Traefik HTTP Services Documentation

- Service Load Balancer
  - Configuration Example
  - Configuration Options
    - Servers
      - Configuration Options
  - Load Balancing Strategies
    - Weighted Round Robin (wrr)
    - Power of Two Choices (p2c)
    - Highest Random Weight (hrw)
    - Least-Time
  - Health Check
  - Sticky Sessions
    - Stickiness on multiple levels
    - Stickiness & Unhealthy Servers
    - Cookie Name
    - MaxAge
    - Secure & HTTPOnly & SameSite flags
    - Domain
  - Passive Health Check
  - Middlewares
- Advanced Service Types
  - Weighted Round robin
    - Health Check
  - Highest Random Weight
    - Health Check
  - Mirroring

    - Health Check
- Failover
    - HealthCheck
    - Errors
    - Chaining Failover Services

Traefik services define how to distribute incoming traffic across your backend servers. This page covers two main concepts:

- **Service Load Balancer**: Routes traffic to backend servers using various load balancing strategies
- **Advanced Service Types**: Compose multiple services together for weighted distribution, mirroring, or failover

Service Load Balancer

The `loadBalancer` service type routes incoming requests to a list of backend servers. Each service has a load-balancer, even if there is only one server to forward traffic to.

The load balancer supports multiple **strategies** for distributing traffic among servers:

- `wrr` (Weighted Round Robin) - Default strategy, distributes requests evenly across servers in rotation
- `p2c` (Power of Two Choices) - Selects two random servers and routes to the one with fewer active connections
- `hrw` (Highest Random Weight) - Uses consistent hashing based on client IP for session affinity
- `leasttime` - Routes to the server with lowest response time combined with fewest active connections

Configuration Example

```
http:
  services:
    my-service:
      loadBalancer:
        strategy: "wrr"
        servers:
          - url: "http://private-ip-server-1/"
```

```
            weight: 2
            preservePath: true
        sticky:
          cookie:
            name: "sticky-cookie"
        healthcheck:
          path: "/health"
          interval: "10s"
          timeout: "3s"
        passiveHealthCheck:
          failureWindow: "3s"
          maxFailedAttempts: 3
        passHostHeader: true
        serversTransport: "customTransport@file"
        responseForwarding:
          flushInterval: "150ms"
```

```
[http.services]
  [http.services.my-service.loadBalancer]
    strategy = "wrr"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-1/"

    [http.services.my-service.loadBalancer.sticky.cookie]
      name = "sticky-cookie"

    [http.services.my-service.loadBalancer.healthcheck]
      path = "/health"
      interval = "10s"
      timeout = "3s"

    [http.services.my-service.loadBalancer.passiveHealthCheck]
      failureWindow = "3s"
      maxFailedAttempts = 3

    passHostHeader = true
    serversTransport = "customTransport@file"

    [http.services.my-service.loadBalancer.responseForwarding]
      flushInterval = "150ms"
```

```
labels:
  - "traefik.http.services.my-service.loadBalancer.strategy=wrr"
  - "traefik.http.services.my-service.loadBalancer.servers[0].url=http://private-ip-server-1/"
  - "traefik.http.services.my-service.loadBalancer.servers[0].weight=2"
  - "traefik.http.services.my-service.loadBalancer.servers[0].preservePath=true"
  - "traefik.http.services.my-service.loadBalancer.sticky.cookie.name=sticky-cookie"
  - "traefik.http.services.my-service.loadBalancer.healthcheck.path=/health"
  - "traefik.http.services.my-service.loadBalancer.healthcheck.interval=10s"
  - "traefik.http.services.my-service.loadBalancer.healthcheck.timeout=3s"
  - "traefik.http.services.my-service.loadBalancer.passiveHealthCheck.failureWindow=3s"
  - "traefik.http.services.my-service.loadBalancer.passiveHealthCheck.maxFailedAttempts=3"
  - "traefik.http.services.my-service.loadBalancer.passHostHeader=true"
  - "traefik.http.services.my-service.loadBalancer.serversTransport=customTransport@file"
  - "traefik.http.services.my-service.loadBalancer.responseForwarding.flushInterval=150ms"
```

```
{
  "Tags": [
    "traefik.http.services.my-service.loadBalancer.strategy=wrr",
    "traefik.http.services.my-service.loadBalancer.servers[0].url=http://private-ip-server-
1/",
    "traefik.http.services.my-service.loadBalancer.servers[0].weight=2",
    "traefik.http.services.my-service.loadBalancer.servers[0].preservePath=true",
    "traefik.http.services.my-service.loadBalancer.sticky.cookie.name=sticky-cookie",
    "traefik.http.services.my-service.loadBalancer.healthcheck.path=/health",
    "traefik.http.services.my-service.loadBalancer.healthcheck.interval=10s",
    "traefik.http.services.my-service.loadBalancer.healthcheck.timeout=3s",
    "traefik.http.services.my-service.loadBalancer.passiveHealthCheck.failureWindow=3s",
    "traefik.http.services.my-service.loadBalancer.passiveHealthCheck.maxFailedAttempts=3",
    "traefik.http.services.my-service.loadBalancer.passHostHeader=true",
    "traefik.http.services.my-service.loadBalancer.serversTransport=customTransport@file",
    "traefik.http.services.my-service.loadBalancer.responseForwarding.flushInterval=150ms"
  ]
}
```

Configuration Options

Field	Description	Required
servers	Represents individual backend instances for your service	Yes

Field	Description	Required
`strategy`	Load balancing strategy for distributing traffic among servers. Valid values: `wrr` (default), `p2c`, `hrw`, `leasttime`.	No
`sticky`	Defines a `Set-Cookie` header is set on the initial response to let the client know which server handles the first response.	No
`healthcheck`	Configures health check to remove unhealthy servers from the load balancing rotation.	No
`passiveHealthCheck`	Configures the passive health check to remove unhealthy servers from the load balancing rotation.	No
`passHostHeader`	Allows forwarding of the client Host header to server. By default, `passHostHeader` is true.	No
`serversTransport`	Allows to reference an HTTP ServersTransport configuration for the communication between Traefik and your servers. If no `serversTransport` is specified, the `default@internal` will be used.	No
`responseForwarding`	Configures how Traefik forwards the response from the backend server to the client.	No
`responseForwarding.flushInterval`	Specifies the interval in between flushes to the client while copying the response body. It is a duration in milliseconds, defaulting to 100ms. A negative value means to flush immediately after each write to the client. The `FlushInterval` is ignored when ReverseProxy recognizes a response as a streaming response; for such responses, writes are flushed to the client immediately.	No

Servers

Servers represent individual backend instances for your service. The service loadBalancer `servers` option lets you configure the list of instances that will handle incoming requests.

Configuration Options

Field	Description	Required
`url`	Points to a specific instance.	Yes for File provider, No for Docker provider
`weight`	Allows for weighted load balancing on the servers.	No
`preservePath`	Allows to preserve the URL path.	No

Load Balancing Strategies

The `strategy` option on the load balancer determines how traffic is distributed among the backend servers.

Weighted Round Robin (wrr)

The default strategy. Distributes requests evenly across all servers in rotation, respecting server weights. This strategy uses Earliest Deadline First (EDF) scheduling to provide weighted round-robin behavior.

??? example "WRR Load Balancing -- Using the File Provider"

```
```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        strategy: "wrr"
        servers:
        - url: "http://private-ip-server-1/"
          weight: 3
        - url: "http://private-ip-server-2/"
          weight: 1
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service.loadBalancer]
    strategy = "wrr"
```
```

```
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-1/"
      weight = 3
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-2/"
      weight = 1
```

Power of Two Choices (p2c)

Selects two servers at random and routes the request to the one with the fewest active connections. This algorithm provides better load distribution when servers have varying response times.

??? example "P2C Load Balancing -- Using the File Provider"

```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        strategy: "p2c"
        servers:
        - url: "http://private-ip-server-1/"
        - url: "http://private-ip-server-2/"
        - url: "http://private-ip-server-3/"
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service.loadBalancer]
    strategy = "p2c"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-1/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-2/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-3/"
```

```
```

Highest Random Weight (hrw)

Uses consistent hashing (Rendezvous Hashing) based on the client's IP address to ensure requests from the same client are consistently routed to the same server. This provides session affinity without requiring sticky cookies.

The algorithm computes a score for each available backend using a hash of the client's source IP combined with the backend's identifier, and assigns the client to the backend with the highest score.

??? example "HRW Load Balancing -- Using the File Provider"

```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        strategy: "hrw"
        servers:
        - url: "http://private-ip-server-1/"
        - url: "http://private-ip-server-2/"
        - url: "http://private-ip-server-3/"
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service.loadBalancer]
    strategy = "hrw"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-1/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-2/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-3/"
```

Least-Time

Selects the server with the lowest average response time (Time To First Byte - TTFB), combined with the fewest active connections, weighted by server capacity. This strategy is ideal for heterogeneous backend environments where servers have varying performance characteristics, different hardware capabilities, or varying network latency.

The algorithm continuously measures each backend's response time and tracks active connection counts. When routing a request, it calculates a score for each healthy server using the formula: `(avg_response_time × (1 + active_connections)) / weight`. The server with the lowest score receives the request. When multiple servers have identical scores, Weighted Round Robin (WRR) with Earliest Deadline First (EDF) scheduling is used as a tie-breaker.

??? example "Least-Time Load Balancing -- Using the File Provider"

````
```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        strategy: "leasttime"
        servers:
        - url: "http://private-ip-server-1/"
        - url: "http://private-ip-server-2/"
        - url: "http://private-ip-server-3/"
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service.loadBalancer]
    strategy = "leasttime"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-1/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-2/"
    [[http.services.my-service.loadBalancer.servers]]
      url = "http://private-ip-server-3/"
```
````

Health Check

The `healthcheck` option configures health check to remove unhealthy servers from the load balancing rotation. Traefik will consider HTTP(s) servers healthy as long as they return a status code to the health check request (carried out every interval) between `2XX` and `3XX`, or matching the configured status. For gRPC servers, Traefik will consider them healthy as long as they return SERVING to gRPC health check v1 requests.

To propagate status changes (e.g. all servers of this service are down) upwards, HealthCheck must also be enabled on the parent(s) of this service.

Below are the available options for the health check mechanism:

Field	Description	Default	Required
`path`	Defines the server URL path for the health check endpoint. The configured path must be relative URL.	""	Yes
`scheme`	Replaces the server URL scheme for the health check endpoint.		No
`mode`	If defined to `grpc`, will use the gRPC health check protocol to probe the server.	http	No
`hostname`	Defines the value of hostname in the Host header of the health check request.	""	No
`port`	Replaces the server URL port for the health check endpoint.		No
`interval`	Defines the frequency of the health check calls for healthy targets.	30s	No
`unhealthyInterval`	Defines the frequency of the health check calls for unhealthy targets. When not defined, it defaults to the `interval` value.	-	No
`timeout`	Defines the maximum duration Traefik will wait for a health check request before considering the server unhealthy.	5s	No
`headers`	Defines custom headers to be sent to the health check endpoint.		No

Field	Description	Default	Required
`followRedirects`	Defines whether redirects should be followed during the health check calls.	true	No
`method`	Defines the HTTP method that will be used while connecting to the endpoint.	GET	No
`status`	Defines the expected HTTP status code of the response to the health check request.		No

Sticky Sessions

When sticky sessions are enabled, a `Set-Cookie` header is set on the initial response to let the client know which server handles the first response. On subsequent requests, to keep the session alive with the same server, the client should send the cookie with the value set.

Stickiness on multiple levels

When chaining or mixing load-balancers (e.g. a load-balancer of servers is one of the "children" of a load-balancer of services), for stickiness to work all the way, the option needs to be specified at all required levels. Which means the client needs to send a cookie with as many key/value pairs as there are sticky levels.

Stickiness & Unhealthy Servers

If the server specified in the cookie becomes unhealthy, the request will be forwarded to a new server (and the cookie will keep track of the new server).

Cookie Name

The default cookie name is an abbreviation of a sha1 (ex: `_1d52e`).

MaxAge

By default, the affinity cookie will never expire as the `MaxAge` option is set to zero.

This option indicates the number of seconds until the cookie expires.
When set to a negative number, the cookie expires immediately.

Secure & HTTPOnly & SameSite flags

By default, the affinity cookie is created without those flags. One however can change that through configuration.

`SameSite` can be `none`, `lax`, `strict` or empty. Values are case-insensitive.

Domain

The Domain attribute of a cookie specifies the domain for which the cookie is valid.

By setting the Domain attribute, the cookie can be shared across subdomains (for example, a cookie set for example.com would be accessible to www.example.com, api.example.com, etc.). This is particularly useful in cases where sticky sessions span multiple subdomains, ensuring that the session is maintained even when the client interacts with different parts of the infrastructure.

??? example "Adding Stickiness -- Using the File Provider"

````
```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        sticky:
         cookie: {}
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service]
    [http.services.my-service.loadBalancer.sticky.cookie]
```
````

??? example "Adding Stickiness with custom Options -- Using the File Provider"

````
```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    my-service:
      loadBalancer:
        sticky:
```
````

```
          cookie:
            name: my_sticky_cookie_name
            secure: true
            domain: mysite.site
            httpOnly: true
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.my-service]
    [http.services.my-service.loadBalancer.sticky.cookie]
      name = "my_sticky_cookie_name"
      secure = true
      httpOnly = true
      domain = "mysite.site"
      sameSite = "none"
```

??? example "Setting Stickiness on all the required levels -- Using the File Provider"

```yaml tab="Structured (YAML)"
## Routing configuration
http:
  services:
    wrr1:
      weighted:
        sticky:
          cookie:
            name: lvl1
        services:
          - name: whoami1
            weight: 1
          - name: whoami2
            weight: 1

    whoami1:
      loadBalancer:
        sticky:
          cookie:
```

```
            name: lvl2
        servers:
          - url: http://127.0.0.1:8081
          - url: http://127.0.0.1:8082

    whoami2:
      loadBalancer:
        sticky:
          cookie:
            name: lvl2
        servers:
          - url: http://127.0.0.1:8083
          - url: http://127.0.0.1:8084
```

```toml tab="Structured (TOML)"
## Routing configuration
[http.services]
  [http.services.wrr1]
    [http.services.wrr1.weighted.sticky.cookie]
      name = "lvl1"
    [[http.services.wrr1.weighted.services]]
      name = "whoami1"
      weight = 1
    [[http.services.wrr1.weighted.services]]
      name = "whoami2"
      weight = 1

  [http.services.whoami1]
    [http.services.whoami1.loadBalancer]
      [http.services.whoami1.loadBalancer.sticky.cookie]
        name = "lvl2"
      [[http.services.whoami1.loadBalancer.servers]]
        url = "http://127.0.0.1:8081"
      [[http.services.whoami1.loadBalancer.servers]]
        url = "http://127.0.0.1:8082"

  [http.services.whoami2]
    [http.services.whoami2.loadBalancer]
      [http.services.whoami2.loadBalancer.sticky.cookie]
```

```
        name = "lvl2"
      [[http.services.whoami2.loadBalancer.servers]]
        url = "http://127.0.0.1:8083"
      [[http.services.whoami2.loadBalancer.servers]]
        url = "http://127.0.0.1:8084"
```

To keep a session open with the same server, the client would then need to specify the two levels within the cookie for each request, e.g. with curl:

```
curl -b "lvl1=whoami1; lvl2=http://127.0.0.1:8081" http://localhost:8000
```

Passive Health Check

The `passiveHealthCheck` option configures passive health check to remove unhealthy servers from the load balancing rotation.

Passive health checks rely on real traffic to assess server health. Traefik forwards requests as usual and evaluates each response or timeout, incrementing a failure counter whenever a request fails. If the number of successive failures within a specified time window exceeds the configured threshold, Traefik will automatically stop routing traffic to that server until it recovers. A server will be considered healthy again after the configured failure window has passed.

Below are the available options for the passive health check mechanism:

Field	Description	Default	Required
`failureWindow`	Defines the time window during which the failed attempts must occur for the server to be marked as unhealthy. It also defines for how long the server will be considered unhealthy.	10s	No
`maxFailedAttempts`	Defines the number of consecutive failed attempts allowed within the failure window before marking the server as unhealthy.	1	No

Middlewares

You can attach a list of middlewares to each HTTP service. The middlewares will take effect for all requests handled by the service, regardless of which router forwards the request.

!!! info "Middlewares Execution Order"

```
When both a router and a service have middlewares configured, the router middlewares are
applied first, followed by the service middlewares.
This means the request passes through router middlewares before reaching service middlewares.
```

!!! info "Supported Providers"

```
Service-level middlewares can be configured with the [File](../../../install-
configuration/providers/others/file.md), [Docker](../../other-providers/docker.md),
[Swarm](../../other-providers/docker.md), [Kubernetes
IngressRoute](../../kubernetes/crd/http/ingressroute.md), [Kubernetes
Ingress](../../kubernetes/ingress.md), and [Kubernetes Gateway API](../../kubernetes/gateway-
api.md) providers.
```

??? example "Attaching Middlewares to a Service -- Using the File Provider"

```
```yaml tab="Structured (YAML)"
## Dynamic configuration
http:
  services:
    my-service:
      middlewares:
        - add-header
      loadBalancer:
        servers:
          - url: "http://127.0.0.1:8080"

  middlewares:
    add-header:
      headers:
        customRequestHeaders:
          X-Custom-Header: "service-middleware"
```

```toml tab="Structured (TOML)"
## Dynamic configuration
[http.services]
  [http.services.my-service]
    middlewares = ["add-header"]
```
```

```
    [http.services.my-service.loadBalancer]
      [[http.services.my-service.loadBalancer.servers]]
        url = "http://127.0.0.1:8080"

[http.middlewares]
  [http.middlewares.add-header.headers]
    [http.middlewares.add-header.headers.customRequestHeaders]
      X-Custom-Header = "service-middleware"
```

??? example "Attaching Middlewares to a Service -- Using Docker Labels"

```yaml
labels:
  # Define the middleware
  - "traefik.http.middlewares.add-header.headers.customRequestHeaders.X-Custom-Header=service-middleware"
  # Attach middleware to the service (at service level, not loadBalancer level)
  - "traefik.http.services.my-service.middlewares=add-header"
  # Configure the service
  - "traefik.http.services.my-service.loadbalancer.server.port=8080"
```

Advanced Service Types

Advanced service types allow you to compose multiple services together for weighted distribution, consistent hashing, mirroring, or failover scenarios. These are distinct from load balancing strategies - they operate at the **service level** rather than the **server level**.

!!! info "Key Difference"

```
- **Load Balancing Strategies** (wrr, p2c, hrw, leasttime): Distribute traffic among **servers** within a single `loadBalancer` service
- **Advanced Service Types** (weighted, highestRandomWeight, mirroring, failover): Distribute or manage traffic among multiple **services**
```

Weighted Round robin

The `weighted` service type load balances requests between multiple services based on weights. This is different from the `wrr` strategy - it operates on services, not servers.

!!! info "Supported Providers"

```
This service type can be defined currently with the [File](../../../install-configuration/providers/others/file.md) provider or [IngressRoute](../../../routing-configuration/kubernetes/crd/http/ingressroute.md).
```

```
## Routing configuration
http:
  services:
    app:
      weighted:
        services:
        - name: appv1
          weight: 3
        - name: appv2
          weight: 1

    appv1:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-1/"

    appv2:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
  [http.services.app]
    [[http.services.app.weighted.services]]
      name = "appv1"
      weight = 3
    [[http.services.app.weighted.services]]
      name = "appv2"
      weight = 1
```

```
  [http.services.appv1]
    [http.services.appv1.loadBalancer]
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Health Check

HealthCheck enables automatic self-healthcheck for this service, i.e. whenever one of its children is reported as down, this service becomes aware of it, and takes it into account (i.e. it ignores the down child) when running the load-balancing algorithm. In addition, if the parent of this service also has HealthCheck enabled, this service reports to its parent any status change.

!!! note "Behavior"

```
If HealthCheck is enabled for a given service and any of its descendants does not have it
enabled, the creation of the service will fail.

HealthCheck on Weighted services can be defined currently only with the [File
provider](../../../install-configuration/providers/others/file.md).
```

```
## Routing configuration
http:
  services:
    app:
      weighted:
        healthCheck: {}
        services:
        - name: appv1
          weight: 3
        - name: appv2
          weight: 1

    appv1:
      loadBalancer:
        healthCheck:
          path: /status
```

```
          interval: 10s
          timeout: 3s
        servers:
        - url: "http://private-ip-server-1/"

    appv2:
      loadBalancer:
        healthCheck:
          path: /status
          interval: 10s
          timeout: 3s
        servers:
        - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
  [http.services.app]
    [http.services.app.weighted.healthCheck]
    [[http.services.app.weighted.services]]
      name = "appv1"
      weight = 3
    [[http.services.app.weighted.services]]
      name = "appv2"
      weight = 1

  [http.services.appv1]
    [http.services.appv1.loadBalancer]
      [http.services.appv1.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [http.services.appv2.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
```

```
        timeout = "3s"
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Highest Random Weight

The `highestRandomWeight` service type uses consistent hashing (Rendezvous Hashing) to load balance requests between multiple services. This ensures that requests from the same client IP are consistently routed to the same service.

This is different from the `hrw` strategy on a loadBalancer - it operates on **services**, not servers.

!!! info "Supported Providers"

```
This service type can be defined currently only with the [File](../../../install-configuration/providers/others/file.md) provider.
```

```
## Routing configuration
http:
  services:
    app:
      highestRandomWeight:
        services:
        - name: appv1
          weight: 1
        - name: appv2
          weight: 1

    appv1:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-1/"

    appv2:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
```

```
  [http.services.app]
    [[http.services.app.highestRandomWeight.services]]
      name = "appv1"
      weight = 1
    [[http.services.app.highestRandomWeight.services]]
      name = "appv2"
      weight = 1

  [http.services.appv1]
    [http.services.appv1.loadBalancer]
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Health Check

HealthCheck enables automatic self-healthcheck for this service, similar to the Weighted Round Robin service type.

!!! note "Behavior"

```
If HealthCheck is enabled for a given service and any of its descendants does not have it enabled, the creation of the service will fail.

HealthCheck on Highest Random Weight services can be defined currently only with the [File provider](../../../install-configuration/providers/others/file.md).
```

```
## Routing configuration
http:
  services:
    app:
      highestRandomWeight:
        healthCheck: {}
        services:
        - name: appv1
          weight: 1
        - name: appv2
```

```
        weight: 1

  appv1:
    loadBalancer:
      healthCheck:
        path: /status
        interval: 10s
        timeout: 3s
      servers:
      - url: "http://private-ip-server-1/"

  appv2:
    loadBalancer:
      healthCheck:
        path: /status
        interval: 10s
        timeout: 3s
      servers:
      - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
  [http.services.app]
    [http.services.app.highestRandomWeight.healthCheck]
    [[http.services.app.highestRandomWeight.services]]
      name = "appv1"
      weight = 1
    [[http.services.app.highestRandomWeight.services]]
      name = "appv2"
      weight = 1

  [http.services.appv1]
    [http.services.appv1.loadBalancer]
      [http.services.appv1.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"
```

```
  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [http.services.appv2.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Mirroring

The `mirroring` service type mirrors requests sent to a service to other services. Please note that by default the whole request is buffered in memory while it is being mirrored. See the `maxBodySize` option in the example below for how to modify this behaviour. You can also omit the request body by setting the `mirrorBody` option to false.

To avoid buffering the request body, set the `streamBody` option to true: the mirrored requests are sent along with the main request, and the body is streamed to the mirrors while the main service reads it. A mirror falling more than 1 MiB behind the main service is aborted, so that it never slows the main request down, and a mirrored request is aborted when the main service does not read the whole body. The `streamBody` option has no effect when `mirrorBody` is false, and cannot be used with `maxBodySize`.

!!! warning "Default behavior of `percent`"

```
When configuring a `mirror` service, if the `percent` field is not set, it defaults to `0`,
meaning **no traffic will be sent to the mirror**.
```

!!! info "Supported Providers"

```
This service type can be defined currently with the [File](../../../install-
configuration/providers/others/file.md) provider or [IngressRoute](../../../routing-
configuration/kubernetes/crd/http/ingressroute.md).
```

```
## Routing configuration
http:
  services:
    mirrored-api:
      mirroring:
        service: appv1
```

```
        # mirrorBody defines whether the request body should be mirrored.
        # Default value is true.
        mirrorBody: false
        # maxBodySize is the maximum size allowed for the body of the request.
        # If the body is larger, the request is not mirrored.
        # Default value is -1, which means unlimited size.
        maxBodySize: 1024
        mirrors:
        - name: appv2
          # Percent defines the percentage of requests that should be mirrored.
          # Default value is 0, which means no traffic will be sent to the mirror.
          percent: 10

    appv1:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-1/"

    appv2:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-2/
```

```
## Routing configuration
[http.services]
  [http.services.mirrored-api]
    [http.services.mirrored-api.mirroring]
      service = "appv1"
      # maxBodySize is the maximum size in bytes allowed for the body of the request.
      # If the body is larger, the request is not mirrored.
      # Default value is -1, which means unlimited size.
      maxBodySize = 1024
      # mirrorBody defines whether the request body should be mirrored.
      # Default value is true.
      mirrorBody = false
    [[http.services.mirrored-api.mirroring.mirrors]]
      name = "appv2"
      percent = 10

  [http.services.appv1]
```

```
    [http.services.appv1.loadBalancer]
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Health Check

HealthCheck enables automatic self-healthcheck for this service, i.e. if the main handler of the service becomes unreachable, the information is propagated upwards to its parent.

!!! note "Behavior"

```
If HealthCheck is enabled for a given service and any of its descendants does not have it
enabled, the creation of the service will fail.

HealthCheck on Mirroring services can be defined currently only with the [File
provider](../../../install-configuration/providers/others/file.md).
```

```
## Routing configuration
http:
  services:
    mirrored-api:
      mirroring:
        healthCheck: {}
        service: appv1
        mirrors:
        - name: appv2
          percent: 10

    appv1:
      loadBalancer:
        healthCheck:
          path: /status
          interval: 10s
          timeout: 3s
        servers:
        - url: "http://private-ip-server-1/"
```

```
    appv2:
      loadBalancer:
        servers:
        - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
  [http.services.mirrored-api]
    [http.services.mirrored-api.mirroring]
      service = "appv1"
      [http.services.mirrored-api.mirroring.healthCheck]
    [[http.services.mirrored-api.mirroring.mirrors]]
      name = "appv2"
      percent = 10

  [http.services.appv1]
    [http.services.appv1.loadBalancer]
      [http.services.appv1.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.appv1.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.appv2]
    [http.services.appv2.loadBalancer]
      [http.services.appv2.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.appv2.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Failover

The `failover` service type forwards requests to a fallback service when the main service is unavailable. Failover can be triggered in two ways:

- **Health check-based**: When the main service becomes unreachable based on health checks.
- **Status code-based**: When the main service responds with specific HTTP status codes defined in the errors configuration.

!!! info "Relation to HealthCheck" The failover service relies on the HealthCheck system to get notified when its main service becomes unreachable, which means HealthCheck needs to be enabled and functional on the main service. However, HealthCheck does not need to be enabled on the failover service itself for it to be functional. It is only required in order to propagate upwards the information when the failover itself becomes down (i.e. both its main and its fallback are down too).

!!! info "Supported Providers" This service type can be defined with the File and Kubernetes CRD providers.

HealthCheck

HealthCheck enables automatic self-healthcheck for this service, i.e. if the main and the fallback services become unreachable, the information is propagated upwards to its parent.

!!! note "Behavior"

```
If HealthCheck is enabled for a given service and any of its descendants does not have it
enabled, the creation of the service will fail.

HealthCheck on a Failover service can be defined currently only with the [File
provider](../../../install-configuration/providers/others/file.md).
```

```
## Routing configuration
http:
  services:
    app:
      failover:
        healthCheck: {}
        service: main
        fallback: backup

    main:
      loadBalancer:
        healthCheck:
          path: /status
          interval: 10s
```

```
          timeout: 3s
        servers:
        - url: "http://private-ip-server-1/"

    backup:
      loadBalancer:
        healthCheck:
          path: /status
          interval: 10s
          timeout: 3s
        servers:
        - url: "http://private-ip-server-2/"
```

```
## Routing configuration
[http.services]
  [http.services.app]
    [http.services.app.failover]
      service = "main"
      fallback = "backup"
      [http.services.app.failover.healthCheck]

  [http.services.main]
    [http.services.main.loadBalancer]
      [http.services.main.loadBalancer.healthCheck]
        path = "/status"
        interval = "10s"
        timeout = "3s"
      [[http.services.main.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.backup]
    [http.services.backup.loadBalancer]
      [http.services.backup.loadBalancer.healthCheck]
        path = "/status"
        interval = "10s"
        timeout = "3s"
      [[http.services.backup.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Errors

The `errors` option enables status code-based failover. When the main service responds with an HTTP status code matching one of the configured ranges, Traefik automatically retries the request on the fallback service.

To support request replay, the request body is buffered up to `maxRequestBodyBytes`. Requests with bodies larger than this limit receive a `413 Request Entity Too Large` response.

Below is a list of options available for the `errors` option and an example of how to configure it for a failover service:

Field	Description	Default
`status`	List of HTTP status code ranges that trigger failover. Supports single codes (`"500"`) and ranges (`"500-504"`).	None
`maxRequestBodyBytes`	Maximum request body size (in bytes) to buffer for replay to the fallback service. Set to `-1` for no limit.	`-1`

```yaml
## Routing configuration
http:
  services:
    app:
      failover:
        service: main
        fallback: backup
        errors:
          status:
            - "500-504"
          maxRequestBodyBytes: 1048576

    main:
      loadBalancer:
        servers:
          - url: "http://private-ip-server-1/"

    backup:
      loadBalancer:
        servers:
          - url: "http://private-ip-server-2/"
```

```toml
## Routing configuration
[http.services]
```

```
  [http.services.app]
    [http.services.app.failover]
      service = "main"
      fallback = "backup"
      [http.services.app.failover.errors]
        status = ["500-504"]
        maxRequestBodyBytes = 1048576

  [http.services.main]
    [http.services.main.loadBalancer]
      [[http.services.main.loadBalancer.servers]]
        url = "http://private-ip-server-1/"

  [http.services.backup]
    [http.services.backup.loadBalancer]
      [[http.services.backup.loadBalancer.servers]]
        url = "http://private-ip-server-2/"
```

Chaining Failover Services

Failover services can be chained together for multi-level redundancy. In the following example, if the primary service fails, traffic goes to the secondary service. If both primary and secondary fail, traffic goes to the tertiary service.

```
## Routing configuration
http:
  services:
    app:
      failover:
        healthCheck: {}
        service: primary-failover
        fallback: tertiary

    primary-failover:
      failover:
        healthCheck: {}
        service: primary
        fallback: secondary

    primary:
```

```
      loadBalancer:
        healthCheck:
          path: /health
          interval: 10s
          timeout: 3s
        servers:
          - url: "http://primary-server/"

    secondary:
      loadBalancer:
        healthCheck:
          path: /health
          interval: 10s
          timeout: 3s
        servers:
          - url: "http://secondary-server/"

    tertiary:
      loadBalancer:
        healthCheck:
          path: /health
          interval: 10s
          timeout: 3s
        servers:
          - url: "http://tertiary-server/"
```

```
## Routing configuration
[http.services]
  [http.services.app]
    [http.services.app.failover]
      service = "primary-failover"
      fallback = "tertiary"
      [http.services.app.failover.healthCheck]

  [http.services.primary-failover]
    [http.services.primary-failover.failover]
      service = "primary"
      fallback = "secondary"
      [http.services.primary-failover.failover.healthCheck]

  [http.services.primary]
```

```
    [http.services.primary.loadBalancer]
      [http.services.primary.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.primary.loadBalancer.servers]]
        url = "http://primary-server/"

  [http.services.secondary]
    [http.services.secondary.loadBalancer]
      [http.services.secondary.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.secondary.loadBalancer.servers]]
        url = "http://secondary-server/"

  [http.services.tertiary]
    [http.services.tertiary.loadBalancer]
      [http.services.tertiary.loadBalancer.healthCheck]
        path = "/health"
        interval = "10s"
        timeout = "3s"
      [[http.services.tertiary.loadBalancer.servers]]
        url = "http://tertiary-server/"
```

□□□ #2
□ □□□ □□□ 2026-10-31 14:02:09 UTC
□ □□□ □□□ 2026-10-31 14:02:12 UTC