Inbound Policies

Authors	Ilya Lobkov
Status	Approved
Implementation status	Not Started
Design tracking issue	
Design PR	
Implementation tracking issue	
Reviewers	see table below
Related	Feature board:

Name	Context Status	Design Status
Krzysztof Słonka	Approved Comments: just minor things, LG overall	Approved
Łukasz Dziedziak	Under review	Not started
Charly Molter	Approved	Approved
Mike Beaumont	Approved	Approved

Notes 13/12/2024

- ✅kind Mesh in top-level makes sense for MeshMetric and MeshTrace
  - Do we need a “proxyTypes” field? No
  - when configuration doesn’t makes (i.e. using MeshExternalService inside “appendRules” for top-level Dataplane) we’re going to ignore it
- MeshMetric on ZoneEgress and multitenancy
  - we need it but probably out of the scope
- MeshProxyPatch on ZoneEgress (out of scope)

- ✅“kuma.io/proxy-type” with 2 values, if we want to distinguish between builtin and delegated we can create additional label “kuma.io/gateway-type: builtin | delegated”
- ✅go with “spec.rules[]” and have targetRef inside “matches” and update all examples
- ✅book call for Mon

Notes 16/12/2024

- ✅Do we need “NOT” in “matches”? Can we do NOT “/prefix” with existing routes in envoy?
    - gateway api doesn’t mention “not” in matches
    - it’s should be possible to express “not” X by adding matching for X and a fallback case
    - envoy routes don’t have “not” as well
- referencing MeshExternalService from “matches” is OK. But we probably want to name the field somehow more informative than “matches[].targetRef”
    - check gateway api, maybe there is something to describe destination based on the incoming traffic
- ✅Does ZoneEgress know about MeshMultiZoneService?
    - it doesn’t know at this moment, but it should, I opened an issue https://github.com/kumahq/kuma/issues/12288
    - added 18th user story to support it
- update migration part with what we’ve discussed – we don’t want to generate new “allow-all” policy, it should be a Kuma CP config flag

Context and Problem Statement

In Kuma 2.9 we completed the redesign of outbound (client-side) policies with the real MeshService, MeshExternalService, MeshMultiZoneService in mind. Previously, MeshService was only a logical concept representing a group of DPPs with the same “kuma.io/service” tag. Introducing MeshService resource allowed us to simplify the matching algorithm and improve the Inspect API as it’s now possible to associate “conf” with the real resource in the store.

Inbound policies are still falling short. Applying policies on the inbound (server-side) is possible only by using the "from" array in the policy spec, but this syntax has a number of limitations:

(A) Impossible to configure inbound ports differently on the same proxy

Kubernetes services can have multiple ports. When Kuma creates a data plane resource, it maps each individual port to a dedicated inbound listener. In example, service with 2 ports:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: my-app
  ports:
    - name: http
      protocol: TCP
      port: 10000
      targetPort: 10001
    - name: https
      protocol: TCP
      port: 20000
      targetPort: 20001
```

results in the following Dataplane resource (the config is not complete, only relevant fields are present):

```yaml
apiVersion: kuma.io/v1alpha1
kind: Dataplane
mesh: default
spec:
  networking:
    inbound:
    - name: http
      port: 10001
      tags:
        kuma.io/service: my-service_kuma-demo_svc_10000
        k8s.kuma.io/service-port: "10000"
    - name: https
      port: 20001
      tags:
        kuma.io/service: my-service_kuma-demo_svc_20000
        k8s.kuma.io/service-port: "20000"
```

From-policies use top-level TargetRef to select a proxy (same way as to-policies). Even though different inbounds could be distinguished by k8s.kuma.io/service-port tag, the current behavior of top-level targetRef is to select a proxy if at least one inbound on this proxy is selected. That's why the policy:

```yaml
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: MeshSubset
    tags:
      app: kuma-demo-frontend
      k8s.kuma.io/service-port: "10000"
  from:
  - targetRef:
      kind: Mesh
    default:
      action: Deny
```

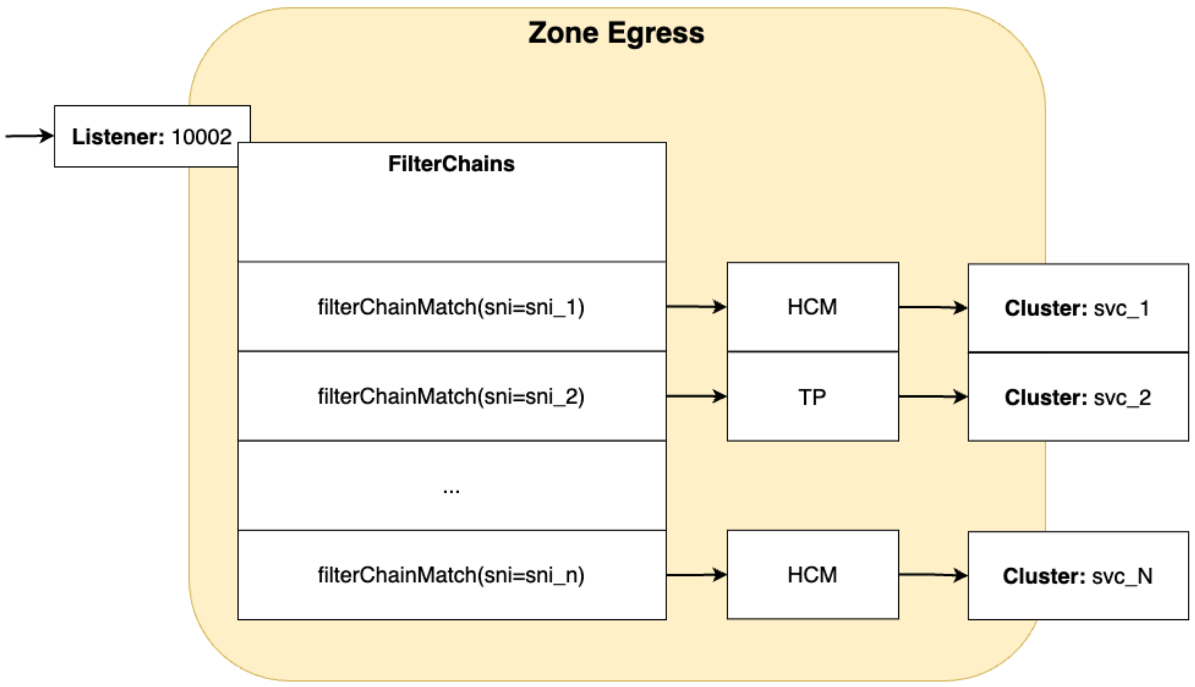
applies to both 10000 and 20000 inbounds.

(B) Impossible to apply inbound policies for MeshExternalService

Policies such as MeshRateLimit, MeshTrafficPermission, MeshAccessLog, MeshFaultInjection, MeshTimeout, MeshTLS don't support MeshExternalService at this moment.

In MeshExternalService and ZoneEgress MADR we decided that all the traffic to MeshExternalServices should always go through ZoneEgress. Though ZoneEgress is now required to use MeshExternalServices this decision simplifies the logic of applying inbound policies for MeshExternalServices – we will always apply policies on ZoneEgress no matter what.

Despite ZoneEgress looks nothing like sidecar and "inbound/outbound" terms don't really apply to it, there is still a filterChains and a cluster per MeshExternalService that could be configured. Which means inbound policies can be applied on filterChain corresponding to MeshExternalService.



(C) Impossible to apply MeshMetrics, MeshTrace and MeshProxyPatch for zone proxies

These single-item policies are extremely useful for running zone proxies in production. Today, Kuma policies don't have a way to target zone proxies (only Sidecar and Gateway are allowed).

(D) Impossible to configure zone proxies for MeshService in cross zone scenarios

Cross-zone communication between mesh services involves intermediate hops through ZoneIngress and eventually ZoneEgress. Though these zone proxies are meant to be fully transparent it might not always be the case. When connections through intermediate components act in unusual manners it's helpful to have a way to increase observability and tweak the configurations. We can't offer much since cross-zone communication is always TCP from a zone proxy point of view, but we can add support for MeshTimeout and MeshAccessLog.

(E) Impossible to apply inbound policies on a specific HTTP route

Inbound policies such as MeshRateLimit, MeshTrafficPermission, MeshAccessLog, MeshFaultInjection, MeshTimeout don't support per-route configuration at this moment. One of the reasons for this is lack of inbound routes. MeshHTTPRoute policy when applied always configures routes on the outbound side. Another reason is similar to the problem (A), inbound policies don't have any syntax except "from" array and thus can't select neither port to configure (problem A) nor inbound route.

(F) Selecting a group of clients subjected for the configuration is possible only with “kind:MeshSubset”

Previously we’ve made a mistake by not differentiating workload and service in our policy API. Any running process is a workload, it can act both as a client and consume services or it can be an endpoint for one or many services. But when it acts as a client we can’t identify it by the service it implements.

At this moment the only way to select a subset of the traffic that should receive the configuration is possible only with “from[].targetRef.kind: MeshSubset”. This method selects a group of workloads by specifying the set of tags the workload should have to be selected. Not only this method doesn’t allow selecting a subset of the traffic by ServiceAccount or SpiffieID, it is also inefficient. Taking into account that:

1. Subsets can have intersections
2. For servers in zone-a there is no way to know what workloads exist in zone-b. That's why the list of rules for MeshTrafficPermission is O(2^N) where N is the number of distinct tags.

As a result, the Inspect API for inbound rules is overwhelmingly detailed (i.e. we might not even have neither zone:east nor namespace:client).

```
None
kind: MeshTrafficPermission
spec:
  from:
    - targetRef:
        kind: MeshSubset
        tags:
          zone: east
      default: conf1
    - targetRef:
        kind: MeshSubset
        tags:
          namespace: client
      default: conf2
---
rules:
  - zone: east
    namespace: client
    action: merge(conf1, conf2)
  - zone: east
    namespace: !client
    action: conf1
```

```
- zone: !east
  namespace: client
  action: conf2
- zone: !east
  namespace: !client
  action: {}
```

User stories

In the braces we have a reference to the problem that needs to be solved to unblock the user story.

1. **As a** service owner **I want** to expose 1 application port with permissive mTLS **so that** external clients (such as health checkers) can use it without being added to the mesh. (A)

2. **As a** service owner **I want** to attach an inbound policy to a specific set of conditions (path, headers...) on an inbound route **so that** I can apply different policies depending on endpoints.

3. **As a** service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

4. **As a** mesh operator **I want** to grant/revoke access to MeshExternalService for a group of workloads **so that** I can implement POLP (principle of least privilege). (B, E)

5. **As a** mesh operator **I want** to use MeshTrafficPermission with MeshExternalServices with autoreachable services feature enabled **so that** I can improve the sidecars performance. (B)

6. **As a** mesh operator **I want** to apply rate limiting to the service exposed to the mesh only as a MeshExternalService **so that** I can protect the service from being overloaded. (B)

7. **As a** mesh operator **I want** to configure the TLS version and ciphers on ZoneEgress as it terminates the connection for traffic to MeshExternalServices **so that** I can fulfill the security requirements. (B)

8. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshExternalService **so that** I can see outgoing requests and their statuses in logs. (B, E)

9. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshService **so that** I can see outgoing connections and traffic in logs. (D)

10. **As a** mesh operator **I want** to enable access logging on ZoneIngress to specific MeshService **so that** I can see outgoing connections and traffic in logs (D)

11. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshServices **so that** I can see outgoing connections and traffic in logs. (D)

12. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshExternalServices **so that** I can see incoming connections and traffic in the zoneEgress logs. (B)

13. **As a** mesh operator **I want** to have a way to apply a proxy level config on zone proxies (MeshMetric, MeshTrace, MeshProxyPatch) (C)

14. **As a** mesh operator **I want** to change timeouts on ZoneEgress to specific MeshExternalService **so that** I can increase the timeout value if the default is too small for the use case. (B, E)

15. **As a** mesh operator **I want** to simulate faulty behaviour of a MeshExternalService **so that** I can perform chaos testing on the system. (B)

16. **As a** mesh operator **I want** to change timeouts on ZoneEgress/ZoneIngress to specific MeshService **so that** I can increase the timeout value if the default is too small for the use case. (D)

17. **As** a service owner **I want** to allow access to my service for a subset of traffic based on the Spiffe ID of the client workload **so that** the client authentication is happening based on the traffic attributed rather than client deployment attributes (such as tags). (F)

18. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a MeshMultiZoneService **so that** I can see outgoing connections and traffic in logs. (D)

Out of scope

- designing inbound routes

Design

Proposed actions

Stop relying on inbound tags (solves A)

In Kubernetes, it's not just a theoretical possibility for two services to select the same workload port – this is exactly how Argo's BlueGreen deployment strategy works. Today, when Kuma generates a Dataplane object, it assigns the "kuma.io/service" tag to each inbound, acting as if two services can never select the same port.

Instead of relying on "inbound[].tags," we should transition to using "metadata.labels". Labels on the Dataplane object are Pod labels plus auto-generated Kuma labels i.e. "kuma.io/zone" or "kuma.io/origin".

We won't cover all the aspects of actually removing "inbound[].tags" in this MADR, but we believe the long-term the Dataplane object should look something like:

```
None
kind: Dataplane
metadata:
  labels: $pod_labels
spec:
  inbound:
    - port: 8080
      name: api
    - port: 9090
      name: admin
status:
  meshServices: [...]
```

Once the Dataplane resource has "metadata.labels" we can introduce a new kind Dataplane for the top-level targetRef:

```
None
targetRef:
  kind: Dataplane
  labels:
    app: backend
  sectionName: http-api
```

By using “sectionName” we can select an inbound and solve the problem A. The field “sectionName” has to work exactly the same way it works for the kind “MeshService”. Given the MeshService with 2 ports:

```None
kind: MeshService
metadata:
  name: my-service
spec:
  ports:
    - port: 8080
      name: http-port
    - port: 9090
```

we can reference individual ports in outbound policy like:

```None
spec:
  to:
    - targetRef:
        kind: MeshService
        name: my-service
        sectionName: http-port
    - targetRef:
        kind: MeshService
        name: my-service
        sectionName: "9090"
```

The field “sectionName” selects the port by name, but if name is not specified it’s possible to select the port by number passed as a string.

Do we keep “kind:Mesh” and “kind:MeshSubset”?

The “MeshSubset” kind should be deprecated. It does not reference actual resources and offers no functionality that cannot already be achieved with the “Dataplane” kind.

The “Mesh” kind stays and acts as an equivalent to “kind: *” which includes Dataplane, ZoneIngress and ZoneEgress. When using “kind: Mesh” and the conf makes sense only for zone proxies then it simply should be ignored for the kind “Dataplane”.

Also we have to deprecate “proxyTypes” as it’s used only with “MeshSubset” and “Mesh”.

New top-level targetRef order

With added topLevel targetRef `Dataplane` we will need to update targetRef priorities. New priorities in order from least to most important:

```None
Mesh -> Dataplane -> MeshSubset (deprecated) -> Dataplane with labels ->
Dataplane with sectionName
```

For new policies with “rules” we don’t allow using “MeshSubset” in the top-level targetRef since no backward compatibility is required.

Do we allow using “proxyTypes” with “kind:Dataplane” or should we add the “kuma.io/proxy-type” label?

The option to add the label to distinguish gateway proxies from sidecars was already considered in 037-configure-all-gateways MADR with the following pros/cons:

Positive Consequences
- Users can set things up for all gateways.
- We can use different defaults for gateways than for the whole Mesh.
- No need for a new Kind.

Negative Consequences
- Less explicit
- We need to add the tag to all gateways.

Since we are now adding labels to all Kuma resources, introducing the “kuma.io/proxy-type” label is no longer a drawback. During the migration, Kuma will reconcile all Dataplane resources, ensuring the new label is added automatically without requiring any extra effort from the user.

The new “kuma.io/proxy-type” label requires at least 2 values “sidecar” and “gateway”. By using kind “Dataplane” and “labels” selector we can fully replicate the functionality of “proxyTypes” field:

```None
# replacement for 'proxyTypes: ["Sidecar"]'
```

```
spec:
  targetRef:
    kind: Dataplane
    labels:
      kuma.io/proxy-type: sidecar
---
# replacement for 'proxyTypes: ["Gateway"]'
spec:
  targetRef:
    kind: Dataplane
    labels:
      kuma.io/proxy-type: gateway
```

For delegated gateway “kuma.io/proxy-type” label is equal to “sidecar”. For builtin gateway it’s “kuma.io/proxy-type: gateway”.

If there is ever a need to distinguish between “builtin” and “delegated” gateways we can introduce a new label “kuma.io/gateway-type: builtin | delegated” (note that this will not be done as part of this ongoing work).

Replace all from-policies with the single item equivalent (solves E, F)

All policies with “from” list that look like:

```
None
spec:
  targetRef: $top_level_tr
  from:
    - targetRef: $tr1
      default: $conf1
    - targetRef: $tr2
      default: $conf2
```

should be converted to a policy with “rules”:

```
None
spec:
  targetRef: $top_level_tr
  rules:
    - matches: CONVERT_FUNC($tr1)
      default: $conf1
    - matches: CONVERT_FUNC($tr2)
      default: $conf2
```

where CONVERT_FUNC is some way to convert the old-looking targetRef with kinds “Mesh” or “MeshSubset” to a new style matches.

The existing “spec.from” field must be deprecated in all policies.

Policies that have both “spec.to” and “spec.rules” will be considered invalid. We already don’t allow mixing “spec.to” and “spec.from” for namespace-scoped policies.

The field “matches” works the same way as explained in gateway api:

Each match is independent, i.e. this rule will be matched if any single match is satisfied.

```
None
spec:
  rules:
    - matches:
        - method: GET
        - method: POST
          spiffeID: spiffe://trusted-domain/trusted-client
```

For a request to match against this rule, it must satisfy either of the following conditions:

- HTTP method is “GET”
- HTTP method is “POST” AND spiffeID is “spiffe://trusted-domain/trusted-client”

Do we need “not” in “matches”?

No. Gateway API doesn’t provide an explicit way to express “not”, instead if we need “not X” we’d have to match “X” and create a fallback rule. Additionally, Envoy doesn’t provide a way to express “not” with HTTP routes.

Merging behaviour

Same as MeshHTTPRoute merging:

1. Pick policies that select the DPP with top-level targetRef

2. Sort these policies by top-level targetRef
3. Concatenate all “rules” arrays from these policies
4. Merge rules items with the same “hash(rules[i].matches)”
    a. the new rule position is the highest of two merged rules
5. Stable sort rules based on the **Extended GAPI matches order**
    a. we need to split individual matches into separate rules

Extended GAPI matches order (based on gateway-api)

1. “Exact” spiffeID
2. “Prefix” spiffeID
3. “Exact” path match.
4. “Prefix” path match with largest number of characters.
5. Method match.
6. Largest number of header matches.
7. Largest number of query param matches.

The resulting list of rules will be translated to the appropriate envoy mechanism (HTTP route, RBAC policies or matching API in the future), assuming the rule with the most priority is going to be hit first by the request.

Policies examples

MeshTrafficPermission

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches:
        - serviceAccount: frontend
      default:
        action: Allow
    - matches:
        - spiffeID: spiffe://trusted-domain/trusted-client
      default:
        action: Allow
```

With the new design MeshTrafficPermission is going to lose the ability to select a subset of incoming traffic based on the inbound tags that are encoded as “kuma://” URIs into the client’s cert. The approach with multiple URIs is not sustainable as it violates the SVID format:

Validators encountering an SVID containing more than one URI SAN MUST reject the SVID.

Selecting a subset of incoming traffic will be possible either based on the identity (spiffeID or service account) or based on the L7 matchers.

Rewrites of existing MTPs:

```
None
# allow all
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec:
  from:
  - targetRef:
      kind: Mesh
    default:
      action: Allow
---
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec: # empty targetRef has kind "Mesh", so it applies on zone proxies as well
  rules:
    - default:
        action: Allow
```

MeshTLS

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-strict
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: tls
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: "/metrics"
      default:
        mode: Permissive
    - matches:
        - path:
            type: PathPrefix
            value: "/"
      default:
        mode: Strict
```

MeshRateLimit

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: kuma-system
  name: mrl
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches:
        - method: POST
      default:
        local:
          http:
            requestRate:
              num: 5
              interval: 10s
```

MeshTimeout

```
None
apiVersion: kuma.io/v1alpha1
```

```
kind: MeshTimeout
metadata:
  name: inbound-timeout
  namespace: kuma-demo
  labels:
    kuma.io/mesh: default
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches:
        - method: POST
      default:
        idleTimeout: 20s
        connectionTimeout: 2s
        http:
          requestTimeout: 10s
```

MeshAccessLog

None
```
kind: MeshAccessLog
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches: [] # http matching when supported
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

MeshFaultInjection

```
None
kind: MeshFaultInjection
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches: [] # http matching when supported
      default:
        http:
          - abort:
              httpStatus: 500
              percentage: "20"
```

#### Example of merging

When merging the rules section we need to merge entries by `matches` to get effective configuration for incoming traffic subset. For example with given policies applied:

```
None
# mesh opeator policy to allow requests from infra on /heatlh
kind: MeshTrafficPermission
metadata:
  name: mtp-1
spec:
  rules:
    - matches:
        - path:
            type: Exact
            value: /health
          spiffe:
            type: Prefix
            value: spiffe://infra/
      default:
        action: Allow
---
kind: MeshTrafficPermission
metadata:
  name: mtp-2
```

```yaml
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: demo-app
  rules:
    - matches:
        - method: GET
        - method: POST
          spiffe:
            type: Prefix
            value: spiffe://frontend/
      default:
        action: Allow
---
kind: MeshTrafficPermission
metadata:
  name: mtp-3
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: demo-app
    sectionName: http-api
  rules:
    - matches:
        - method: GET
      default:
        action: Deny
---
kind: MeshTrafficPermission
metadata:
  name: mtp-4
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: demo-app
    sectionName: http-api
  rules:
    - matches:
        - path:
            type: Exact
            value: /health
          spiffe:
            type: Prefix
            value: spiffe://infra/
      default:
        action: Deny
```

Let’s find the configuration for “http-api” port of the “demo-app”:

1. **Pick policies that select the DPP with top-level targetRef**
   All 4 policies will be picked for “http-api” port
2. **Sort these policies by top-level targetRef (most to least priority)**
   mtp-4 > mtp-3 > mtp-2 > mtp-1
3. **Concatenate all “rules” arrays from these policies**

```None
rules:
    - matches: # from mtp-4
        - path:
            type: Exact
            value: /health
          spiffe:
            type: Prefix
            value: spiffe://infra/
      default:
        action: Deny
    - matches: # from mtp-3
        - method: GET
      default:
        action: Deny
    - matches: # from mtp-2
        - method: GET
        - method: POST
          spiffe:
            type: Prefix
            value: spiffe://frontend/
      default:
        action: Allow
    - matches: # from mtp-1
        - path:
            type: Exact
            value: /health
          spiffe:
```

```
          type: Prefix
          value: spiffe://infra/
    default:
      action: Allow
```

4. **Merge rules items with the same “hash(rule[i].matches)”**

```
None
rules:
    - matches: # merge of mtp-1 and mtp-4
        - path:
            type: Exact
            value: /health
          spiffe:
            type: Prefix
            value: spiffe://infra/
      default:
        action: Deny
    - matches: # from mtp-3
        - method: GET
      default:
        action: Deny
    - matches: # from mtp-2
        - method: GET
        - method: POST
          spiffe:
            type: Prefix
            value: spiffe://frontend/
      default:
        action: Allow
```

5. **Stable sort rules based on the extended GAPI matches order**

we need to split individual matches into separate rules

```
None
rules:
    - matches: # merge of mtp-1 and mtp-4
        - path:
            type: Exact
            value: /health
          spiffe:
            type: Prefix
            value: spiffe://infra/
      default:
        action: Deny
    - matches: # from mtp-2
        - method: POST
          spiffe:
            type: Prefix
            value: spiffe://frontend/
      default:
        action: Allow
    - matches: # from mtp-3
        - method: GET
      default:
        action: Deny
    - matches: # from mtp-2
        - method: GET
      default:
        action: Allow
```

Add top-level targetRef kind ZoneEgress/ZoneIngress (solves B,C,D)

To simplify this experience for users we want to introduce new topLevel targetRef kind: `ZoneIngress` and `ZoneEgress`. To select all zone egresses you could just create policy with topLevel targetRef:

```
None
spec:
  targetRef:
    kind: ZoneEgress
```

We should be able to use ZoneEgress/ZoneIngress with `labels` to select a subset of proxies. For example, a user is creating a policy on global but wants it to be applied only on one zone. Example of this targeting:

```
None
spec:
  targetRef:
    kind: ZoneEgress
    labels:
      kuma.io/zone: zone-1
```

Applying config to MeshExternalService/MeshService on ZoneEgress/ZoneIngress

For targeting MeshExternalService and MeshService on zone proxies we basically have two options:

1. Treat is as now as destination and put it in `to` section
2. Use proposed in this madr structure with rules

Ad 1. Example with MeshTimeout configuration

```
None
kind: MeshTimeout
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      default:
        idleTimeout: 30s
```

Advantages:

- We treat MeshExternalService as destination as we do now
- We already have mechanism for targeting MES in to section

Disadvantages:

- It is not consistent with how we target zone proxies in inbound policies, and zone proxy does not use inbound/outbound concepts. For example you could apply timeout to whole ZoneEgress like this:

```
None
kind: MeshTimeout
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - default:
        idleTimeout: 30s
```

But when you want to apply it to MES you would need to use form with `to`:

```
None
kind: MeshTimeout
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      default:
        idleTimeout: 30s
```

Ad 2. Example with MeshTimeout configuration

```
None
kind: MeshTimeout
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      matches:
        - path: /slow-endpoint
      default:
        idleTimeout: 30s
```

For targetRef inside rules, we allow: Mesh, MeshService, MeshMultizoneService and MeshExternalService. When targetRef is empty, Mesh kind will be used by default. Which means it matches every resource from this Mesh.

Advantages:

- This is consistent with redesigned inbound policies
- Single way of configuring zone proxies
- Consistent with Gateway API

Disadvantages:

-

Merging

When applying configuration on ZoneEgress/ZoneIngress we would need to merge configuration per targetRef. To get effective configuration for real resources on a given zone proxy.

1. Pick policies that select the zone proxies with top-level targetRef
2. Sort these policies by top-level targetRef

```None
Mesh > ZoneEgress/ZoneIngress > ZoneEgress/ZoneIngress with labels
```

3. Concatenate all “rules” arrays from these policies
4. Resolve “rules[i].targetRef” to the real resources (to avoid ambiguity when using name/namespace vs labels)
5. Group rules by “rules[i].targetRef”
    a. map[ResourceIdentifier]Rules
6. Do steps 4 and 5 from the original merging algo for each (ResourceIdentifier, Rules) pair

Decision

We have decided that we will go with option 2. We will use the rules section with targetRef pointing at MeshExternalService should be at the same level as matches. This gives us a clear distinction between incoming and outgoing traffic. This will also be easier for merging policies. This approach is also consistent with Gateway API `backendRef`. Example from Gateway API docs with timeout on route:
https://gateway-api.sigs.k8s.io/api-types/httproute/?h=httproute#timeouts-optional

Examples

1. **As a** service owner **I want** to expose 1 application port with permissive mTLS **so that** external clients (such as health checkers) can use it without being added to the mesh. (A)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-strict
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: tls
  rules:
    - default:
        mode: Strict
---
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-permissive
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: plaintext
  rules:
    - default:
        mode: Permissive
```

2. **As a** service owner **I want** to attach an inbound policy to a specific set of conditions (path, headers...) on an inbound route **so that** I can apply different policies depending on endpoints.

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: mtp-on-route
  namespace: kuma-demo
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
  rules:
    - matches:
        - path: /secured
          spiffeId: spiffe://trusted-domain/trusted-client
      default:
        action: Allow
```

3. **As a** service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshHTTPRoute
metadata:
  namespace: backend-ns
  name: compute-to-hp-compute
  labels:
    kuma.io/policy-role: producer
spec:
  targetRef:
    kind: Dataplane
    labels:
      backend-access: priority
  to:
    - targetRef:
        kind: MeshService
        name: backend
      rules:
          default:
            filter:
              type: RequestHeaderModifier
              requestHeaderModifier:
                add:
                  - name: priority-access
                    value: enabled
---
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: backend-ns
  name: high-priority
spec:
  rules:
    - matches:
        - headers:
            type: Exact
            name: priority-access
            value: enabled
      default:
        local:
          http:
            requestRate:
              num: 100
              interval: 1s
---
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: backend-ns
  name: default
spec:
  rules:
    - default:
        local:
          http:
            requestRate:
              num: 10
              interval: 1s
```

4. **As a** mesh operator **I want** to grant/revoke access to MeshExternalService for a group of workloads **so that** I can implement POLP (principle of least privilege). (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: mtp-on-mes
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - matches:
        - spiffeId: spiffe://trusted-domain/trusted-client
      targetRef:
        kind: MeshExternalService
        name: httpbin
      default:
        action: Allow
```

5. **As a** mesh operator **I want** to use MeshTrafficPermission with MeshExternalServices with autoreachable services feature enabled **so that** I can improve the sidecars performance. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: httpbin-access
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - matches:
        - method: POST
          spiffeId: spiffe://trusted-domain/trusted-client
      targetRef:
        kind: MeshExternalService
        name: httpbin
      default:
        action: Allow
```

When building the reachable service graph the client should know what SPIFFE identity or service account is in use. By iterating over the list of rules it's possible to take only rules that're matched for the client's identity. The HTTP section of the rule under "matches" has to be disregarded. From the resulting list of rules it's possible to figure out what MeshExternalServices are accessible to the client.

6. **As a** mesh operator **I want** to apply rate limiting to the service exposed to the mesh only as a MeshExternalService **so that** I can protect the service from being overloaded. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: kuma-system
  name: mrl-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - matches:
        - spiffeID: spiffe://trusted-domain/hello
          path: ...
      targetRef:
        kind: MeshExternalService
        name: httpbin
      default:
        local:
          http:
            requestRate:
              num: 5
              interval: 10s
```

7. **As a** mesh operator **I want** to configure the TLS version and ciphers on ZoneEgress as it terminates the connection for traffic to MeshExternalServices **so that** I can fulfill the security requirements. (B)

Updating TLS version and ciphers is possible only when both “targetRef.kind” and “from[].targetRef.kind” are “Mesh”. With the new inbound policies we have to validate top-level targetRef is “Mesh” and “rules” has the only one item with empty matches.

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-strict
  namespace: kuma-system
spec:
  targetRef:
    kind: Mesh
  rules:
    - default:
        tlsVersion:
          min: TLS13
          max: TLS13
        tlsCiphers:
          - ECDHE-ECDSA-AES256-GCM-SHA384
```

8. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshExternalService **so that** I can see outgoing requests and their statuses in logs. (B, E)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

9. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshService **so that** I can see outgoing connections and traffic in logs. (D)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

10. **As a** mesh operator **I want** to enable access logging on ZoneIngress to specific MeshService **so that** I can see outgoing connections and traffic in logs (D)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ingress
spec:
  targetRef:
    kind: ZoneIngress
  rules:
    - targetRef:
        kind: MeshService
        name: demo-app
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

11. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshServices **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneEgress # or ZoneIngress
  rules:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/zone: zone-of-interest
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

12. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshExternalServices **so that** I can see outgoing connections and traffic in logs. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshExternalService
        labels:
          access-logging: enabled
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

13. **As a** mesh operator **I want** to have a way to apply a proxy level config on zone proxies (MeshMetric, MeshTrace, MeshProxyPatch)

DISCLAIMER: since Zone proxies are multi mesh policies like MeshMetric and MeshTrace will need extra work to accommodate it to this. This is out of scope of this MADR, it will be covered in a separate document.

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshMetric
metadata:
  name: metrics-default
  namespace: kuma-system
  labels:
    kuma.io/mesh: default
spec:
  targetRef:
    kind: ZoneEgress
  default:
    backends:
    - type: Prometheus
      prometheus:
        port: 5670
        path: "/metrics"
```

14. **As a** mesh operator **I want** to change timeouts on ZoneEgress to specific MeshExternalService **so that** I can increase the timeout value if the default is too small for the use case. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
```

```
      default:
        idleTimeout: 30s
```

15. **As a** mesh operator **I want** to simulate faulty behaviour of a MeshExternalService **so that** I can perform chaos testing on the system. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshFaultInjection
metadata:
  name: faults-on-httpbin
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      default:
        http:
          - abort:
              httpStatus: 500
              percentage: 50
```

16. **As a** mesh operator **I want** to change timeouts on ZoneEgress/ZoneIngress to specific MeshService **so that** I can increase the timeout value if the default is too small for the use case. (D)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-for-mes
spec:
  targetRef:
    kind: ZoneIngress
  rules:
    - targetRef:
        kind: MeshService
        name: kuma-demo
      default:
        idleTimeout: 30s
```

17. **As** a service owner **I want** to allow access to my service for a subset of traffic based on the Spiffe ID of the client workload **so that** the client authentication is happening based on the traffic attributed rather than client deployment attributes (such as tags).

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: httpbin-access
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
  rules:
    - matches:
        - spiffeId: spiffe://trusted-domain/trusted-client
      default:
        action: Allow
```

18. **As a** mesh operator **I want to** enable access logging on ZoneIngress/ZoneEgress to a MeshMultiZoneService **so that** I can see outgoing connections and traffic in logs. (D)

As we've learned zone proxies are aware of MeshMutliZoneServices and there are separate filter chains for the MeshService and MeshMultiZoneService.

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mmzs
```

```
spec:
  targetRef:
    kind: ZoneEgress # or ZoneIngress
  rules:
    - targetRef:
        kind: MeshMultiZoneService
        labels:
          kuma.io/display-name: multizone-backend
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

Migration

Adding labels to Dataplane resource

At the moment we are adding some labels to the Dataplane resource like: kuma.io/env, kuma.io/mesh, kuma.io/origin, kuma.io/zone. We have a good process for adding extra labels.

1. Kuberetes - adding labels from pods or adding kuma.io/proxy-type labels should be easy, and they will be added automatically on resource reconciliation by control plane.
2. Universal - for users to start using new Dataplane kind in targetRef users will need to manually add labels on Dataplane resource based on existing tags on inbounds

Migrating old “from” policies to “rules”

After adding a new API with `rules` field, old policies containing the “from” section should work as previously.

For MeshTrafficPermission we could be able to combine old “from” with new “rules”. Since this is an RBAC filter we can chain them and put new rules on top of old ones.
For other policies: MestTLS, MeshRateLimit, MeshTimeout, MeshAccessLog, MeshFaultInjection we can utilize shadow policies for migration. Unfortunately we cannot use the same approach like with MeshTrafficPermission, since some policies are configured on clusters.

Decision

We have decided that for now we will advise against mixing old “from” policies with “rules” policies as behavior is undefined. We can think of a consistent migration strategy in the future if it will be needed.

MeshTrafficPermission and MeshExternalService

At this moment you don't need MeshTrafficPermission when mTLS is configured on Mesh to reach MeshExternalService (it is implicitly allowed by default). When we introduce MTP for MES traffic should be blocked by default. This will be problematic for users already using MES for communicating with services from outside the Mesh.

To solve this issue we need to be able to specify all MeshTrafficPermission targeting only ZoneEgress before we disable traffic. Example of this policy:

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: allow-all-mes
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  rules:
    - default:
        action: Allow
```

With this applied whole traffic will work as previously, and users can gradually add more specific policies for clients.

This approach has one problem, users need to be able to apply this policy before we change default accessibility for MeshTrafficPermission. We can solve this in two ways:

1. Preserve accessibility of MeshExternalService by default after adding MTP, and change this behaviour in next release
2. Add configuration flag that will preserve old behaviour and allow traffic flow without MTP.
3. Don't do anything since MeshExternalService is still experimental and we can break traffic

Decision

We have decided to go with option 3. Since MeshExternalService is still experimental we can just change default behaviour and block traffic without MeshTrafficPermission applied on Mesh with mTLS enabled. We should mention this with an example of Allow All MeshTrafficPermission in the UPGRADE.md file.

[Rejected, ignore for review] Alternative designs

[ ]Create “on” section

- not relying on inbound tags
- replacing from-policies with single item, etc

- 1 inbound port can be selected by multiple services, that’s why “kuma.io/service” tag makes no sense
- we have to stop relying on “inbound[].tags” and use “labels” instead (todo: maybe open issue for “status”)

```
None
kind: Dataplane
metadata:
  labels: $pod_labels
spec:
  inbound:
    - port: 8080
      name: api
    - port: 9090
      name: admin
```

- “targetRef.tags” is obsolete in this case, we can use the following to select workloads and their ports (similar to MeshService)

```
None
targetRef:
  kind: Dataplane
  labels: {}
  sectionName: ""
```

- we can introduce “kind: ZoneIngress” and “kind: ZoneEgress” to target zone proxies
- rename “from” to “rules”
    - looks similar to MeshHTTPRoute when applied for MeshExternalServices on ZoneEgress (especially if we want to support “matches”)
    - look less awkward if we want to allow empty “targetRef.kind: Mesh”

None
```
spec:
  rules:
    - default:
        action: Allow
---
spec:
  from:
    - default:
        action: Allow
```

None
```
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: mtp-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - # match the traffic from DPP
          default:
            action: Allow
---
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: mtp-for-mes
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches:
       - path: "/hello"
         spiffeID: "spiffe://client"
      default:
        action: Allow
```

todo: what autoreachable will look like with this rules ^

Examples (meshttproute style rules, discussed on 05/12/24)

1. **As a** service owner **I want** to expose 1 application port with permissive mTLS **so that** external clients (such as health checkers) can use it without being added to the mesh. (A)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-strict
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: tls
  rules:
    - targetRef:
        kind: Mesh
      default:
        mode: Strict
---
apiVersion: kuma.io/v1alpha1
kind: MeshTLS
metadata:
  name: backend-permissive
  namespace: kuma-system
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: plaintext
  rules:
    - targetRef:
        kind: Mesh
      default:
        mode: Permissive
```

2. **As a** service owner **I want** to attach an inbound policy to a specific set of conditions (path, headers...) on an inbound route **so that** I can apply different policies depending on endpoints.

This should be handled in the “inbound routes” MADR.

3. **As a** service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

Requires inbound route functionality, should be handled in the “inbound routes” MADR.

4. **As a** mesh operator **I want** to grant/revoke access to MeshExternalService for a group of workloads **so that** I can implement POLP (principle of least privilege). (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: mtp-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
       - path: /foo
      default:
        action: Allow
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - path: /bar
      default:
        action: Deny
```

5. **As a** mesh operator **I want** to use MeshTrafficPermission with MeshExternalServices with autoreachable services feature enabled **so that** I can improve the sidecars performance. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  name: httpbin-access
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      rules:
        - targetRef:
            kind: MeshSubset
            tags:
              httpbin: allow
          default:
            action: Allow
```

6. **As a** mesh operator **I want** to apply rate limiting to the service exposed to the mesh only as a MeshExternalService **so that** I can protect the service from being overloaded. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: kuma-system
  name: mrl-for-mes
```

```
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - default:
            local:
              http:
                requestRate:
                  num: 5
                  interval: 10s
```

7. **As a** mesh operator **I want** to configure the TLS version and ciphers on ZoneEgress as it terminates the connection for traffic to MeshExternalServices **so that** I can fulfill the security requirements. (B)

Updating TLS version and ciphers is possible only when both “targetRef.kind” and “from[].targetRef.kind” are “Mesh”. Adding functionality to apply MeshTLS on zone egress should be in the same MADR as applying MeshTLS on delegated gateway.

8. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshExternalService **so that** I can see outgoing requests and their statuses in logs. (B, E)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
       default:
         backends:
           - type: File
             file:
               path: "/dev/stdout"
```

9. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshService **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
      default: {}
      rules:
         - default:
             backends:
               - type: File
                 file:
                   path: "/dev/stdout"
---
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  to:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
      default:
        backends:
         - type: File
           file:
             path: "/dev/stdout"
```

10. **As a** mesh operator **I want** to enable access logging on ZoneIngress to specific MeshService **so that** I can see outgoing connections and traffic in logs (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneIngress # or ZoneEgress
  to:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
      default:
         backends:
           - type: File
             file:
               path: "/dev/stdout"
```

11. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshServices **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneIngress # or ZoneEgress
  to:
    - targetRef:
        kind: MeshService
      rules:
        - matches: ?
          default:
            backends:
              - type: File
                file:
                  path: "/dev/stdout"
```

12. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshExternalServices **so that** I can see outgoing connections and traffic in logs. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: ZoneIngress # or ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        labels:
          log-on-egress: true
      default:
        backends:
          - type: File
            file:
              path: "/dev/stdout"
```

13. **As a** mesh operator **I want** to have a way to apply a proxy level config on zone proxies (MeshMetric, MeshTrace, MeshProxyPatch)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshMetric
metadata:
  name: metrics-default
  namespace: kuma-system
  labels:
    kuma.io/mesh: default
spec:
  targetRef:
    kind: ZoneEgress
  default:
    backends:
    - type: Prometheus
      prometheus:
        port: 5670
        path: "/metrics"
```

14. **As a** mesh operator **I want** to change timeouts on ZoneEgress to specific MeshExternalService **so that** I can increase the timeout value if the default is too small for the use case. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - default:
            connectTimeout: 10s
```

15. **As a** mesh operator **I want** to simulate faulty behaviour of a MeshExternalService **so that** I can perform chaos testing on the system. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshFaultInjection
metadata:
  name: faults-on-httpbin
  namespace: kuma-system
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      rules:
        - default:
            http:
              - abort:
                  httpStatus: 500
                  percentage: 50
```

16. **As a** mesh operator **I want** to change timeouts on ZoneEgress/ZoneIngress to specific MeshService **so that** I can increase the timeout value if the default is too small for the use case. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: ZoneEgress # or ZoneIngress
  to:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
      rules:
        - default:
            connectTimeout: 10s
```

17. **As** a service owner **I want** to allow access to my service for a subset of traffic based on the Spiffe ID of the client workload **so that** the client authentication is happening based on the traffic attributed rather than client deployment attributes (such as tags).

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: spiffe-based
spec:
  targetRef:
    kind: Dataplane
    labels:
      app: backend
    sectionName: http-api
  rules:
    - matches:
        - spiffeID: spiffe://orders/frontend
      default:
        action: Allow
```

STOP READING

[WIP] Design

In this design we are working on a user facing policy API. We have three ideas that we consider that should allow users to configure Kuma according to needs from listed user stories.

Ideas

Introduce new “on” keyword that can accept either port or targetRef for MES

This is most likely the simplest idea. We will introduce a new keyword `on` that can accept a list of `ports` or `targetRef`. Simple example:

```None
# select inbound
spec:
  on:
    - port: 8080
---
# select MeshExternalService on zone proxy
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
```

When we select `port` we will apply configuration to Dataplane inbound. When MES is selected by targetRef we will apply configuration on ZoneEgress. To apply configuration on data plane you will need to use as previously `to` section in policy.

Introduce new “on” keyword and use only targetRef for selecting inbounds

In this idea as previously we introduce new keyword `on` but we allow only usage of `targetRef`. If we want to preserve the rule that policies should select real resources we would need to create new resource for Dataplane inbounds. For example `Inbound` or `MeshInbound`. For example:

```None
spec:
  on:
    - targetRef:
        kind: Inbound
        labels:
          port-name: http-port
---
# select MeshExternalService
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
```

With this in place we can go even further and get rid of `to` section in policies and use `on` for selecting all resources like MeshService and Mesh*Route. For example:

```
None
# this will apply configutation on outbounds to MS
spec:
  on:
    - targetRef:
        kind: MeshService
        name: demo-app
---
# this will apply configutation on outbound routes selected by targetRef
spec:
  on:
    - targetRef:
        kind: MeshHTTPRoute
        name: demo-reoute
```

This approach can also accommodate inbound routes in the future if we decide to introduce them.

Using `on` keyword for selecting real resources feels more natural than using separate keywords like `from` and `to`. At least for selecting routes, users are `applying configuration on` route not `to route`.

Now with MeshService being destination users should clearly know that they are configuring outbounds, when you select Inbound you know that inbound will be configured. Unfortunately it is not that clear for MES as it can be configured on both dpp and zone egress. However if you make sure that configuration is applied on specific workload you can select it with topLevel targetRef by selecting dpp or zone proxy.

Introduce “sectionRef” keyword to select ephemeral resource

This approach is similar to previous one, but because we would like to keep the rule that `targetRef` can only select real resource and we don’t want to introduce new resource for Inbound we could create new selector `sectionRef` which will select specific part of the resource. For example:

```
None
targetRef:
  kind: MeshSubset
  tags:
    app: demo-app
spec:
  on:
    - sectionRef:
        kind: InboundPort
        value: 8080
```

Ideas

1. “on[].port: <value>” + “on[].targetRef”
2. “on[].targetRef” and introduce new “kind: Dataplane”
3. introduce “sectionRef” to reference “ephemeral” resources (resources that don’t exist in the cluster)

1. **As a** service owner **I want** to expose 1 application port without mTLS **so that** external clients (such as health checkers) can use it without being added to the mesh. (A)

```
None
kind: MeshTLS
spec:
  targetRef:
    kind: MeshSubset
    tags:
      app: backend
  on:
    - port: 443
      from:
        - targetRef:
            kind: Mesh
          default:
            mode: Strict
    - port: 80
      from:
        - targetRef:
            kind: Mesh
          default:
```

```
          mode: Permissive
```

2. **As a** service owner **I want** to attach an inbound policy to a specific set of conditions (path, headers...) on an inbound route **so that** I can apply different policies depending on endpoints.

This should be handled in the “inbound routes” MADR.

3. **As a** service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

Requires inbound route functionality, should be handled in the “inbound routes” MADR.

4. **As a** mesh operator **I want** to grant/revoke access to MeshExternalService for a group of workloads **so that** I can implement POLP (principle of least privilege). (B)

todo: check if today we can use MeshExternalService without “allow-all” MTP?

None
```
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: mtp-for-mes
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: priviliged-client
          default:
            action: Allow
```

5. **As a** mesh operator **I want** to use MeshTrafficPermission with MeshExternalServices with autoreachable services feature enabled **so that** I can improve the sidecars performance. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              httpbin: allow
          default:
            action: Allow
```

6. **As a** mesh operator **I want** to apply rate limiting to the service exposed to the mesh only as a MeshExternalService **so that** I can protect the service from being overloaded. (B)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: kuma-system
  name: mrl-for-mes
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      default: ...
```

7. **As a** mesh operator **I want** to configure the TLS version and ciphers on ZoneEgress as it terminates the connection for traffic to MeshExternalServices **so that** I can fulfill the security requirements. (B)

Updating TLS version and ciphers is possible only when both “targetRef.kind” and “from[].targetRef.kind” are “Mesh”. Adding functionality to apply MeshTLS on zone egress should be in the same MADR as applying MeshTLS on delegated gateway.

8. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshExternalService **so that** I can see outgoing requests and their statuses in logs. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mes
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      default:
         backends:
           - type: File
             file:
               path: "/dev/stdout"
```

9. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshService **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef: {} # some way to target zone proxy
  to:
    - targetRef:
        kind: MeshService
        labels:
          kuma.io/display-name: backend
          k8s.kuma.io/namespace: backend-ns
       default:
         backends:
           - type: File
             file:
               path: "/dev/stdout"
```

10. **As a** mesh operator **I want** to enable access logging on ZoneIngress to specific MeshService **so that** I can see outgoing connections and traffic in logs (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-ingress
spec:
  targetRef:
    kind: Mesh
    proxyTypes: Ingress
  to:
    - targetRef:
        kind: MeshService
        name: demo-app
        namespace: kuma-demo
      default: ...
```

11. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshServices **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
  targetRef:
    kind: Mesh
    proxyTypes:
      - ZoneEgress
      - ZoneIngress
  on:
    - targetRef:
        kind: MeshService
       default:
         backends:
           - type: File
             file:
               path: "/dev/stdout"
```

12. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshExternalServices **so that** I can see outgoing connections and traffic in logs. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-mltiple-mes
spec:
  targetRef:
    kind: Mesh
    proxyTypes: [ZoneIngress, ZoneEgress]
  on:
    - targetRef:
        kind: MeshExternalService
        labels:
          logging: enabled
      default: ...
---
# simplified
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-mltiple-mes
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        labels:
          logging: enabled
      default: ...
```

13. **As a** mesh operator **I want** to use MeshMetric policy on ZoneEgress/ZoneIngress the same way I do on Sidecars **so that** I can configure metric profiles (what metrics to expose). (C)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshMetric
metadata:
  name: metrics-default
  namespace: kuma-system
  labels:
    kuma.io/mesh: default
spec:
  targetRef:
    kind: Mesh
    proxyTypes:
      - ZoneIngress
      - ZoneEgress
  default:
    backends:
    - type: Prometheus
      prometheus:
        port: 5670
        path: "/metrics"
```

14. **As a** mesh operator **I want** to use MeshTrace policy on ZoneEgress the same way I do on Sidecars **so that** I can configure trace publishing on ZoneEgress for MeshExternalServices that operate over the HTTP. (C)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrace
metadata:
  namespace: kuma-system
  name: trace-egress
spec:
  targetRef:
    kind: Mesh
    proxyTypes: ZoneEgress
  on:
    - targetRef: MeshExternalService
      name: my-external-service
  default: ...
```

15. **As a** mesh operator **I want** to use the MeshProxyPatch policy on ZoneEgress/ZoneIngress **so that** I can patch the Envoy configuration when Kuma CP is not capable of doing so. (C)

todo: do we need to change MeshProxyPatch policy to support zone proxies?

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshProxyPatch
metadata:
  name: custom-template-1
  namespace: kuma-system
spec:
  targetRef:
    kind: Mesh
    proxyTypes:
     - ZoneEgress
  default:
    appendModifications: [...]
```

16. **As a** mesh operator **I want** to change timeouts on ZoneEgress to specific MeshExternalService (or to the HTTP route of MES) **so that** I can increase the timeout value if the default is too small for the use case. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: Mesh
    proxyTypes: ZoneEgress
  on:
    - targetRef: MeshExternalService
      name: my-external-service
  default: ...
---
# on route
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: Mesh
    proxyTypes: ZoneEgress
  on:
    - targetRef: MeshHttpRoute
      name: my-external-route
  default: ...
```

17. **As a** mesh operator **I want** to simulate faulty behaviour of a MeshExternalService **so that** I can perform chaos testing on the system. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshFaultInjection
metadata:
  name: faults-on-httpbin
  namespace: kuma-system
spec:
 on:
  - targetRef:
      kind: MeshExternalService
      name: httpbin
    default:
      http:
      - abort:
          httpStatus: 500
          percentage: 50
```

18. **As a** mesh operator **I want** to change timeouts on ZoneEgress/ZoneIngress to specific MeshService **so that** I can increase the timeout value if the default is too small for the use case. (D)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: Mesh
    proxyTypes: ZoneEgress
  to:
    - targetRef: MeshService
      name: demo-app
      namespace: kuma-demo
  default: ...
```

19. **As** a service owner **I want** to allow access to my service for a subset of traffic based on the Spiffe ID of the client workload **so that** the client authentication is happening based on the traffic attributed rather than client deployment attributes (such as tags).

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: spiffe-based
spec:
  on:
    - port: 8080
      from:
        - targetRef:
            kind: SpiffeID
            name: spiffe://orders/frontend
          default:
            action: Allow
```

New model

1. **As a** service owner **I want** to expose 1 application port without mTLS **so that** external clients (such as health checkers) can use it without being added to the mesh. (A)

```None
kind: MeshTLS
spec:
  targetRef:
    kind: MeshSubset
    tags:
      app: backend
  on:
    - port: 443
      from:
        - targetRef:
            kind: Mesh
          default:
            mode: Strict
    - port: 80
      from:
        - targetRef:
            kind: Mesh
          default:
            mode: Permissive
```

2. **As a** service owner **I want** to attach an inbound policy to a specific set of conditions (path, headers...) on an inbound route **so that** I can apply different policies depending on endpoints.

```None
kind: MeshAccessLog
spec:
  rules:
    - matches:
        method: GET
      default: ...
```

3. **As a** service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

Requires inbound route functionality, should be handled in the “inbound routes” MADR.

4. **As a** mesh operator **I want** to grant/revoke access to MeshExternalService for a group of workloads **so that** I can implement POLP (principle of least privilege). (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: mtp-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - matches:
            - spiffeID: ...
          default:
            action: Allow
```

5. **As a** mesh operator **I want** to use MeshTrafficPermission with MeshExternalServices with autoreachable services feature enabled **so that** I can improve the sidecars performance. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
spec:
```

6. **As a** mesh operator **I want** to apply rate limiting to the service exposed to the mesh only as a MeshExternalService **so that** I can protect the service from being overloaded. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: kuma-system
  name: mrl-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - default: ...
```

7. **As a** mesh operator **I want** to configure the TLS version and ciphers on ZoneEgress as it terminates the connection for traffic to MeshExternalServices **so that** I can fulfill the security requirements. (B)

Updating TLS version and ciphers is possible only when both “targetRef.kind” and “from[].targetRef.kind” are “Mesh”. Adding functionality to apply MeshTLS on zone egress should be in the same MADR as applying MeshTLS on delegated gateway.

8. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshExternalService **so that** I can see outgoing requests and their statuses in logs. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
      rules:
        - default: ...
```

9. **As a** mesh operator **I want** to enable access logging on ZoneEgress to specific MeshService **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
```

10. **As a** mesh operator **I want** to enable access logging on ZoneIngress to specific MeshService **so that** I can see outgoing connections and traffic in logs (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-ingress
spec:
  targetRef:
    kind: ZoneIngress
  to:
    - targetRef:
        kind: MeshService
        name: demo-app
        namespace: kuma-demo
      rules:
        - default: ...
```

11. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshServices **so that** I can see outgoing connections and traffic in logs. (D)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-for-ms
spec:
```

12. **As a** mesh operator **I want** to enable access logging on ZoneIngress/ZoneEgress to a group of MeshExternalServices **so that** I can see outgoing connections and traffic in logs. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshAccessLog
metadata:
  namespace: kuma-system
  name: log-mltiple-mes
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        labels:
          access-logging: enabled
      rules:
        - default: ...
```

13. **As a** mesh operator **I want** to use MeshMetric policy on ZoneEgress/ZoneIngress the same way I do on Sidecars **so that** I can configure metric profiles (what metrics to expose). (C)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshMetric
metadata:
  name: metrics-default
  namespace: kuma-system
  labels:
    kuma.io/mesh: default
spec:
```

14. **As a** mesh operator **I want** to use MeshTrace policy on ZoneEgress the same way I do on Sidecars **so that** I can configure trace publishing on ZoneEgress for MeshExternalServices that operate over the HTTP. (C)

```None
apiVersion: kuma.io/v1alpha1
```

```
kind: MeshTrace
metadata:
  namespace: kuma-system
  name: trace-egress
spec:
  targetRef:
    kind: ZoneEgress
  default: ...
```

15. **As a** mesh operator **I want** to use the MeshProxyPatch policy on ZoneEgress/ZoneIngress **so that** I can patch the Envoy configuration when Kuma CP is not capable of doing so. ©

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshProxyPatch
metadata:
  name: custom-template-1
  namespace: kuma-system
spec:
```

16. **As a** mesh operator **I want** to change timeouts on ZoneEgress to specific MeshExternalService **so that** I can increase the timeout value if the default is too small for the use case. (B, E)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: my-external-service
```

```
      rules:
        - default: ...
```

17. **As a** mesh operator **I want** to simulate faulty behaviour of a MeshExternalService **so that** I can perform chaos testing on the system. (B)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshFaultInjection
metadata:
  name: faults-on-httpbin
  namespace: kuma-system
spec:
```

18. **As a** mesh operator **I want** to change timeouts on ZoneEgress/ZoneIngress to specific MeshService **so that** I can increase the timeout value if the default is too small for the use case. (D)

```None
apiVersion: kuma.io/v1alpha1
kind: MeshTimeout
metadata:
  namespace: kuma-system
  name: timeout-egress
spec:
  targetRef:
    kind: ZoneIngess
  to:
    - targetRef:
        kind: MeshService
        name: demo-app
        namespace: kuma-demo
      rules:
        - default: ...
```

19. **As** a service owner **I want** to allow access to my service for a subset of traffic based on the Spiffe ID of the client workload **so that** the client authentication is happening based on the traffic attributed rather than client deployment attributes (such as tags).

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshTrafficPermission
metadata:
  namespace: kuma-system
  name: spiffe-based
spec:
```

Appendices

As a service owner I want to create a stub endpoint with a static response so that I can enable early adopters to start using the API and test the integration between client and server (with the real networking and permissions). (E)

As a service owner I want to rate limit an API endpoint so that I can prevent server overload. (E)

As a service owner I want to give access to "/health" to everyone and to all other endpoints only to a group of workloads so that infra monitoring services could check the health of my service. (E)

As a service owner I want to exclude the "/health" endpoint from access logging so that the noisy endpoint doesn't pollute logs. (E)

As a service owner I want to remove malicious HTTP headers on the server-side when there is a known vulnerability in my service so that I can prevent an attack (i.e. log4j vulnerability) in situations when there are non-mesh clients or there is a risk of clients overriding my "producer" routes. (E)

As a service owner I want to change the timeout for the specific route on the server-side so that I could prevent connection leaks due to abusive non-mesh clients or if there is a risk of client overriding my "producer" timeouts (E)

As a service owner I want to simulate faulty behaviour of a specific HTTP route of the service I'm running so that I can perform chaos testing of the clients. (E)

As a service owner **I want** to have 2 groups of clients – low priority and high priority and specify different rate limits for these groups of clients **so that** server resources are managed efficiently. (E)

Backup

```None
Inbound route IR1 with PathRewrite /lp/orders -> /orders
Inbound route IR2 with PathRewrite /hp/orders -> /orders

MeshRateLimit MRL1 on IR1 with quota 1 rps
MeshRateLimit MRL2 on IR2 with quota 100 rps

Producer MeshHTTPRoute MHR1 for LP clients with PathRewrite /orders ->
/lp/orders
Producer MeshHTTPRoute MHR2 for HP clients with PathRewrite /orders ->
/hp/orders

Producer MeshRetry MR1 for MHR1 with backoff to not retry too often as the
quota is low.
Producer MeshRetry MR2 for MHR2 with smaller backoff as the quota is higher.

MeshTrafficPermission MTP1 on IR1 to allow requests only from LP clients
MeshTrafficPermission MTP2 on IR2 to allow requests only from HP clients
```

```None
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: MeshSubset
    tags:
      app: backend
  from:
    - targetRef:
        kind: MeshSubset
        tags:
          app: frontend
      default:
        action: Allow
---
spec:
  on:
    - targetRef:
```

```yaml
        kind: Dataplane
        labels:
          app: backend
        sectionName: http
      targetRef:
        sectionName: http
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: frontend
          default:
            action: Allow
---
spec:
  targetRef:
    kind: MeshSubset
    labels:
      app: backend
  on:
    - targetRef:
        sectionName: http
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: frontend
          default:
            action: Allow
---
spec:
  targetRef:
    kind: MeshSubset
    labels:
      app: backend
  on:
    - targetRef:
        kind: Port
        name: http
      from:
    - port/sectionName:
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: frontend
          default:
            action: Allow
---
spec:
  on:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: frontend
          default:
            action: Allow
---
spec:
  on:
    - sectionName: ""
      targetRef: {}
      from:
        - targetRef:
            kind: MeshSubset
            tags:
              app: frontend
          default:
            action: Allow
```

Idea 1: mutually exclusive “sectionName” and “targetRef” inside “spec.on[]”
Idea 2: always use “targetRef” inside “spec.on[]” (introduce new kind Dataplane)

```
None
apiVersion: kuma.io/v1alpha1
kind: MeshHTTPRoute
metadata:
  namespace: backend-ns
  name: compute-to-hp-compute
  labels:
    kuma.io/policy-role: producer
```

```
spec:
  targetRef:
    kind: MeshSubset
    tags:
      backend-access: priority
  to:
    - targetRef:
        kind: MeshService
        name: backend
      rules:
        - matches:
          - path:
              type: PathPrefix
              value: "/compute"
          default:
            filter:
              type: URLRewrite
              urlRewrite:
                path: "/hp/compute"
---
apiVersion: kuma.io/v1alpha1
kind: MeshInboundHTTPRoute
metadata:
  namespace: backend-ns
  name: hp-compute-to-compute
  labels:
    kuma.io/policy-role: consumer
spec:
  on:
    - port: 8080
      rules:
        - matches:
          - path:
              type: PathPrefix
              value: "/hp/compute"
          default:
            filter:
              type: URLRewrite
              urlRewrite:
                path: "/compute"
---
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: backend-ns
  name: high-priority
spec:
  on:
    - targetRef:
        kind: MeshInboundHTTPRoute
        name: compute
      from:
        targetRef:
          kind: Mesh
        default:
          local:
            http:
              requestRate:
                num: 100
                interval: 1s
---
apiVersion: kuma.io/v1alpha1
kind: MeshRateLimit
metadata:
  namespace: backend-ns
  name: default
spec:
  on:
    - port: 8080
      from:
        targetRef:
          kind: Mesh
        default:
          local:
            http:
              requestRate:
                num: 10
                interval: 1s
```

```None
kind: Inbound
metadata:
  name: $app + $port
  labels: $pod_labels (+ $service_labels)?
spec:
  port: 8080
status:
  meshServices:
    - name: backend
      namespace: backend-ns
---
kind: Dataplane # MeshProxyConfig
spec:
  transparentProxying: {}
  adminPort: 9901
---
kind: MeshTrafficPermission
spec:
  on:
    - targetRef:
        kind: Inbound
        labels:
```

1 inbound port -> multiple services
generating "kuma.io/service" tag per inbound hides this fact

We have to stop generating tags for inbounds and instead we should generate dataplane labels. Also, we have to add Dataplane status.

```None
kind: Dataplane
metadata:
  labels: $pod_labels
spec:
  inbound:
    - port: 8080
      name: api
    - port: 9090
      name: admin
status:
  meshServices:
   - name: backend
     namespace: backend-ns
     zone: zone-1
```

```None
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: Dataplane
    labels:
      app.kubernetes.io/name: backend
    sectionName: api
  rules:
    - targetRef:
        kind: Dataplane
        labels:
          app.kubernetes.io/name: frontend
       default:
         action: Allow
---
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: MeshSubset
    labels:
      kuma.io/proxy-type: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      from:
        - targetRef:
```

```yaml
            kind: Mesh
          default:
            action: Allow
---
kind: MeshTrafficPermission
spec:
  targetRef:
    kind: MeshSubset
    labels:
      kuma.io/proxy-type: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      rules:
        - default:
            action: Allow
---
kind: MeshAccessLog
spec:
  targetRef:
    kind: ZoneIngress
  to:
    - targetRef:
        kind: MeshService
        name: backend
        namespace: backend-ns
      default:
        action: Allow
      rules:
        - targetRef:
            kind: Mesh
          default:
            action: Allow
---
kind: MeshTrafficPermission
spec:
  selector:
    labels:
      kuma.io/proxy-type: ZoneIngress
    sectionName: non-existing-section-name
  from:
    - targetRef:
        kind: Workload | ServiceAccount | SpiffeID | JWT
        labels:
          app.kubernetes.io/name: frontend
       default:
         action: Allow
---
kind: MeshTimeout
spec:
  targetRef:
    kind: MeshSubset
    labels:
      kuma.io/proxy-type: ZoneEgress
  to:
    - targetRef:
        kind: MeshExternalService
        name: httpbin
      default:
        connectTimeout: 5s
---
kind: MeshTimeout
spec:
  targetRef:
    kind: MeshSubset
  to:
    - targetRef:
        kind: MeshInboundHTTPRoute | MeshHTTPRoute
        name: httpbin
      rules:
        - targetRef:
            kind: Mesh
          default:
            action: Allow
```

None

```
kind: Workload
spec:
  ports:
    - port: 8080
      name: http-port
      appProtocol: http
      targetPort: 8081
    - port: 9090
```

```
      name: tcp-port
      appProtocol: http
      targetPort: 9091
status:
  meshServices:
    -
```

1. Remove “spec.networking.inbound[].tags” and use “metadata.labels” instead
2. Create “status” for Dataplane

```None
spec:
  rules: []
  to:
    - targetRef:
        kind: MeshService | MeshExternalService | MeshMultiZoneService
      default: {}
      rules:
        - matches: []
          targetRef: {}
          default: {}
```