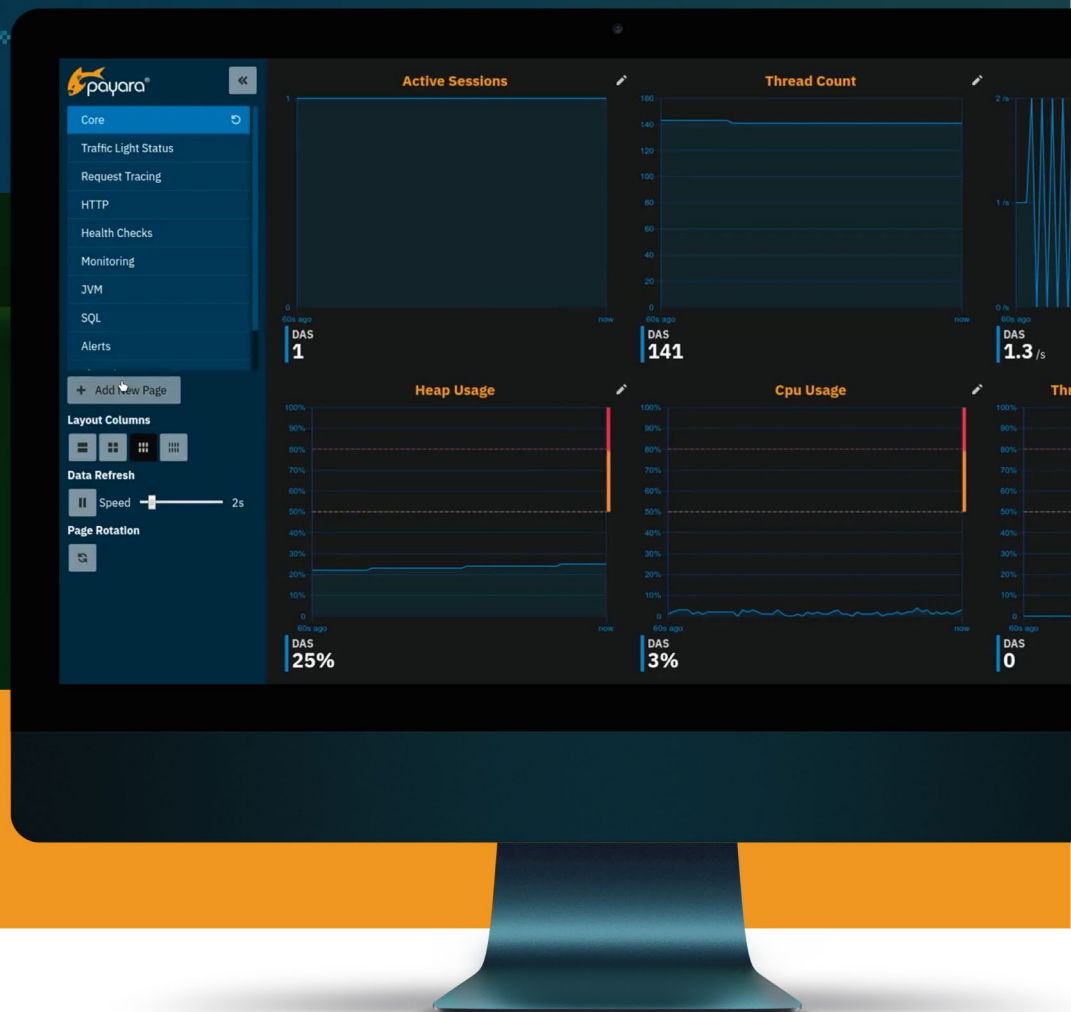




Payara Server Deployment: A Guide to Integrating with Nginx And Traefik



The Payara® Platform - Production-Ready,
Cloud Native and Aggressively Compatible.

User Guide

Contents

Guide Updated: **November 2023**

Reverse Proxy With Jakarta EE.....	1
Uses Of A Reverse Proxy.....	2
Load Balancing.....	2
Web Acceleration.....	2
Security and Anonymity.....	2
SSL Termination.....	2
Content Filtering.....	2
Why Use A Reverse Proxy?.....	3
Enhanced Security.....	3
Performance Gains.....	3
Scalability and Flexibility.....	3
Redundancy and Reliability.....	3
Control and Logging.....	3
Simplified Encryption Management.....	3
Types Of Reverse Proxy Servers.....	4
Nginx.....	4
Apache HTTP Server.....	4
HAProxy.....	4
Traefik.....	4
Squid.....	4
Caddy.....	4
Nginx As Reverse Proxy.....	5
Configuring Nginx.....	5
Configuring Nginx In Docker.....	7
Traefik Reverse Proxy.....	9
Traefik Components.....	9
Routers.....	9
Services.....	9
Middlewares.....	9
Load Balancer.....	9
Providers.....	10
Certificates.....	10
Traefik Configuration.....	10
Summary.....	13

In the realm of web architecture, reverse proxies play a critical role in improving security, manageability, and performance of deployment applications and services. Serving as an intermediary for requests from clients seeking resources from servers, reverse proxies can provide additional layers of security defence, handle load balancing, manage SSL provisioning and termination, and cache static content, thereby enhancing the overall efficiency and reliability of web applications served by specialised servers like Payara.

In this guide, we take a look at using Payara Server behind two popular reverse proxy servers - Nginx and Traefik. We start off with a brief discussion of what a reverse proxy is, and then move to see the various configuration options for using the two reverse proxy servers with a very typical Payara Server running both standalone and in docker.

Reverse Proxy

A reverse proxy is a type of server that retrieves resources on behalf of a client from one or more servers. These resources are then returned to the client as if they originated from the proxy server itself. Using a reverse proxy, the client does not know, or need to know the existence of the proxied server. To the client, the reverse proxy is the source of its requests. It is called "reverse" because it is the opposite of a [forward proxy](#), which protects clients by hiding their identities. Reverse proxies are a later evolution in web architecture, coming after the widespread use of forward proxies.

Reverse proxies play an important role in ensuring clients do not have direct, unfettered access to critical servers. Reverse proxies act as filters to requests from clients. Allowing through valid requests and blocking invalid requests based on custom configurations. Reverse proxies can also help reduce server load by caching resources and returning those to clients directly, instead of retrieving from the proxied servers.

Reverse Proxy With Jakarta EE

Reverse proxies can be used with a wide variety of servers. In the Jakarta EE ecosystem, reverse proxies are used to proxy application runtimes, such that the client makes a request to a http path such as `https://my-wicked-cool-company.com/amazing-app`. The amazing-app being a Jakarta EE app running on Payara Server would be proxied by a proxy server such that the client request, whether made from the browser or some REST client, goes to the proxy server, which then redirects the request to a Payara Server running instance, exposed internally through port 8080. The response is then returned to the client. In this example, the client is oblivious of the existence of Payara Server. The proxy acts as a messenger that takes the requests, gets a response and returns the same to the client.

Uses Of A Reverse Proxy

A reverse proxy can be used for a number of reasons. Depending on the business domain, the same reverse proxy can be configured to do different things depending on which server it's proxying for. Some of the uses of reverse proxies in typical application deployment scenarios include the following.

Load Balancing

It distributes incoming network traffic across multiple servers, ensuring no single server becomes overwhelmed. This can optimise resource use, maximise throughput, minimise response time, and avoid overload on any single resource. As an example, a reverse proxy can be configured to distribute incoming traffic to different Payara Server nodes in a given cluster.

Web Acceleration

Reverse proxies can compress inbound and outbound data, as well as cache commonly requested content, which can drastically decrease server load and improve the speed of content delivery to the client.

Security and Anonymity

Serving as the public face of your backend Payara servers, a reverse proxy protects against direct attacks. By hiding the characteristics and details of your servers, it helps to anonymize internal networks and safeguard sensitive data. This helps reduce the surface of possible attack as the actual server doing the heavy lifting is not directly exposed to the public facing internet.

SSL Termination

A proxy server can decrypt incoming requests and encrypt server responses so that the web servers do not have to perform these potentially resource-intensive tasks. This is known as [SSL offloading](#).

Content Filtering

Reverse proxies can restrict content based on incoming requests, which can be used for security, compliance, or to enforce company policies. For example, a reverse proxy can be configured to block requests from certain IP addresses.

Why Use A Reverse Proxy?

As discussed above, reverse proxies do play a critical role in production deployments of enterprise applications. Some of the core benefits of employing a reverse proxy for your Payara Server deployment include the following.

Enhanced Security

Can act as a shield for servers through the use of Web Application Firewalls to filter out malicious traffic, thus mitigating threats like SQL injections, cross-site scripting (XSS), and Distributed Denial-of-Service (DDoS) attacks.

Performance Gains

Caching, compressing, and SSL termination all reduce the load on the origin servers, enabling them to operate more efficiently. These are all tasks your Payara Server would not have to do, leaving it with enough resources to serve your application.

Scalability and Flexibility

They make it easier to scale out an application as traffic grows. Also, it can provide a way to load balance traffic between multiple servers or data centres. You can have your reverse proxy route traffic to different instances of your Payara Server nodes based on different criteria.

Redundancy and Reliability

If a server goes down, the reverse proxy can reroute traffic to other servers, ensuring that the application remains available.

Control and Logging

It can provide a central point of control and monitor all traffic for better insight and compliance reporting.

Simplified Encryption Management

By centralising encryption and decryption operations, a reverse proxy simplifies the management of SSL certificates.

Types Of Reverse Proxy Servers

There are several popular reverse proxy servers widely used in various environments, each with its own set of features and strengths.

Nginx

Known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. Nginx is often used for load balancing, SSL termination, and as a web server.

Apache HTTP Server

While primarily a web server, it can also be configured as a reverse proxy. It's known for its flexibility due to a vast number of modules that extend its functionality.

HAProxy

A reverse proxy that is particularly efficient in terms of CPU and memory usage, making it suitable for environments where resources are a concern. It specialises in high availability, load balancing, and proxying TCP and HTTP-based applications.

Traefik

A modern reverse proxy and load balancer designed to be easy to configure, especially in dynamic and container-centric environments. It integrates with popular orchestrators like Kubernetes, Docker, and others.

Squid

Originally a client-side cache proxy, Squid has evolved and now also operates as a reverse proxy. It is often used to cache static content, reducing the load on web servers.

Caddy

An open-source web server that includes automatic HTTPS among its features. It's known for its simplicity and zero-config SSL, which simplifies the setup of encrypted sites.

Each of these reverse proxy servers can be configured to improve performance, manage SSL/TLS, provide additional security layers, and much more. The choice often depends on the specific needs of your environment, such as the requirement for high availability, specific protocols, or integration

with other tools and systems. All of them can be used to reverse proxy Payara Server as well. The next sections discuss configuration options for using Nginx and Traefik as reverse proxies for Payara Server.

Nginx As Reverse Proxy

Using Nginx as a reverse proxy for Payara Server is a straightforward thing. The most important configuration file for Nginx is the `nginx.conf` file, located in `/etc/nginx` directory. This central configuration file is used to control the behaviour of the Nginx server. It can be structured into multiple contexts such as `events`, `http`, `server`, and `location`, each serving different configuration scopes from general to specific. The `nginx.conf` file can include other configuration files to organise settings for maintainability. Most of the time however, you would not need to make major changes to that file.

Configuring Nginx

Your custom reverse proxy configurations will most often be saved to the `/etc/nginx/conf.d/default.conf` file. This file is typically used to configure the default server block (virtual host) and is located within the `conf.d` directory, which is included in the main Nginx configuration. It is read by the main `nginx.conf` file when the Nginx service starts or reloads its configuration. The `default.conf` file is where you define server-specific directives, such as the listening port, server name, location blocks, and other settings pertinent to the server you want to proxy.

The following is a typical example of the `default.conf` file, delegating calls to a local running instance of Payara Server on port 8080.

```
# Reverse Proxy Config
server {
    listen 80; # Listen on port 80 for HTTP traffic
    server_name your-cool-domain-name.com;

    # Redirect all HTTP traffic to HTTPS by sending a 301 Moved
    Permanently response
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl;
    server_name your-cool-domain-name.com;
```

```
# Paths to SSL certificate and private key for HTTPS encryption
ssl_certificate /path/to/your/fullchain.pem;
ssl_certificate_key /path/to/your/privatekey.pem;

# SSL optimizations and security settings
ssl_session_cache shared:SSL:1m;
ssl_session_timeout 10m;
ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
SHA256:...';
ssl_prefer_server_ciphers on;

location / {
    proxy_pass http://localhost:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    # Enable WebSocket support
    proxy_http_version 1.1; # Use HTTP/1.1 for proxying
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
}
```

The above configuration describes two server blocks for Nginx:

The first server block is listening on port 80 (standard for HTTP) and is set up to redirect all HTTP requests to HTTPS. When a request is made to your-cool-domain-name.com using HTTP, it sends a 301 redirect to the client's browser, instructing it to request the page again using HTTPS.

The second server block is listening on port 443, which is the standard port for HTTPS traffic. It is configured to handle secure requests with SSL for your-cool-domain-name.com. It specifies the paths to the SSL certificate and key for encrypting the communication. It also defines SSL settings for security and performance optimization. The location / block sets up a reverse proxy to pass all requests to your Payara server running on localhost port 8080. It includes headers to pass the original host, the real client IP, and the scheme used (HTTP or HTTPS) to the Payara server. Additionally, it configures support for WebSocket connections.

Most of the time this configuration should suffice and should work once copied to the /etc/nginx/conf.d/ directory.

Configuring Nginx In Docker

When running both the reverse proxy and Payara Server as docker containers, the above configuration will only need to be changed slightly. Given the following sample docker compose file;

```
version: '3.9'

services:
  payara-server:
    image: payara/server-full:6.2023.10-jdk17
    ports:
      - "8080:8080"
    # Define environment variables or volumes for Payara

  nginx:
    image: nginx:latest
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./path/to/your/default.conf:/etc/nginx/conf.d/default.conf:ro
      - ./path/to/your/certs:/etc/nginx/certs:ro
    depends_on:
      - payara-server
```

The previously discussed default.conf file changes as follows;

```
# Reverse Proxy Config
server {
    listen 80; # Listen on port 80 for HTTP traffic
    server_name your-cool-domain-name.com;

    # Redirect all HTTP traffic to HTTPS by sending a 301 Moved
    Permanently response
    return 301 https://$server_name$request_uri;
}
```

```
server {
    listen 443 ssl;
    server_name your-cool-domain-name.com;

    # Paths to SSL certificate and private key for HTTPS encryption
    ssl_certificate /path/to/your/fullchain.pem;
    ssl_certificate_key /path/to/your/privatekey.pem;

    # SSL optimizations and security settings
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout 10m;
    ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
    SHA256:...';
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass http://payara-server:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Enable WebSocket support
        proxy_http_version 1.1; # Use HTTP/1.1 for proxying
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

The main change is the `proxy_pass` value. This time around we are proxying to the Payara Server service named `payara-server` in the docker-compose file. The `default.conf` file largely stays the same. The most important part of the docker compose file is mounting the `default.conf` file to the relevant folder in the Nginx container.

This is an example of how you would use Nginx as a reverse proxy for your Payara Server app. Of course, there is a lot more you can do, so do check out the full [Nginx documentation](#) for all the various options.

Traefik Reverse Proxy

Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy. Traefik integrates with your existing infrastructure components (Docker, Swarm mode, Kubernetes, Marathon, Consul, Etcd, Rancher, Amazon ECS) and configures itself automatically and dynamically. Pointing Traefik at your orchestrator should be the only configuration step you need.

It's designed to simplify the complexity of dynamic network infrastructure configurations and to handle a variety of network protocols. Traefik is particularly well-suited for environments that dynamically change, like when services are frequently deployed or have their network locations updated.

Traefik Components

Traefik is made up of various components that can be configured to create a reverse proxy. These components are as follows.

Routers

Routers are the entry points to Traefik; they define rules to evaluate each incoming request's headers or URL and decide which service it should be routed to.

Services

Services in Traefik refer to the backends that actually run the application. A service could be a web server or a set of load-balanced servers that Traefik forwards requests to after they've been received and routed. Your Payara Server serving your application is an example of a service in Traefik

Middlewares

These are the mechanisms that can alter the request or response during the routing process. They can handle tasks like authentication, request headers modification, rate-limiting, etc.

Load Balancer

Traefik can distribute the incoming requests across multiple instances of a service, based on various algorithms, to ensure optimal resource utilisation and fault tolerance.

Providers

Providers in Traefik are the source of backend configurations. Traefik can integrate with a variety of providers like Docker, Kubernetes, and many others to discover and manage services dynamically.

Certificates

For HTTPS traffic, Traefik can automatically issue and renew SSL/TLS certificates using Let's Encrypt or use provided SSL certificates for secure communication.

The “magic” of Traefik is that it uses labels defined on docker services to discover which services it should reverse proxy to. This full decoupling makes it a very powerful option for orchestrating very sophisticated deployments with Payara Server.

Traefik Configuration

Traefik can be configured in a number of ways. The following docker compose file shows a basic straightforward way to get up and running. This option combines container declaration and configuration into the same docker compose file.

```
version: '3.9'

services:
  traefik:
    image: traefik:v3.0
    command:
      - "--providers.docker=true"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--certificatesresolvers.myresolver.acme.tlschallenge=true"
      - "--certificatesresolvers.myresolver.acme.email=your-email@example.com"
      - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.
json"
      - "--providers.docker.network=web"
      - "--providers.docker.exposedbydefault=false"
      - "--middlewares.redirect-to-https.redirectscheme.scheme=https"
      - "--entrypoints.web.http.middlewares=redirect-to-https@docker"
    ports:
      - "80:80"
      - "443:443"
```

```
volumes:
  - "/var/run/docker.sock:/var/run/docker.sock:ro"
  - "./letsencrypt:/letsencrypt"

networks:
  - web

networks:
  web:
    external: true
```

The Docker Compose file defines a single service, traefik, which is based on the traefik:v3.0 image. It specifies a number of command-line options to configure Traefik:

- It enables Docker as the provider, allowing Traefik to automatically discover services running in Docker.
- It defines two entry points, web for HTTP traffic on port 80 and websecure for HTTPS traffic on port 443.
- It sets up Let's Encrypt for automatic SSL certificate generation and renewal, with ACME TLS challenge for validation.
- It specifies an email for ACME registration and a file to store the certificates.
- It restricts Traefik to only include containers in the web network that are explicitly labelled for Traefik.
- It adds a middleware directive to redirect all HTTP traffic to HTTPS.
- It maps the host's Docker socket and a local directory for storing certificates to the container, enabling Traefik to interact with the Docker API and store certificates.
- It attaches the Traefik service to an external network named web, which must be created outside of this Docker Compose file.
- The configuration ensures that any HTTP traffic hitting Traefik on port 80 will be redirected to HTTPS on port 443, securing all communications with SSL encryption.

That docker compose file for the Payara Server to which the Traefik will act as a proxy for is shown below.

```
version: '3.9'

services:
  payara-server:
    image: payara/server-full:6.2023.10-jdk17
    networks:
      - web
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.payara-server.rule=Host(`my-amazing-domain.com`)"
      - "traefik.http.routers.payara-server.entrypoints=web"
      - "traefik.http.routers.payara-server-secure.rule=Host(`my-amazing-domain.com`)"
      - "traefik.http.routers.payara-server-secure.entrypoints=websecure"
      - "traefik.http.routers.payara-server-secure.tls.certresolver=myresolver"
      - "traefik.http.middlewares.redirect-to-https.redirectscheme.scheme=https"
      - "traefik.http.routers.payara-server.middlewares=redirect-to-https"
      - "traefik.http.services.payara-server.loadbalancer.server.port=8080"
    expose:
      - "8080"

networks:
  web:
    external: true
```

The above docker compose defines a service named `payara-server` that runs the Payara Server on Java Development Kit (JDK) 17. It connects the service to an external network named `web`, indicating it should be used with a reverse proxy like Traefik. It's important to note that as the network for both docker-compose definitions are set to external, it must exist before attempting to create containers from the files.

Labels are attached to the service, configuring Traefik to route traffic:

- `traefik.enable=true` tells Traefik to discover this service.
- `traefik.http.routers.payara-server.rule` defines a rule for HTTP traffic to route requests for `payara.mydomain.com` to this service.
- Two routers are set up, one for HTTP (`payara-server`) and one for HTTPS (`payara-server-secure`), with the HTTPS router using a TLS certificate resolver named `myresolver`.
- The HTTPS middleware `redirect-to-https` is applied to the HTTP router, redirecting all HTTP traffic to HTTPS.
- The exposed port 8080 is the internal port of the Payara Server where the application is running, and Traefik will load balance requests to this port.

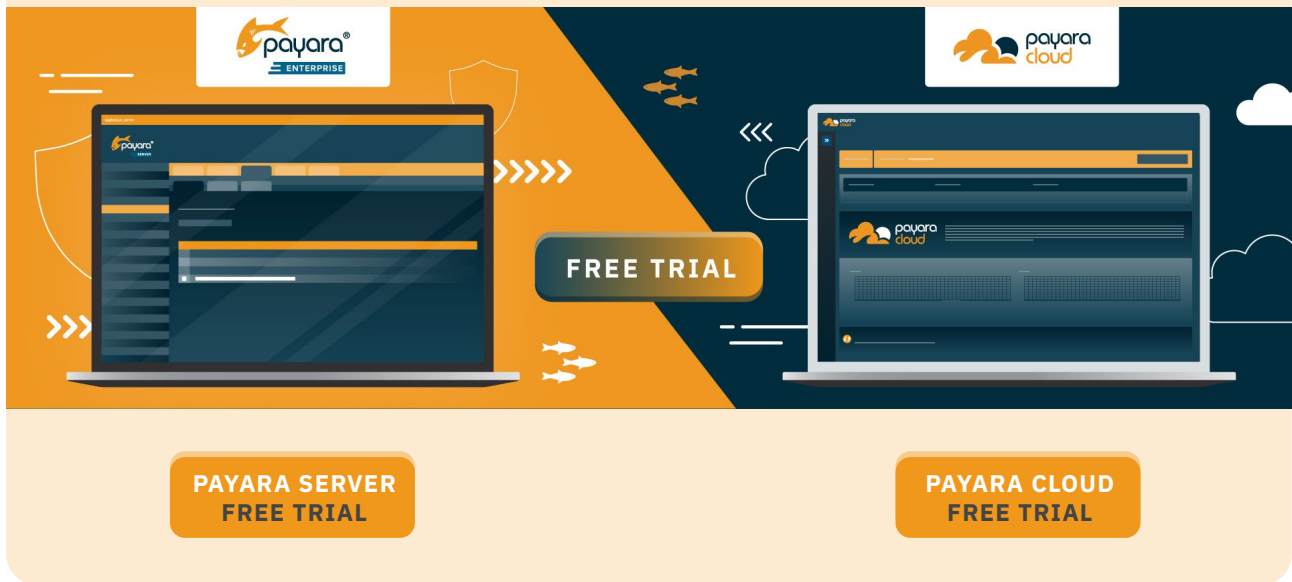
This configuration allows the Payara server to be accessed via `my-amazing-domain.com` with automatic HTTPS using certificates from Let's Encrypt.

This is one option to have Traefik act as a reverse proxy server for Payara Server running in a docker container. There are other options such as splitting Traefik configuration into a separate `traefik.yaml` file that you mount to the created container. You can find all the configuration options in the [Traefik documentation](#). You definitely should take a look to discover the infinite possibilities of using Traefik with Payara Server.

Summary

In this brief guide, we have taken a look at reverse proxying Payara Server. We looked at what a reverse proxy server is, the uses and benefits, then looked at some popular reverse proxy options and finally took a look at common configuration options for setting up Nginx and Traefik as reverse proxies for Payara Server. As noted in the respective sections, there are many different ways these servers can be set up. Do check out the linked documentations for all you can do with them. When you are ready, you can download a [free trial of Payara Enterprise](#) to start setting up your production deployment with Payara Server!

Interested in Payara? Try Before You Buy



The banner features two laptops. The left laptop displays the Payara Enterprise interface, with the logo 'payara ENTERPRISE' above it. The right laptop displays the Payara Cloud interface, with the logo 'payara cloud' above it. A central orange button with white text says 'FREE TRIAL'. Below each laptop is an orange button with white text: 'PAYARA SERVER FREE TRIAL' on the left and 'PAYARA CLOUD FREE TRIAL' on the right. The background is split into orange and dark blue sections with decorative elements like fish and arrows.



sales@payara.fish



UK: +44 800 538 5490
Intl: +1 888 239 8941



www.payara.fish

Payara Services Ltd 2023 All Rights Reserved. Registered in England and Wales; Registration Number 09998946
Registered Office: Malvern Hills Science Park, Geraldine Road, Malvern, United Kingdom, WR14 3SZ