Traffic Director includes *route rules* and *traffic policies* that enable you to control traffic within your Traffic Director deployment. These capabilities are based on xDS API-compatible sidecar proxies, such as Envoy. They are enabled using the load balancing components URL maps, forwarding rules, and backend services.

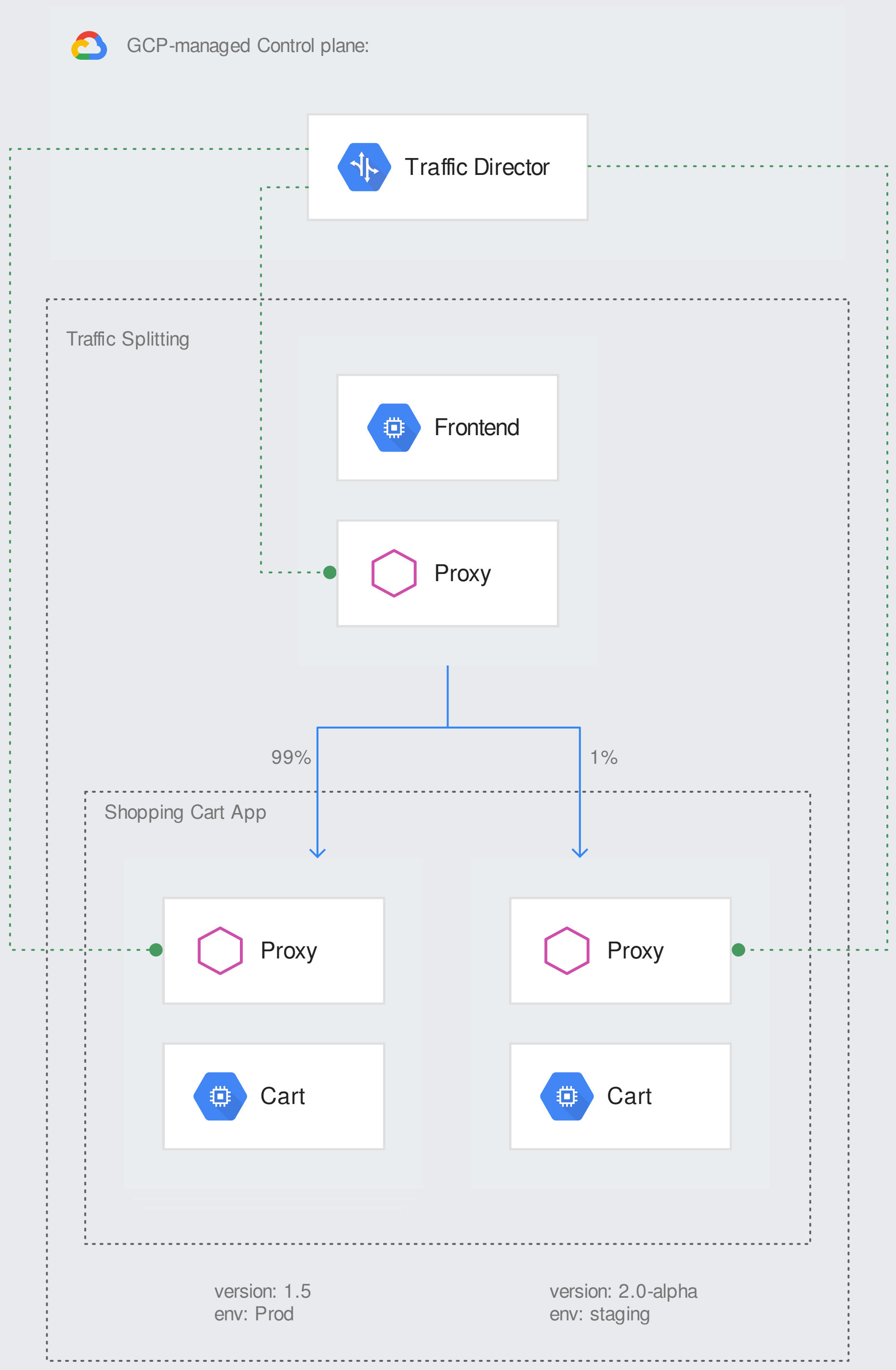
A route rule matches information you specify to an incoming request and makes a routing decision based on the match. Route rules are executed according to priority.

After traffic is routed, traffic policies take further action to manage your traffic. Traffic policies generally belong to these categories:

- Load balancer settings, which control the load balancing algorithm
- Circuit breakers, which control the volume of connections to an upstream service
- Outlier detection, which controls the eviction of unhealthy hosts from the load balancing pool

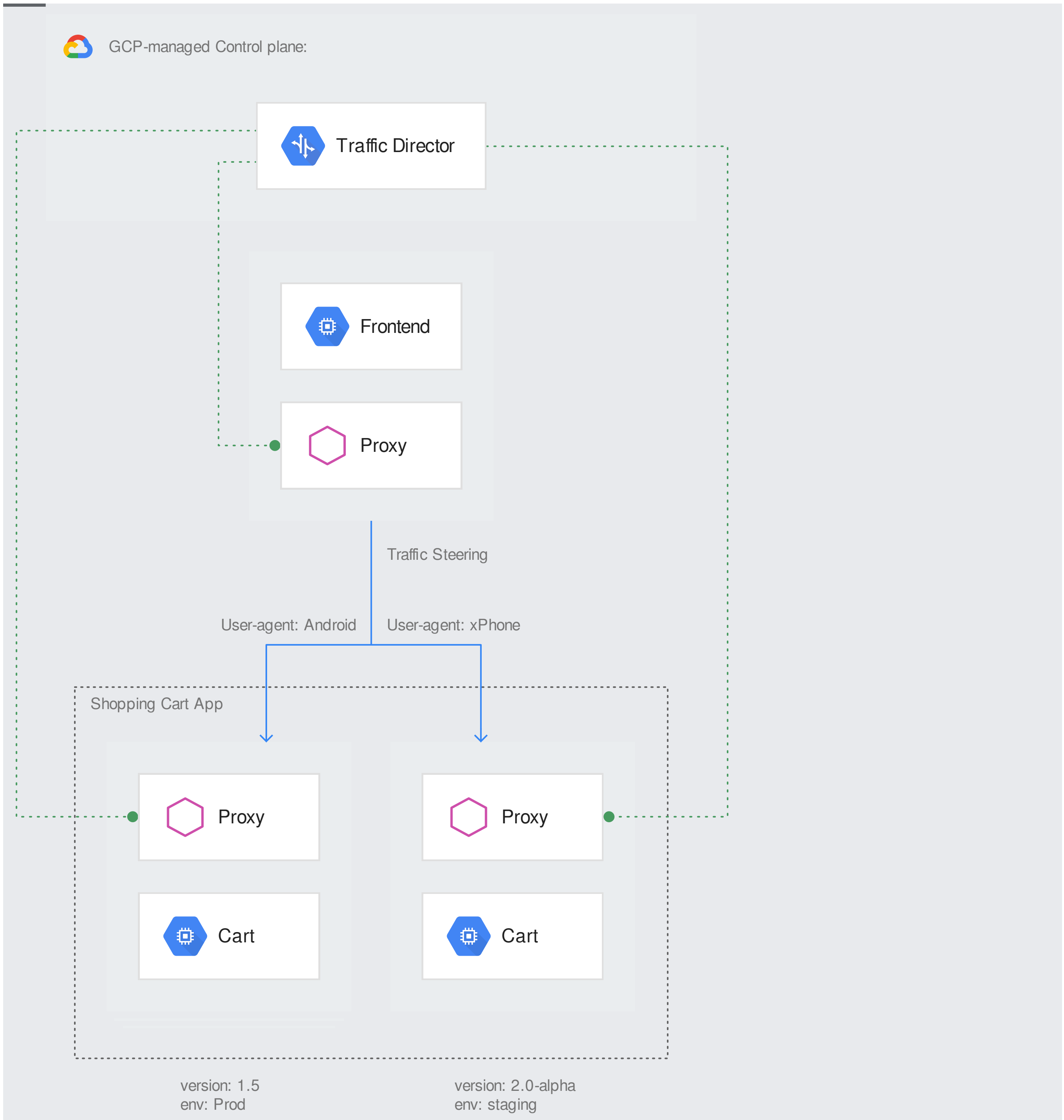
Route rules and traffic policies let you implement *traffic splitting*, *traffic steering*, *fault injection*, *circuit breaking*, and *mirroring*.

Traffic splitting splits traffic based on weight. For example, you can send 99% of traffic to one service instance and 1% to another service instance. Traffic splitting is typically used for deploying new versions, A/B testing, service migration, and similar processes.



(/traffic-director/images/td-traffic-splitting.svg)
Traffic Director traffic splitting (click to enlarge)

Traffic steering directs traffic to service instances based on content in HTTP request headers. For example, if a user's device is an Android device, with `user-agent:Android` in the request header, that traffic is sent to service instances designated to receive Android traffic, and traffic that does not have `user-agent:Android` is sent to instances that handle other devices.



(/traffic-director/images/td-traffic-steering.svg)
Traffic Director traffic steering (click to enlarge)

Fault injection enables you to test the resiliency of services by simulating service failure scenarios, such as delays and aborted requests. *Delay injection* configures the client side proxy to introduce delays for a fraction of requests before sending the request to the selected backend service. *Abort injection* configures the sidecar proxy to directly respond to a fraction of requests with failure response codes, rather than forwarding those requests to the backend service.

Circuit breaking enforces limits on requests to a particular service, such as the maximum number of connections or maximum number requests after which requests are prevented from reaching the service to protect the service from degrading further.

Mirroring is implemented by deploying a debug application that receives a copy of your real traffic. The production application processes the original traffic, while the debug application discards responses. You can use mirroring to test binaries with production traffic and to debug productions errors on the debug application rather than on your production application.

Route rules provide several ways to match HTTP requests and perform various actions such as traffic splitting across several backend services, responding with redirects and altering requests or responses. Route rules are configured using the URL map (/compute/docs/reference/rest/v1/urlMaps).

- Match rules enable Traffic Director to match one or more attributes of a request and take actions specified in the route rule. The following attributes of a request can be used to specify the matching criteria:
    - Host: A host name is the domain name portion of a URL; for example, the host name portion of the URL `http://example.net/video/hd` is `example.net`.
    - Paths are the part of the URL that follows the hostname; for example `/images` in `http://service-host-name/images`. The rule can specify whether the entire path or only the leading portion of the path must match. The rule can alternatively supply a regular expression that must match the path.
    - Other HTTP request parameters such as HTTP headers, which allows cookie matching, as well as matching based on query parameters (GET variables).
    - Metadata filters: These are opaque match criteria for proxies conforming to Envoy's xDS protocol. For example, sidecar proxies making xDS requests may supply node metadata such as application version, staged deployment version, and other criteria. Traffic Director supplies route configuration only to those sidecar proxies when settings in metadata filters match corresponding settings in the node metadata supplied by the proxy.
- Route actions configure Traffic Director with specific actions to take when a route rule matches the attributes of a request. Use the following advanced route actions:

- Redirects: Traffic Director returns a configurable 3xx response code. It also sets the `Location` response header with the appropriate URI, replacing the host and path as specified in the redirect action.
- URL rewrites: Traffic Director can re-write the host name portion of the URL, the path portion of the URL, or both, before sending a request to the selected backend service.
- Header transformations: Traffic Director can add or remove request headers before sending a request to the backend service. It can also add or remove response headers after receiving a response from the backend service. You configure header transformations at various levels of the URL map, including its top level, at a path matcher, in a route rule, and in a weighted backend service. This allows you to specify hierarchical control of headers.
- Traffic mirroring: In addition to forwarding the request to the selected backend service, Traffic Director sends an identical request to the configured mirror backend service on a "fire and forget" (https://www.envoyproxy.io/docs/envoy/latest/api-v2/api/v2/route/route.proto#route-routeaction-requestmirrorpolicy) basis. This capability is useful for logging requests.
- Weighted traffic splitting is a configuration that allows traffic for a matched rule to be distributed to multiple backend services, proportional to a user-defined weight assigned to the individual backend service. This capability is useful for configuring staged deployments or A/B testing. For example, the route action could be configured such that 99% of the traffic is sent to a service that's running a stable version of an application, while 1% of traffic is sent to a separate service running a newer version of that application.
- Retries configure the conditions under which Traffic Director retries failed requests, how long it waits before retrying and the maximum number of retries permitted.
- Fault injection: Traffic Director can intentionally introduce errors when servicing requests to simulate failures, including high latency, service overload, service failures, and network partitioning. This feature is useful for testing the resiliency of a service to simulated faults.
  - Delay injection configures the proxy to introduce delays for a user-defined portion of requests before sending the request to the selected backend service.
  - Abort injection configures the proxy to respond directly to a fraction of requests with user-defined HTTP status codes instead of forwarding those requests to

the backend service.

- Security policies: Cross-origin resource sharing(CORS) (https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) policies handle Traffic Director settings for enforcing CORS requests.

Route rules are executed in priority order, providing you with flexibility in associating match rules with actions.

For route rules within a given path matcher, the priority of each rule determines the order in which the load balancer interprets the route rules. The route rules are evaluated in the order of priority, from the lowest to the highest number. The priority of a rule decreases as its number increases (1, 2, 3, *N*+1). The first rule that matches a request is applied. After the first match is made, Traffic Director stops evaluating the rules and any remaining rules are ignored. For example, if you have four rules with priorities 2, 16, 23, and 45, and the first match is to the rule with priority 16, the rules with priority 23 and priority 45 are ignored.

You cannot configure two or more route rules with the same priority. You must set the priority for each rule to a number from 0 through 2147483647.

Priority numbers can have gaps. This allows you to add or remove rules in the future without affecting any existing rules. For example, you might have rules with priority 1, 2, 3, 4, 5, 9, 12, and 16. This is a valid series, to which you can add rules with the values 6, 7, 8, 11, 13, 14, and 15. The new rules that you add do not have any impact on the existing rules.

You must provide a priority for each route rule. Creating or updating a route rule requires that you provide a value for the priority field. Existing route rules that you do not update continue to work even though the priority field does not have a value.

Traffic policies are groups of settings that define how load balancing behaves, including the response to failing backend services and how to prevent localized failures from affecting the rest of the service mesh. The following traffic policy features are configured in the backend service (/compute/docs/reference/rest/v1/backendServices).

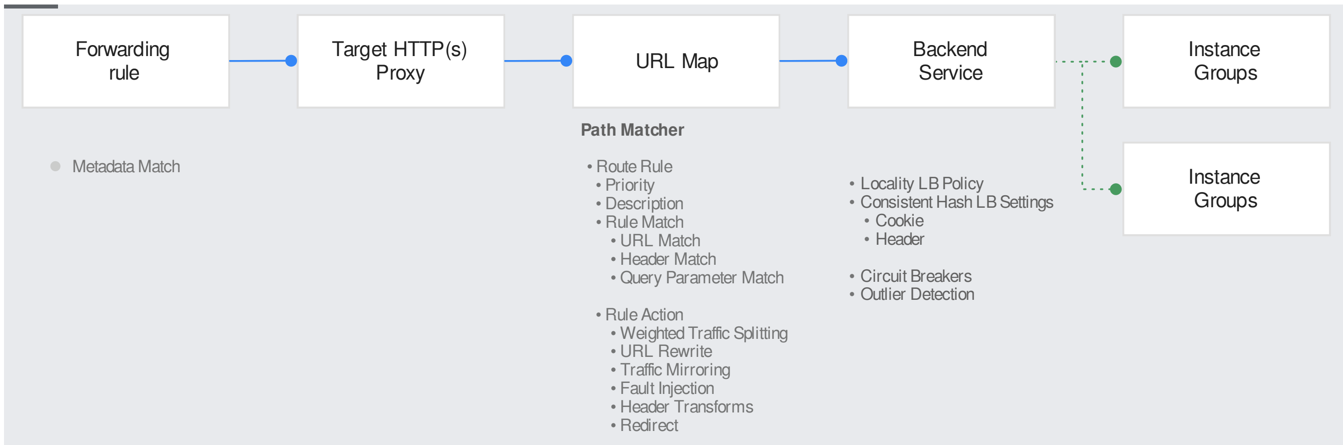
- Load balancing policy: Traffic Director performs global load balancing based on available capacity, and, as stated in the Envoy documentation (https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/upstream/load_balancing/localit y_weight.html) , sets the locality level (per Google Cloud zone) load balancing weights, for the proxy to choose a particular Google Cloud zone for the backend VMs or endpoints in NEGs. The load balancing policies specified in the backend service further determine the algorithm used for backend VMs or endpoints in NEGs within the locality to be chosen after a weighted locality choice. Load balancing algorithms include round robin, ring hash, and backends VMs or endpoints in NEGs with the least request (https://www.envoyproxy.io/docs/envoy/latest/api-v2/api/v2/cds.proto#envoy-api-enum-cluster-lbpolicy) . Note that in Traffic Director, global load balancing is performed using URL maps, forwarding rules, and backend services.

- Session affinity: Traffic Director offers several session affinity options: HTTP cookie-based affinity, HTTP header-based affinity, client IP address affinity, and generated cookie affinity. Note that affinity settings are not honored if more than one `weightedBackendServices` is included in `routeAction`. For more information on session affinity, read Session affinity (/load-balancing/docs/https/#session_affinity).

- Outlier detection is a set of policies that specify the criteria for eviction of unhealthy backend VMs or endpoints in NEGs, along with criteria defining when a backend or endpoint is considered healthy enough to receive traffic again.

- Circuit breaking sets upper limits on the volume of connections and requests per connection to a backend service.

Three existing Google Cloud resources are used to implement capabilities such as advanced route and traffic policies. The following diagram shows which resources are used to implement each feature:



(/traffic-director/images/td-data-model.svg)
Traffic Director data model (click to enlarge)

Traffic Director uses a forwarding rule whose load balancing scheme is INTERNAL_SELF_MANAGED. The forwarding rule intercepts traffic on a configurable RFC 1918 IP address, then sends it to a target HTTP proxy and URL map. The forwarding rule resource supports advanced traffic management with selective configuration of proxies. Traffic Director supplies route configuration to only those proxies that satisfy criteria specified in the forwarding rule's metadata filters.

You use metadata filters to push configurations to a subset of sidecar proxies. If no metadata filter is in place, all proxies receive all of the configurations associated with the VPC network. Using metadata filters gives you finer control of the scope of the forwarding rule, and therefore greater control over which URL map and backend service define traffic distribution.

This is useful for the following scenarios:

1. To comply with your network requirements, multiple projects use the same Shared VPC network, but each service project wants its Traffic Director resources to be visible *only* to proxies in the same project.
2. The data plane is partitioned and each proxy needs to know about only a subset of configurations in the network.
3. You are introducing a new configuration and you need to test it on a subset of proxies before you make it available to all proxies.

For information on how to configure metadata filters, see Setting up config filtering based on `MetadataFilter` match
(/traffic-director/docs/configure-advanced-traffic-management#config-filtering-metadata).

The URL map component is extended to support advanced routing features in the path matcher. You can match requests based on path prefix, full path, path regex, HTTP header, and query parameter. The prefix, full path, and regex all apply to the path portion of the match. For more information, read the Envoy proxy RouteMatch documentation
(https://www.envoyproxy.io/docs/envoy/latest/api-v2/api/v2/route/route.proto#envoy-api-msg-route-routematch)
.

- Path matcher: Traffic Director extends the URL map's path matcher concept. You can continue to use simple *path rules* as you would for a GCP HTTP(S) load balancer, or you can use *route rules* instead. Use the two types of rules exclusively. Path rules are evaluated on the longest-path-matches-first basis and these rules can be specified in any order. Route rules are interpreted in order. This allows greater flexibility in defining complex route criteria.
  - Route rules (HttpRouteRules): Traffic Director evaluates route rules in priority order. The following are the components of a route rule:
    - Priority: A number from 0 to 2147483647 inclusive assigned to a rule within a given path matcher that determines the order in which the load balancer interprets the route rules. The priority of a rule decreases as its number increases, so that a rule with the number 4 takes priority over a rule with the number 25. The first rule that matches the request is applied. Priority numbers can have gaps. You cannot create two or more rules with the same priority.
    - Description: An optional description of up to 1024 characters.
    - Route rule match (HttpRouteRuleMatch): Allows you to determine if the route rule applies to a request by matching all or a subset of the request's attributes such as the path, HTTP headers, and query (GET) parameters. Within an HttpRouteRuleMatch, all matching criteria must be met for the rule's actions to take effect. If a rule has multiple HttpRouteRuleMatches, the actions of the rule take effect when a request matches any of the rule's HttpRouteRuleMatches.

    - Route action (HttpRouteAction): Allows you to specify what actions Traffic Director should take when the criteria within HttpRouteRuleMatch are met. These actions include traffic splitting, URL rewrites, retry and mirroring, fault injection, and CORS policies.
    - Redirect action (HttpRedirectAction): You can configure Traffic Director to respond with an HTTP redirect when the criteria within HttpRouteRuleMatch are met.
    - Header action (HttpHeaderAction): You can configure request and response header transformation rules when the criteria within HttpRouteRuleMatch are met.
    - Metadata filters (MetadataFilters): You can specify criteria for which xDS compliant proxies get routing configuration tied to a route rule.
  - Path rules (PathRule): Traffic Director supports path rule objects in path matchers to maintain compatibility with existing URL maps. You can enter path rules in any order. Traffic Director tries to match the request's path to each of the paths from all path rules within that path matcher, on the longest-path-matches-first basis. If a path match occurs, traffic is routed to the backend service of the corresponding path rule.

The URL map hierarchy for advanced route features is as follows:

- Default backend service or default backend bucket
- Traffic Director only: default route action
- Traffic Director only: default redirect
- List of host rules
- List of path matchers, each path matcher containing
  - Traffic Director only: default route action
  - Traffic Director only: default redirect
  - Default backend service or default backend bucket for path matcher
  - List of path rules
  - Traffic Director only: list of route rules
    - Traffic Director only: list of match rules

- Traffic Director only: route action
- Traffic Director only: redirect
- Traffic Director only: header action

Traffic Director uses a backend service whose load balancing scheme is INTERNAL_SELF_MANAGED. A backend service with this scheme supports the settings that implement the bulk of the traffic policies. The following attributes can be specified for an internal self-managed backend service:

- Load balancing policy (LocalityLoadBalancingPolicy): For an internal self-managed backend service, traffic distribution is accomplished by using a combination of a *load balancing mode* and a *load balancing policy*. The backend service first directs traffic to a backend (instance group or NEG) according to the backend's balancing mode, then, once a backend has been selected, Traffic Director distributes traffic according to the load balancing policy. The balancing mode allows Traffic Director to first select a locality, such as a Google Cloud zone; then, the load balancing policy is used to select a specific backend VM or endpoint in a NEG.
- Session affinity (SessionAffinity): Internal self-managed backend services support four session affinities: client IP address, HTTP cookie-based, HTTP header-based, and generated cookie (generated by Traffic Director itself) affinity.
- Consistent hash (ConsistentHashLoadBalancerSettings) define criteria for building consistent hashes from cookies and headers for Traffic Director to consistently route new requests to the same backend VMs or endpoints in NEGs.
- Circuit breakers (CircuitBreakers) define parameters for limiting the volume of traffic to any particular backend VM or endpoint of the backend service. This prevents overloading services with requests that they cannot handle in a meaningful way.
- Outlier detection (OutlierDetection) defines criteria for determining when a backend VM or endpoint in a NEG is deemed unhealthy and excluded from load balancing considerations, as well as the conditions that must be satisfied to reconsider the backend VM or endpoints for load balancing. A backend VM or endpoint is deemed unhealthy when the health check linked to the backend service marks the backend VM or endpoint in a NEG as unhealthy.

See Configuring advanced traffic management (/traffic-director/docs/configure-advanced-traffic-management) for information on how to set up advanced traffic management.