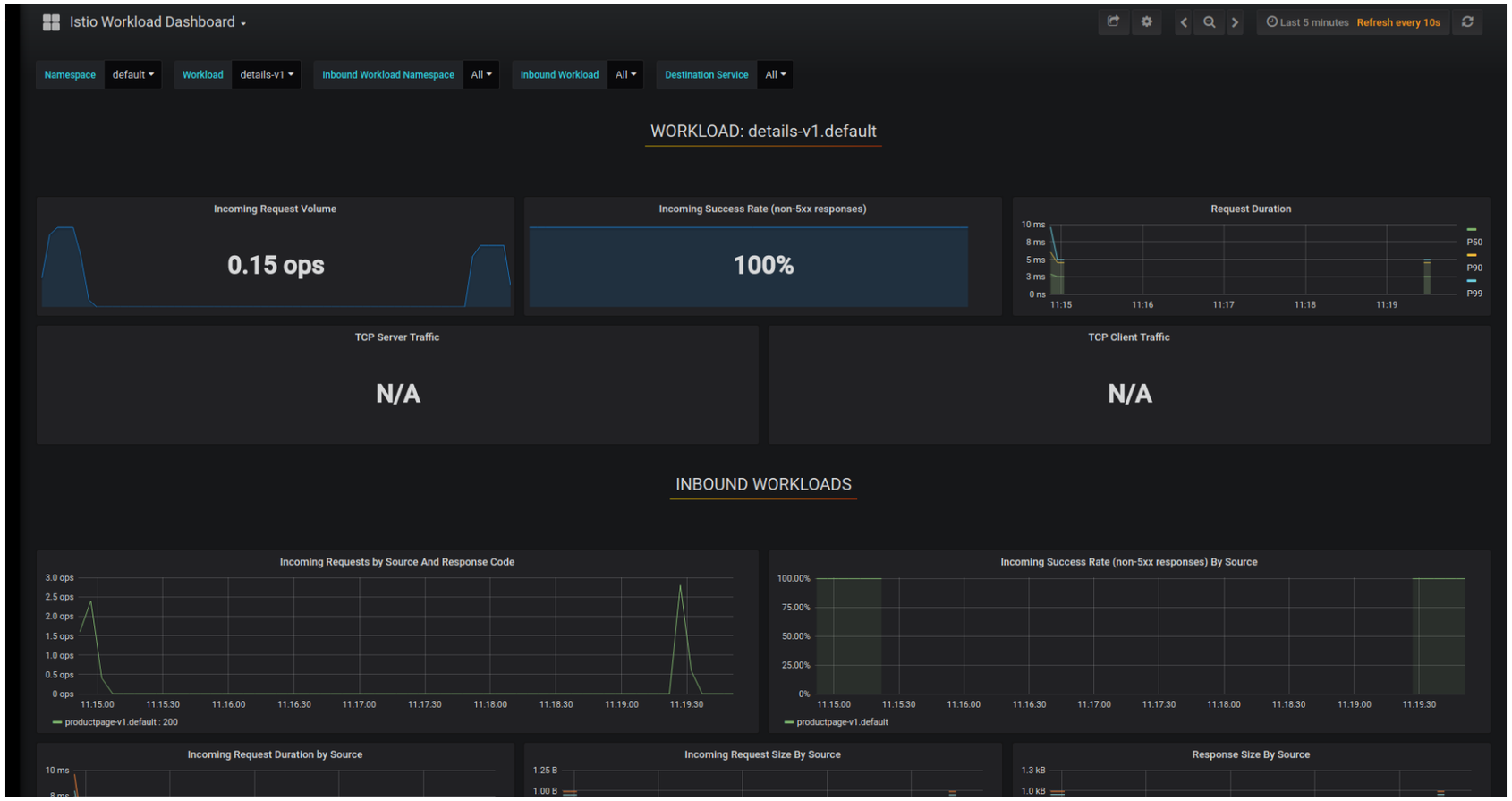
Each Istio component (Pilot, Galley, Mixer) also provides a collection of self-monitoring metrics.

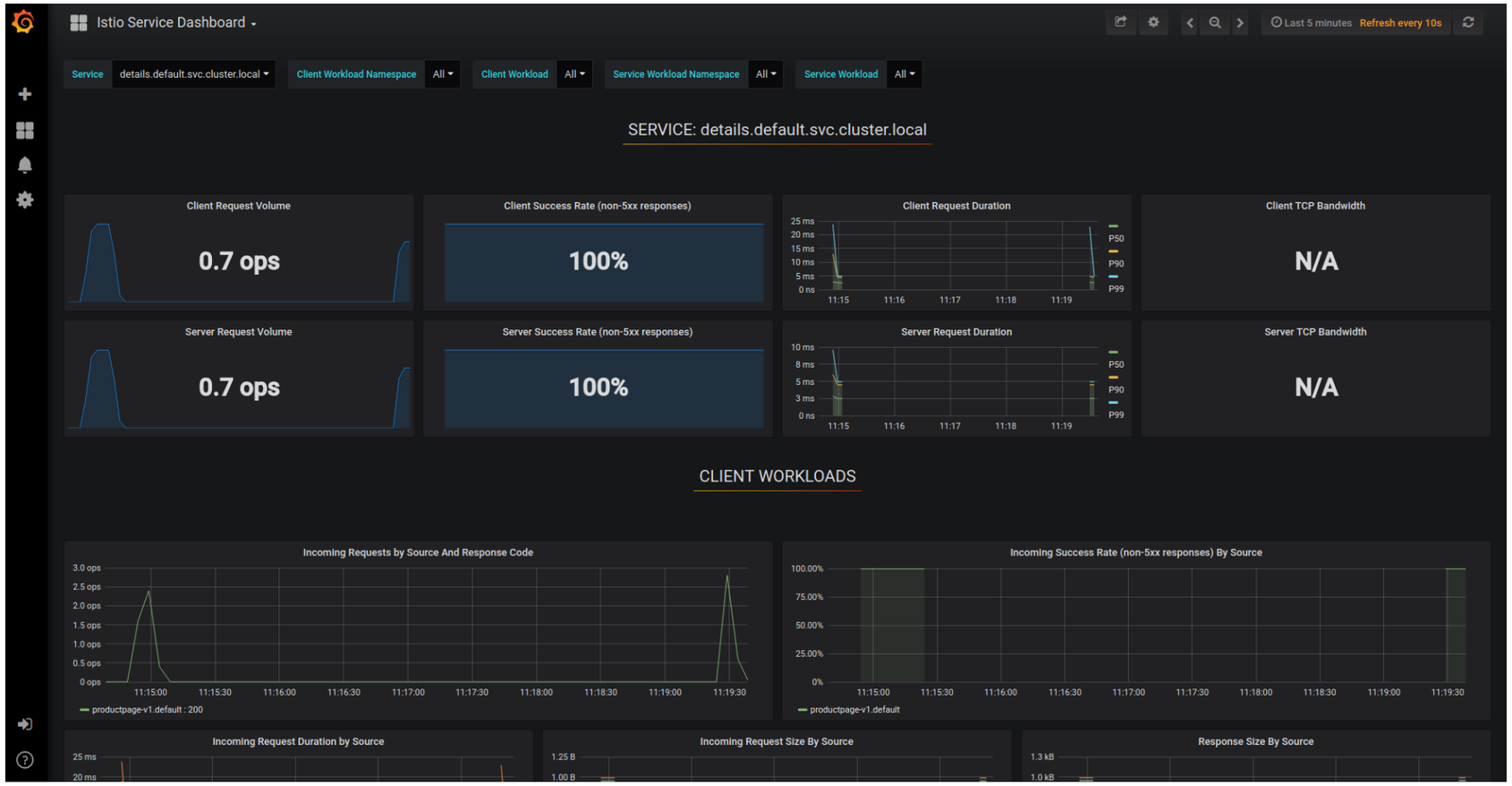
These metrics allow monitoring of the behaviour of Istio itself (as distinct from that of the services within the mesh).

Grafana

```
kubectl apply -f ${ISTIO_INSTALL_DIR}/samples/addons/grafana.yaml
istioctl dashboard grafana
```


Istio Mesh Dashboard
Last 5 minutes
Refresh every 5s
Istio
Istio is an open platform that provides a uniform way to connect, manage, and secure microservices.
Need help? Join the Istio community.
Global Request Volume
1.4 ops
Global Success Rate (non-5xx responses)
100%
4xxs
N/A
5xxs
N/A
HTTP/GRPC Workloads
Service
Workload
Requests
P50 Latency
P90 Latency
P99 Latency
Success Rate
reviews.default.svc.cluster.local
reviews-v3.default
0.02 ops
18 ms
24 ms
25 ms
100.00%
reviews.default.svc.cluster.local
reviews-v2.default
0.05 ops
31 ms
46 ms
50 ms
100.00%
reviews.default.svc.cluster.local
reviews-v1.default
0.07 ops
8 ms
19 ms
24 ms
100.00%
ratings.default.svc.cluster.local
ratings-v1.default
0.07 ops
3 ms
5 ms
5 ms
100.00%
productpage.default.svc.cluster.local
productpage-v1.default
0.15 ops
25 ms
60 ms
96 ms
100.00%
istio-telemetry.istio-system.svc.cluster.local
istio-telemetry.istio-system
0.85 ops
3 ms
7 ms
18 ms
100.00%
istio-policy.istio-system.svc.cluster.local
istio-policy.istio-system
0.29 ops
3 ms
5 ms
5 ms
100.00%
details.default.svc.cluster.local
details-v1.default
0.15 ops
3 ms
5 ms
5 ms
100.00%


Istio Workload Dashboard
Last 5 minutes
Refresh every 10s
Namespace
default
Workload
details-v1
Inbound Workload Namespace
All
Inbound Workload
All
Destination Service
All
WORKLOAD: details-v1.default
Incoming Request Volume
0.15 ops
Incoming Success Rate (non-5xx responses)
100%
Request Duration
10 ms
8 ms
5 ms
3 ms
0 ns
11:15
11:16
11:17
11:18
11:19
P50
P90
P99
TCP Server Traffic
N/A
TCP Client Traffic
N/A
INBOUND WORKLOADS
Incoming Requests by Source And Response Code
3.0 ops
2.5 ops
2.0 ops
1.5 ops
1.0 ops
0.5 ops
0 ops
11:15:00
11:15:30
11:16:00
11:16:30
11:17:00
11:17:30
11:18:00
11:18:30
11:19:00
11:19:30
productpage-v1.default : 200
Incoming Success Rate (non-5xx responses) By Source
100.00%
75.00%
50.00%
25.00%
0%
productpage-v1.default
Incoming Request Duration by Source
Incoming Request Size By Source
1.25 B
1.00 B
Response Size By Source
1.3 kB
1.0 kB


Istio Service Dashboard
Last 5 minutes
Refresh every 10s
Service
details.default.svc.cluster.local
Client Workload Namespace
All
Client Workload
All
Service Workload Namespace
All
Service Workload
All
SERVICE: details.default.svc.cluster.local
Client Request Volume
0.7 ops
Client Success Rate (non-5xx responses)
100%
Client Request Duration
Client TCP Bandwidth
N/A
Server Request Volume
0.7 ops
Server Success Rate (non-5xx responses)
100%
Server Request Duration
Server TCP Bandwidth
N/A
CLIENT WORKLOADS
Incoming Requests by Source And Response Code
productpage-v1.default : 200
Incoming Success Rate (non-5xx responses) By Source
productpage-v1.default
Incoming Request Duration by Source
Incoming Request Size By Source
Response Size By Source

Workshop - Monitoring Blocked External Service Traffic

Observability

Logging

Access Logs

Access logs provide a way to monitor and understand behaviour from the perspective of an individual workload instance.

Istio can generate access logs for service traffic in a configurable set of formats, providing operators with full control of the how, what, when and where of logging.

Istio exposes a full set of source and destination metadata to the access logging mechanisms, allowing detailed audit of network transactions.

Access logs may be generated locally or exported to custom backends, including Fluentd.

Logs can be generated in TEXT or JSON format

Example Istio access log (formatted in JSON):


```
{
  "protocol": "HTTP/1.1",
  "duration": 16,
  "upstream_local_address": "10.60.2.8:43826",
  "response_flags": "-",
  "response_code": 418,
  "path": "/status/418",
  "upstream_transport_failure_reason": null,
  "x_forwarded_for": null,
  "upstream_host": "10.60.1.7:80",
  "upstream_cluster": "outbound|8000||httpbin.default.svc.cluster.local"
  "connection_termination_details": null,
  "start_time": "2021-07-29T08:55:33.467Z",
  "request_id": "9c9ecb6e-c240-4d17-b8c3-7bbac329faa2",
  "user_agent": "curl/7.78.0-DEV",
  "upstream_service_time": "15",
  "response_code_details": "via_upstream",
  "requested_server_name": null,
  "method": "GET",
  "bytes_sent": 135,
  "authority": "httpbin:8000",
  "downstream_remote_address": "10.60.2.8:47826",
  "bytes_received": 0,
  "downstream_local_address": "10.63.243.74:8000",
  "route_name": "default"
}
```

Logging configuration

Istio provide these 3 settings to configure the access log on the Envoy proxies:

- meshConfig.accessLogFile
- meshConfig.accessLogEncoding
- meshConfig.accessLogFormat

Workshops - Visualising metrics with Grafana

Distributed Tracing

Distributed tracing provides a way to monitor and understand behaviour by monitoring individual requests as they flow through a mesh.

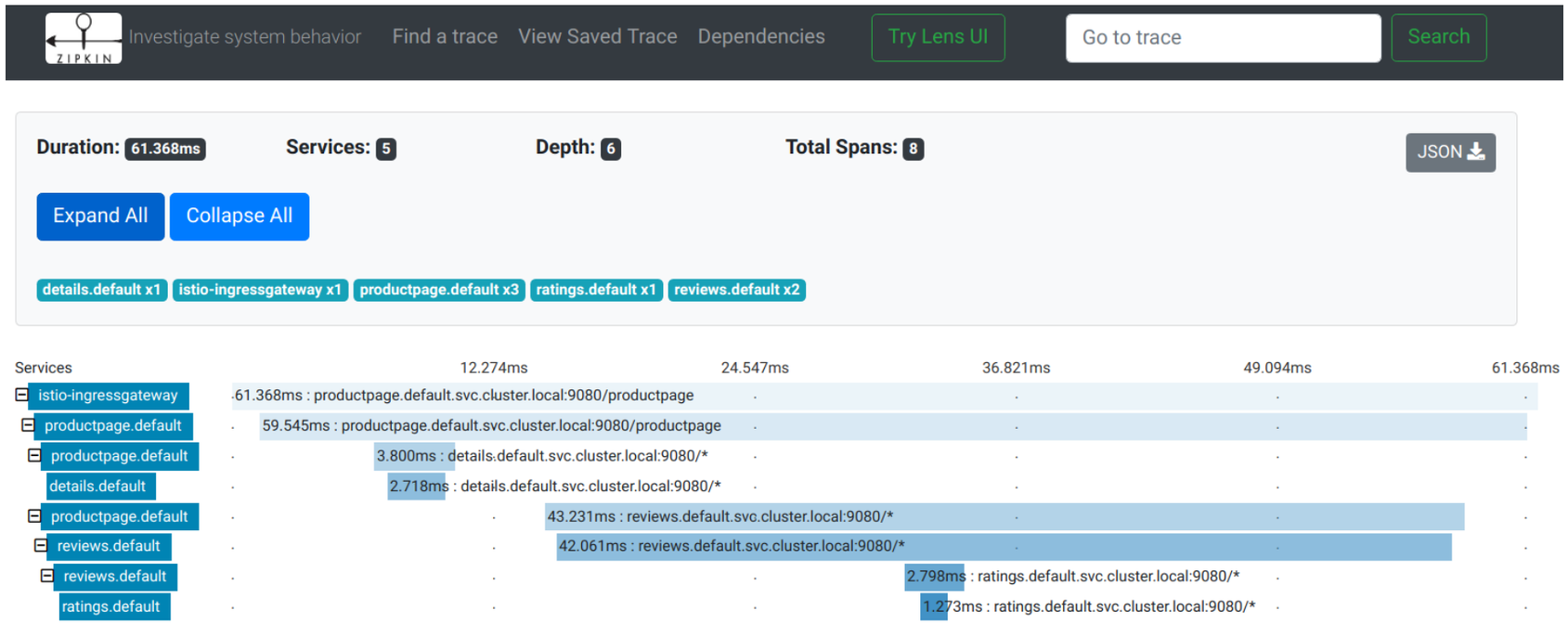
Traces empower mesh operators to understand service dependencies and the sources of latency within their service mesh.

Istio supports distributed tracing through the Envoy proxies.

The proxies automatically generate trace spans on behalf of the applications they proxy, requiring only that the applications forward the appropriate request context.

Istio supports a number of tracing backends, including Zipkin, Jaeger, LightStep, and Datadog.

```
istioctl dashboard zipkin|jaeger
```



Trace context propagation

Istio proxies can automatically send spans, however applications need to propagate the appropriate HTTP headers so that spans can be correlated correctly into a single trace.

To do this, an application needs to collect and propagate the following headers from the incoming request to any outgoing requests:

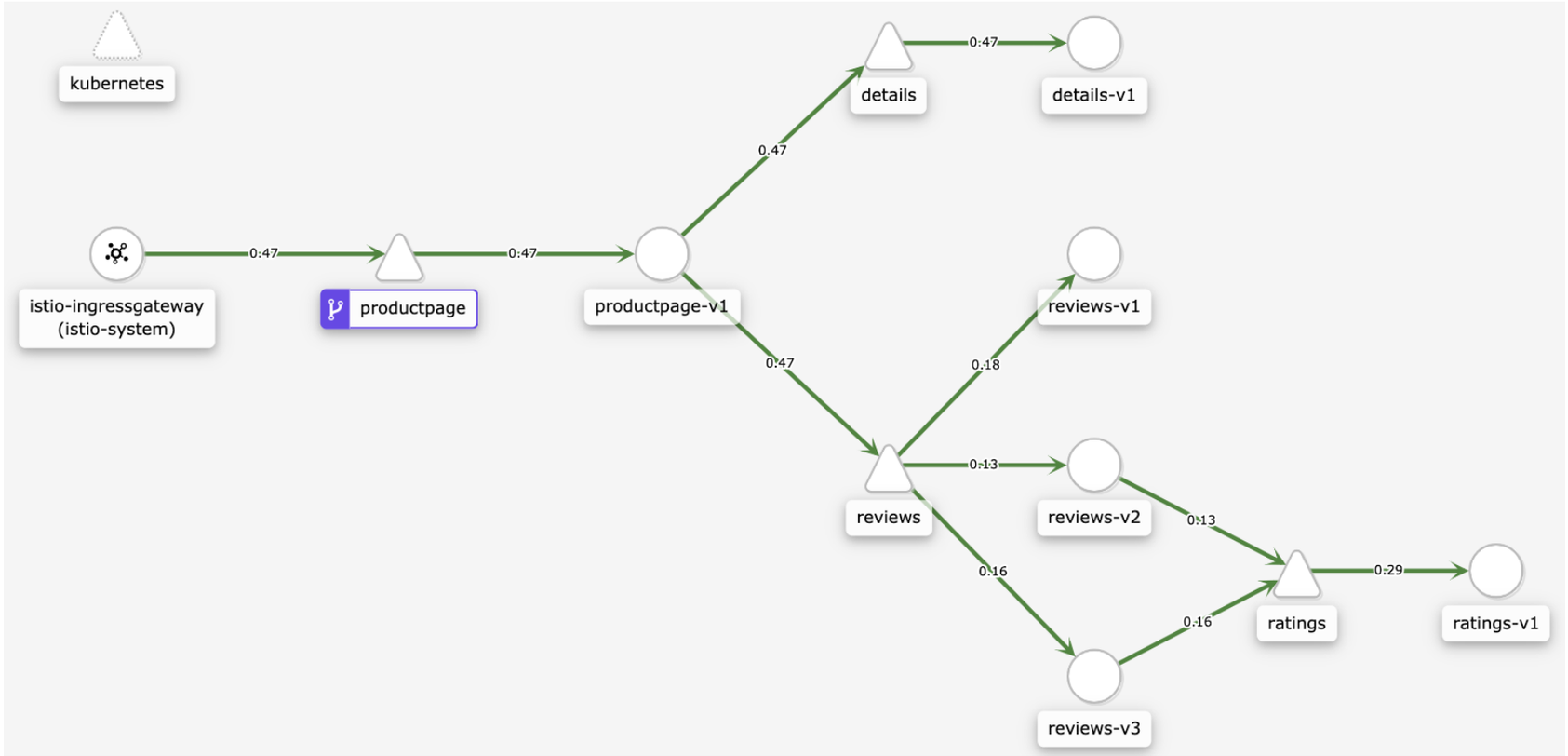
- `x-request-id`
- `x-b3-traceid`
- `x-b3-spanid`
- `x-b3-parentspanid`
- `x-b3-sampled`
- `x-b3-flags`
- `x-ot-span-context`

```
def getForwardHeaders(request):
    headers = {}
    # x-b3-*** headers can be populated using the opentracing span
    span = get_current_span()
    carrier = {}
    tracer.inject(
        span_context=span.context,
        format=Format.HTTP_HEADERS,
        carrier=carrier)
    headers.update(carrier)
    incoming_headers = ['x-request-id']
    for ihdr in incoming_headers:
        val = request.headers.get(ihdr)
        if val is not None:
            headers[ihdr] = val
    return headers
```

Workshops - Tracing requests with Jaeger

Kiali

```
istioctl dashboard kiali
```



Workshops - Observability with Kiali

Finally...

Training

- Kubernetes in Practice (Day One and Day Two)
- Kubernetes for Application Developers (Day One and Day Two)
- Istio in Practice (Day One and Day Two)
- Operational Wargaming
- Kubernetes from Scratch
- Extending Kubernetes with Operators
- Serverless on Kubernetes with Knative

Consulting

On-site / Remote / Project-based

Production Readiness Review

Cloud Native Discovery Workshop

Jetstack Subscription

Jetstack Subscription is designed to help you verify, support and continually optimise your Kubernetes production environment.

- Break Fix Support →
- Customer Reliability Engineering →
- Kubernetes Reference Architecture →
- Automated Cluster Compliance Checking →
- Online Training and Wargaming →

Register Interest

Jetstack is trusted by

monzo Google Cloud babylon Pearson

The End