Caddy

Introduction

Dans le monde de l'hébergement de sites web, choisir le bon serveur web peut faire toute la différence en termes de performance, de sécurité et de facilité de gestion. Caddy est un serveur web moderne et performant qui se distingue par sa simplicité d'utilisation et ses fonctionnalités avancées. Conçu pour automatiser les tâches fastidieuses comme la gestion des certificats SSL/TLS, Caddy permet aux administrateurs systèmes de se concentrer sur des aspects plus importants de leur infrastructure. Dans cet article, je vais vous présenter les caractéristiques principales de Caddy, son historique, les concepts de base, et quelques exemples de configurations pour que vous puissiez tirer le meilleur parti de ce serveur web innovant.

Historique de Caddy

Caddy a vu le jour en 2015, créé par **Matt Holt** avec une vision claire : simplifier la configuration et la gestion des serveurs web tout en intégrant des fonctionnalités modernes dès le départ. À l'époque, la gestion des certificats SSL/TLS était une tâche compliquée et chronophage pour les administrateurs systèmes. **Matt Holt** a voulu changer cela en intégrant le support automatique de Let's Encrypt directement dans **Caddy**, rendant ainsi la sécurité web accessible à tous sans effort supplémentaire.

Dès sa sortie, **Caddy** s'est distingué par sa philosophie axée sur la simplicité et l'automatisation. En plus de la gestion automatisée des certificats SSL/TLS, **Caddy** a introduit une syntaxe de configuration claire et lisible, appelée le **Caddyfile**, facilitant ainsi la vie des développeurs et des administrateurs systèmes.

Au fil des années, **Caddy** a continué d'évoluer et d'innover, ajoutant des fonctionnalités comme le support des plugins pour étendre ses capacités, une meilleure gestion des performances et des améliorations constantes en termes de sécurité. La communauté autour de **Caddy** a également grandi, contribuant à son développement et à son adoption dans divers environnements, des petits sites web personnels aux grandes infrastructures d'entreprise.

Aujourd'hui, **Caddy** est reconnu non seulement pour sa simplicité et ses fonctionnalités intégrées, mais aussi pour sa robustesse et sa flexibilité, faisant de lui un choix populaire parmi les administrateurs systèmes cherchant une solution moderne et efficace pour leurs besoins en serveur web.

Fonctionnalités principales de Caddy

Caddy se distingue par un ensemble de fonctionnalités qui le rendent unique parmi les serveurs web. Voici quelques-unes des fonctionnalités principales qui font de **Caddy** un choix populaire pour les administrateurs systèmes et les développeurs :

L'une des forces de Caddy réside dans sa facilité de configuration. Il utilise un fichier de configuration appelé **Caddyfile**, qui est conçu pour être lisible et facile à écrire. Voici un exemple de **Caddyfile** minimaliste :

```
example.com {
    root * /var/www/html
    file_server
}
```

Dans cet exemple, **Caddy** sert le contenu du répertoire `/var/www/html` lorsque le domaine `example.com` est accédé.

HTTPS automatique

Caddy automatise entièrement la gestion des certificats SSL/TLS via Let's Encrypt. Cela signifie que dès que vous configurez un domaine dans **Caddy**, il obtient automatiquement les certificats nécessaires et les renouvelle périodiquement sans intervention manuelle. Cette fonctionnalité simplifie grandement la sécurisation des sites web.

Extensibilité

Caddy est conçu pour être extensible grâce à un système de plugins. Ces plugins peuvent ajouter des fonctionnalités supplémentaires comme des authentifications spécifiques, des redirections avancées ou des intégrations avec d'autres services. La communauté développe activement des plugins pour répondre à divers besoins.

Performances élevées

Caddy est conçu pour être rapide et efficient en termes de ressources. Il utilise des techniques modernes pour gérer les connexions et les requêtes, ce qui permet de servir un grand nombre de requêtes simultanées sans sacrifier la performance.

Support natif des HTTP/2 et HTTP/3

Caddy prend en charge les protocoles HTTP/2 et HTTP/3 nativement, ce qui améliore la vitesse de chargement des pages et la sécurité. Ces protocoles sont conçus pour offrir une meilleure performance en termes de latence et de débit.

Gestion des fichiers statiques

Avec la directive `file_server`, **Caddy** peut facilement servir des fichiers statiques. Il peut également gérer les répertoires et fournir des listings de répertoires automatiquement.

Reverse proxy

Caddy peut agir en tant que revers-proxy, redirigeant les requêtes vers d'autres serveurs backend. Cette fonctionnalité est particulièrement utile pour les applications réparties sur plusieurs serveurs ou pour les microservices.

Redirections et réécritures d'URL

Caddy offre des fonctionnalités puissantes pour la redirection et la réécriture d'URL. Vous pouvez facilement configurer des redirections permanentes ou temporaires, ainsi que des réécritures d'URL complexes.

Journaux et surveillance

Caddy fournit des journaux détaillés et des outils de surveillance intégrés qui permettent de suivre l'activité du serveur et de diagnostiquer les problèmes. Ces outils sont essentiels pour maintenir une infrastructure stable et performante.

Sécurité renforcée

En plus de la gestion automatisée des certificats, **Caddy** intègre des fonctionnalités de sécurité telles que la prévention des attaques DDoS, la protection contre les scripts intersites (XSS) et bien d'autres.

Avec toutes ces fonctionnalités, **Caddy** se positionne comme un serveur web moderne et performant, idéal pour une variété de cas d'utilisation, des sites web simples aux infrastructures complexes nécessitant une gestion avancée des requêtes et des ressources.

Concepts de base

Pour tirer pleinement parti de **Caddy**, il est essentiel de comprendre quelques concepts de base. Ces concepts vous aideront à configurer et à gérer efficacement votre serveur web avec Caddy.

Caddyfile

Le **Caddyfile** est le fichier de configuration principal de Caddy. Il est conçu pour être simple et lisible, ce qui facilite la configuration des sites et des services. Un Caddyfile typique est structuré en blocs de configuration, chacun représentant un site ou un service distinct.

Voici un exemple de Caddyfile de base :

```
example.com {
    root * /var/www/html
    file_server
}
```

Dans cet exemple, le bloc `example.com` configure le domaine `example.com` pour servir le contenu du répertoire `/var/www/html` en utilisant la directive `file_server`.

Directives

Les **directives** sont des commandes spécifiques utilisées dans le Caddyfile pour configurer le comportement du serveur. Chaque directive a une fonction particulière, comme servir des fichiers statiques, rediriger des requêtes, ou configurer des paramètres de sécurité. Par exemple :

```
root * /var/www/html
file_server
```

Ces directives configurent le serveur pour servir des fichiers statiques à partir du répertoire `/var/www/html`.

Plugins

Caddy est conçu pour être extensible grâce à son support pour les **plugins**. Ces plugins permettent d'ajouter des fonctionnalités supplémentaires à Caddy, telles que l'authentification, la gestion des URL courtes ou des intégrations avec d'autres services. Les plugins peuvent être installés et configurés via le Caddyfile.

Routes et gestion des chemins

Les **routes** dans Caddy permettent de gérer comment les requêtes sont dirigées et traitées. Vous pouvez configurer des routes pour rediriger certaines requêtes vers des services backend, pour appliquer des réécritures d'URL, ou pour servir différents contenus selon le chemin de la requête. Voici un exemple de configuration de route pour un proxy inversé :

```
example.com {
reverse_proxy /api/* http://backend:8080
}
```

Dans cet exemple, toutes les requêtes vers `example.com/api/*` sont redirigées vers un serveur backend fonctionnant sur `http://backend:8080`.

Installation de Caddy

L'installation de **Caddy** est un processus simple et direct, que vous soyez sur un système d'exploitation Linux, macOS ou Windows. Dans cette section, je vais vous guider à travers les étapes d'installation de Caddy sur différents systèmes.

Installation sur Linux

Pour installer **Caddy** sur un système basé sur Debian (comme Ubuntu), vous pouvez suivre ces étapes :

Mettez à jour la liste des paquets et installez les prérequis :

```
sudo apt update
sudo apt install -y debian-keyring debian-archive-keyring apt-transport-https
```

Ajoutez la clé GPG de Caddy et le dépôt APT :

```
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/gpg.key' | sudo tee
/etc/apt/trusted.gpg.d/caddy-stable.asc
curl -1sLf 'https://dl.cloudsmith.io/public/caddy/stable/debian.deb.txt' | sudo tee
/etc/apt/sources.list.d/caddy-stable.list
```

Mettez à jour la liste des paquets et installez Caddy :

```
sudo apt update
sudo apt install caddy
```

Installation sur macOS

Pour installer Caddy sur macOS, vous pouvez utiliser Homebrew, un gestionnaire de paquets populaire pour macOS :

Installez Homebrew si ce n'est pas déjà fait. Vous pouvez le faire en suivant les instructions sur brew.sh ↗.

Installez Caddy en utilisant Homebrew :

```
brew install caddy
```

Installation sur Windows

Pour installer Caddy sur Windows, vous pouvez télécharger l'exécutable directement depuis le site officiel ou utiliser un gestionnaire de paquets comme Scoop ou Chocolatey.

Utilisation de Scoop

Installez Scoop si ce n'est pas déjà fait. Vous pouvez le faire en suivant les instructions sur scoop.sh ↗.

Installez Caddy en utilisant Scoop :

```
scoop install caddy
```

Utilisation de Chocolatey

Installez Chocolatey si ce n’est pas déjà fait. Vous pouvez le faire en suivant les instructions sur chocolatey.org ↗.

Installez Caddy en utilisant Chocolatey :

```
choco install caddy
```

Vérification de l’installation

Après avoir installé Caddy, vous pouvez vérifier que l’installation a réussi en exécutant la commande suivante dans votre terminal ou votre invite de commande :

```
caddy version
```

Cette commande doit afficher la version de Caddy installée sur votre système.

Démarrage et arrêt de Caddy

Pour démarrer Caddy en utilisant un fichier de configuration, utilisez la commande suivante :

```
caddy run --config /path/to/Caddyfile
```

Pour arrêter Caddy, vous pouvez utiliser `Ctrl+C` dans le terminal où Caddy est en cours d’exécution.

Exécution en tant que service

Sur les systèmes Linux, vous pouvez configurer Caddy pour qu’il s’exécute en tant que service système, ce qui permet de démarrer automatiquement Caddy au démarrage du système. Voici comment faire sur un système utilisant `systemd` :

Créez un fichier de service `systemd` pour Caddy :

```
sudo nano /etc/systemd/system/caddy.service
```

Ajoutez les lignes suivantes dans ce fichier :

```
[Unit]
Description=Caddy web server
After=network.target


[Service]
ExecStart=/usr/bin/caddy run --config /etc/caddy/Caddyfile
Restart=always
User=caddy
Group=caddy


[Install]
WantedBy=multi-user.target
```

Rechargez `systemd` et démarrez le service Caddy :

```
sudo systemctl daemon-reload
sudo systemctl start cadd
sudo systemctl enable caddy
```

Avec ces instructions, vous êtes prêt à installer et à exécuter Caddy sur différentes plateformes. Dans la prochaine section, je vais vous montrer comment configurer Caddy pour servir des sites web et des applications.

Configuration de base

Une fois **Caddy** installé, la prochaine étape consiste à le configurer pour servir des sites web et des applications. Dans cette section, je vais vous montrer comment créer un fichier de configuration Caddyfile et expliquer les directives de base pour démarrer rapidement.

Création du Caddyfile

Le **Caddyfile** est le fichier de configuration principal de Caddy. Il est conçu pour être simple et lisible. Voici un exemple de Caddyfile minimaliste pour servir un site web statique :

```
example.com {
root * /var/www/html
file_server
}
```

Dans cet exemple :

- `example.com` est le domaine pour lequel Caddy doit servir les fichiers.
- `root * /var/www/html` spécifie le répertoire racine contenant les fichiers du site web.
- `file_server` indique à Caddy de servir les fichiers statiques à partir du répertoire racine.

Servir des fichiers statiques

Pour servir des fichiers statiques à partir d'un répertoire, utilisez la directive `file_server`. Voici un exemple de configuration complète :

```
example.com {
root * /var/www/html
file_server browse
}
```

Avec `browse`, Caddy générera automatiquement une liste des fichiers et des répertoires si l'index du répertoire n'est pas trouvé.

Utilisation de variables d'environnement

Caddy permet l'utilisation de variables d'environnement pour rendre la configuration plus flexible. Voici comment définir et utiliser une variable d'environnement dans le Caddyfile :

```
{$SITE_DOMAIN} {
root * /var/www/html
file_server
tls {$EMAIL}
}
```

Avant de démarrer Caddy, vous pouvez définir les variables d’environnement :

```
export SITE_DOMAIN=example.com
export EMAIL=email@example.com
caddy run --config /path/to/Caddyfile
```

Gérer les erreurs personnalisées

Caddy permet de définir des pages d’erreur personnalisées pour différents codes d’erreur HTTP. Voici comment configurer une page d’erreur personnalisée pour les erreurs 404 :

```
example.com {
root * /var/www/html
file_server
handle_errors {
@404 {
expression {http.error.status_code} == 404
}
rewrite @404 /html
file_server
}
}
```

Activer la compression

Pour améliorer les performances, vous pouvez activer la compression des réponses HTTP. Caddy prend en charge la compression gzip et zstd :

```
example.com {
root * /var/www/html
file_server
encode gzip zstd
}
```

Utilisation des directives de réécriture

Les directives de réécriture permettent de modifier les URL des requêtes avant qu'elles ne soient traitées. Voici un exemple de réécriture d'URL :

```
example.com {
root * /var/www/html
file_server
rewrite /old-path /new-path
}
```

Configuration des en-têtes HTTP

Caddy permet de définir des en-têtes HTTP personnalisés pour les réponses. Voici un exemple de configuration des en-têtes :

```
example.com {
root * /var/www/html
file_server
header {
Strict-Transport-Security "max-age=31536000;"
X-Content-Type-Options "nosniff"
}
}
```

Rechargement de la configuration

Pour appliquer les modifications apportées au Caddyfile, vous devez recharger la configuration de Caddy. Cela peut être fait sans redémarrer le serveur en utilisant la commande suivante :

```
caddy reload --config /path/to/Caddyfile
```

Avec ces configurations de base, vous pouvez commencer à utiliser Caddy pour servir des sites web et des applications. La prochaine section abordera la gestion des certificats SSL/TLS de manière plus détaillée.

Gestion des certificats SSL/TLS

Une des fonctionnalités les plus appréciées de **Caddy** est sa capacité à gérer automatiquement les certificats SSL/TLS. Cette fonctionnalité simplifie grandement la sécurisation des sites web, éliminant la nécessité de gérer manuellement les certificats.

Obtention automatique des certificats

Caddy obtient automatiquement les certificats SSL/TLS pour vos domaines via Let’s Encrypt. Dès que vous configurez un domaine dans le Caddyfile, Caddy se charge d’obtenir et de renouveler les certificats sans intervention supplémentaire. Voici un exemple de configuration de base pour un site sécurisé :

```
example.com {
root * /var/www/html
file_server
tls email@example.com
}
```

Dans cet exemple, la directive `tls` avec une adresse email permet à Caddy de gérer les certificats SSL/TLS pour `example.com`.

Configuration avancée des certificats

Caddy offre des options avancées pour la gestion des certificats, comme l’utilisation de certificats personnalisés ou le contrôle des paramètres de TLS. Voici comment configurer des certificats personnalisés :

```
example.com {
root * /var/www/html
file_server
```

```
tls /path/to/cert.pem /path/to/key.pem
}
```

Dans cet exemple, Caddy utilise les fichiers `cert.pem` et `key.pem` pour le certificat et la clé privée.

Utilisation de DNS Challenge

Pour les domaines où le HTTP Challenge ne fonctionne pas (par exemple, les domaines wildcard), vous pouvez utiliser le DNS Challenge. Cela nécessite des configurations spécifiques pour le DNS provider. Voici un exemple pour Cloudflare :

Installez le plugin DNS correspondant :

```
xcaddy build --with github.com/caddy-dns/cloudflare
```

Configurez le DNS Challenge dans le Caddyfile :

```
example.com {
root * /var/www/html
file_server
tls {
dns cloudflare {env.CLOUDFLARE_API_TOKEN}
}
}
```

Définissez la variable d’environnement avec votre token API Cloudflare :

```
export CLOUDFLARE_API_TOKEN=your-cloudflare-api-token
caddy run --config /path/to/Caddyfile
```

Renouvellement des certificats

Caddy gère le renouvellement automatique des certificats avant leur expiration. Vous n’avez donc pas besoin de vous inquiéter de l’expiration des certificats. Caddy surveille les certificats et les renouvelle au besoin.

Gestion des erreurs liées aux certificats

Il est important de surveiller les logs de Caddy pour détecter toute erreur liée aux certificats, comme l'échec de l'obtention ou du renouvellement des certificats. Voici comment configurer Caddy pour journaliser les erreurs liées aux certificats :

```
example.com {
root * /var/www/html
file_server
tls email@example.com
log {
output file /var/log/caddy/caddy.log
level ERROR
}
}
```

Paramètres TLS avancés

Pour les utilisateurs avancés, Caddy permet de configurer des paramètres TLS spécifiques, comme les suites de chiffrement ou les protocoles à utiliser. Voici un exemple de configuration TLS avancée :

```
example.com {
root * /var/www/html
file_server
tls email@example.com {
protocols tlstls3
ciphers TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
}
}
```

Dans cet exemple, Caddy est configuré pour utiliser uniquement TLS et avec des suites de chiffrement spécifiques.

Monitoring et gestion des certificats

Pour une gestion proactive, vous pouvez utiliser des outils de monitoring pour surveiller l'état des certificats SSL/TLS et recevoir des alertes en cas de problème. Des solutions comme Let's Monitor ou des scripts personnalisés peuvent être intégrés à votre infrastructure de surveillance.

Avec ces fonctionnalités de gestion des certificats SSL/TLS, Caddy simplifie la sécurisation de vos sites web, en s'assurant que les certificats sont toujours à jour et configurés correctement. Dans la prochaine section, nous allons explorer l'utilisation des plugins pour étendre les capacités de Caddy.

Utilisation des plugins Caddy

Caddy est conçu pour être extensible et une des manières principales d'étendre ses fonctionnalités est par l'utilisation de **plugins**. Ces plugins permettent d'ajouter des fonctionnalités supplémentaires telles que l'authentification, la gestion des URL courtes, ou des intégrations avec d'autres services.

Les plugins pour Caddy sont disponibles sous forme de modules que vous pouvez inclure lors de la compilation de Caddy. Ils peuvent ajouter des fonctionnalités spécifiques ou modifier le comportement par défaut de Caddy. Vous pouvez trouver une liste des plugins disponibles sur le site officiel de Caddy.

Installation de plugins avec xcaddy

Pour installer des plugins, vous pouvez utiliser l'outil `xcaddy`, qui permet de compiler Caddy avec les plugins souhaités. Voici comment installer `xcaddy` et compiler Caddy avec un plugin :

Installez `xcaddy` :

```
go install github.com/caddyserver/xcaddy/cmd/xcaddy@latest
```

Compilez Caddy avec un plugin, par exemple le plugin `caddy-l4` pour le support de niveau 4 (TCP/UDP) :

```
xcaddy build --with github.com/mholt/caddy-l4
```

Une fois la compilation terminée, un nouveau binaire `caddy` est créé avec le plugin intégré.

Authentification

Le plugin `http.authz` permet d'ajouter des mécanismes d'authentification à vos sites web. Voici un exemple de configuration pour utiliser HTTP Basic Auth :

```
example.com {
root * /var/www/html
file_server



basicauth {
user hashed-password
}
}
```

Vous pouvez générer le mot de passe haché en utilisant une commande comme `htpasswd` :

```
htpasswd -nB user
```

Gestion des URL courtes

Le plugin `http.urlshort` permet de créer des redirections courtes. Voici un exemple de configuration :

```
example.com {
route /short {
redir /long-url
}
}
```

WebSocket

Pour ajouter le support WebSocket, vous pouvez utiliser le plugin `caddy.websocket`. Voici un exemple de configuration pour un serveur WebSocket simple :

```
example.com {
root * /var/www/html
file_server
reverse_proxy /ws localhost:8080 {
transport http {
versions h2c 2
}
}
}
```

Cloudflare DNS

Le plugin `caddy-dns/cloudflare` permet d'utiliser le DNS Challenge avec Cloudflare pour les certificats SSL/TLS. Voici un exemple de configuration :

```
example.com {
root * /var/www/html
file_server
tls {
dns cloudflare {env.CLOUDFLARE_API_TOKEN}
}
}
```

Configuration des plugins dans le Caddyfile

Les plugins sont configurés dans le Caddyfile en utilisant des directives spécifiques à chaque plugin. Chaque plugin dispose de sa propre documentation pour les options de configuration disponibles. Voici un exemple de Caddyfile avec plusieurs plugins configurés :

```
example.com {
root * /var/www/html
file_server



# Basic Auth plugin
```

```
basicauth {
user hashed-password
}


# URL Shortener plugin
route /short {
redir /long-url
}


# WebSocket plugin
reverse_proxy /ws localhost:8080 {
transport http {
versions h2c 2
}
}


# Cloudflare DNS plugin
tls {
dns cloudflare {env.CLOUDFLARE_API_TOKEN}
}
}
```

Mise à jour et gestion des plugins

Pour mettre à jour Caddy avec les plugins, vous pouvez simplement recompiler Caddy avec `xcaddy` en spécifiant les versions mises à jour des plugins. Il est important de vérifier régulièrement les mises à jour de sécurité et les nouvelles fonctionnalités des plugins utilisés.

Dépannage des plugins

En cas de problèmes avec les plugins, consultez les logs de Caddy pour obtenir des informations sur les erreurs. Vous pouvez également consulter la documentation spécifique du plugin et les forums de support pour obtenir de l’aide.

Avec cette flexibilité offerte par les plugins, vous pouvez adapter Caddy à une multitude de scénarios et besoins spécifiques. La prochaine section traitera des meilleures pratiques pour la gestion et l'optimisation des performances de Caddy.

Gestion et optimisation des performances

Caddy est conçu pour être performant dès la sortie de la boîte, mais il existe des pratiques et des configurations supplémentaires qui peuvent améliorer encore plus ses performances. Dans cette section, je vais aborder les techniques de gestion et d'optimisation des performances pour tirer le meilleur parti de votre serveur Caddy.

Gestion de la mémoire et des ressources

Pour surveiller et gérer l'utilisation de la mémoire et des ressources CPU par Caddy, vous pouvez utiliser des outils de monitoring comme **Prometheus** et **Grafana**. Voici comment configurer Caddy pour exporter des métriques compatibles avec Prometheus :

Ajoutez le module Prometheus lors de la compilation de Caddy :

```
xcaddy build --with github.com/prometheus/client_golang/prometheus
```

Configurez le Caddyfile pour exporter les métriques :

```
{
    metrics {
        prometheus
    }
}

example.com {
    root * /var/www/html
    file_server
}
```

Configurez Prometheus pour scraper les métriques exportées par Caddy en ajoutant cette section dans le fichier de configuration de Prometheus (`prometheus.yml`) :

```
scrape_configs:
- job_name: "caddy"
static_configs:
- targets: ["localhost:2019"]
```

Compression et mise en cache

La compression et la mise en cache des réponses peuvent considérablement améliorer les performances de votre site web.

Compression

Pour activer la compression des réponses HTTP, utilisez la directive `encode` :

```
example.com {
root * /var/www/html
file_serve
encode gzip zstd
}
```

Mise en cache

Bien que Caddy ne propose pas de solution de mise en cache intégrée, vous pouvez utiliser des solutions externes comme **Varnish** ou des modules tiers pour gérer la mise en cache des réponses. Voici un exemple de configuration de Varnish avec Caddy :

Installez Varnish sur votre serveur.

Configurez Varnish pour écouter sur le port 80 et configurer Caddy pour écouter sur un port différent (par exemple, 8080).

Modifiez le fichier de configuration de Varnish (`default.vcl`) pour rediriger les requêtes vers Caddy :

```
vcl 0;



backend default {
.host = "1";
.port = "8080";
}



sub vcl_recv {
if (req.method == "PURGE") {
if (!client.ip ~ purgers) {
return (synth(405, "Not allowed."));
}
return (purge);
}
}



sub vcl_backend_response {
if (beresp.status == 200) {
set beresp.ttl = 10m;
}
}
```

Configurez le Caddyfile pour écouter sur le port 8080 :

```
example.com {
root * /var/www/html
file_server
encode gzip zstd
reverse_proxy localhost:8080
}
```

Optimisation des en-têtes HTTP

L’optimisation des en-têtes HTTP peut améliorer la sécurité et les performances de votre site web.

En-têtes de sécurité

Ajoutez des en-têtes de sécurité pour protéger votre site contre les attaques courantes :

```
example.com {
root * /var/www/html
file_server
header {
Strict-Transport-Security "max-age=31536000;"
X-Content-Type-Options "nosniff"
X-Frame-Options "DENY"
Referrer-Policy "no-referrer"
Content-Security-Policy "default-src 'self'"
}
}
```

En-têtes de mise en cache

Configurez des en-têtes de mise en cache pour les ressources statiques :

```
example.com {
    root * /var/www/html
    file_server
    header /static/* {
        Cache-Control "public, max-age=31536000, immutable"
    }
}
```

Gestion des connexions

Configurer le nombre maximal de connexions et de workers peut aider à gérer la charge sur votre serveur :

```
example.com {
    root * /var/www/html
    file_server
    tls email@example.com
    max_conns 100
    max_header_bytes 1048576
}
```

Utilisation de HTTP/2 et HTTP/3

Caddy supporte HTTP/2 par défaut et peut également être configuré pour utiliser HTTP/3 pour des performances encore meilleures :

Activez HTTP/3 dans le Caddyfile :

```
example.com {
    root * /var/www/html
    file_server
    tls {
        alpn h3
    }
}
```

Vérifiez que votre serveur supporte HTTP/3 et que les ports UDP nécessaires sont ouverts.

Load Balancing

Pour gérer un trafic important, vous pouvez configurer Caddy en tant que load balancer :

```
example.com {
    reverse_proxy {
        to backend1:8080 backend2:8080 backend3:8080
        lb_policy round_robin
    }
}
```

Optimisation des logs

```
example.com {
    root * /var/www/html
    file_server
    log {
        output file /var/log/caddy/access.log {
            roll_size 50mb
            roll_keep 5
            roll_keep_for 48h
        }
        level INFO
    }
}
```

En appliquant ces techniques de gestion et d'optimisation des performances, vous pouvez garantir que votre serveur Caddy fonctionne efficacement et reste performant même sous des charges élevées. Dans la prochaine section, je conclurai notre exploration de Caddy et résumerai les points clés abordés.

Conclusion

Au fil de ce billet, j'ai détaillé les nombreuses fonctionnalités et avantages qu'offre **Caddy** en tant que serveur web moderne et performant. Nous avons exploré son historique, ses concepts clés et ses fonctionnalités principales telles que la gestion automatique des certificats SSL/TLS, la flexibilité du fichier de configuration Caddyfile et l'utilisation de plugins pour étendre ses capacités.

En conclusion, **Caddy** se positionne comme un choix robuste et flexible pour la gestion des serveurs web. Grâce à ses fonctionnalités intégrées et à son extensibilité, il offre une solution complète pour déployer et gérer des sites web sécurisés et performants. J'espère que ce billet vous a aidé à mieux comprendre les avantages de **Caddy** et comment l'utiliser efficacement dans vos projets.

Set Up a CrowdSec Using OPNsense LAPI on Caddy

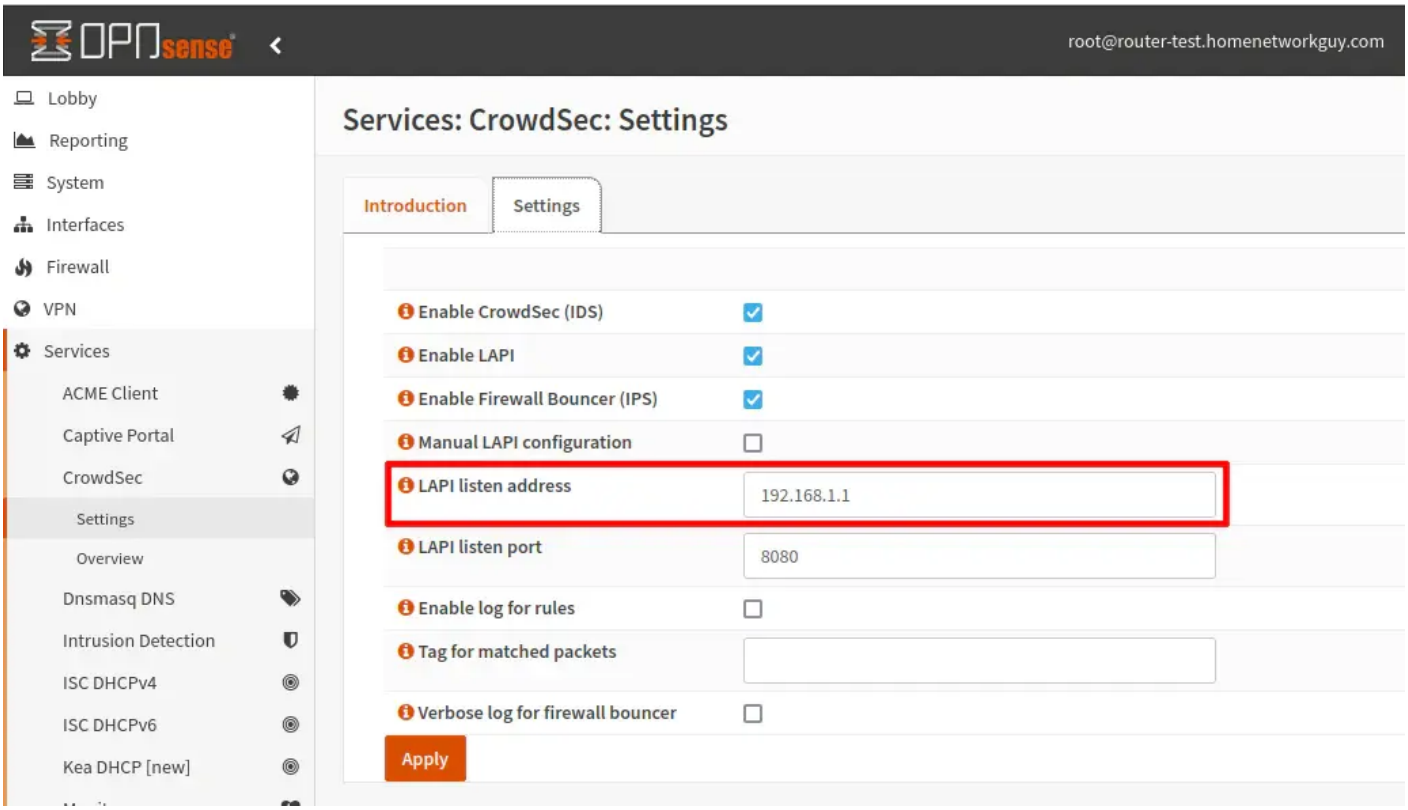
Prepare the OPNsense CrowdSec Configuration

Before setting up the Caddy reverse proxy, some settings for CrowdSec and firewall rules can be configured in OPNsense to prepare for a CrowdSec multi-server environment.

Update the Existing CrowdSec Plugin Configuration

The first thing you can do is change your CrowdSec plugin settings in OPNsense to allow other CrowdSec agents/bouncers to use the LAPI (Local API) on OPNsense. By default, the LAPI on OPNsense only listens on `localhost` (`127.0.0.1`).

In this example, I am setting the IP address to be on the LAN interface of `192.168.1.1`. You may wish to put it on a different interface.



You will need to start/stop the CrowdSec plugin for changes to take effect.

Create API Key for Caddy CrowdSec Bouncer

To prepare for setting up the CrowdSec bouncer for Caddy in a later step, you will need an API key generated for the bouncer.

Log into your OPNsense system via SSH or the console and issue the following command to create an API key. You may use any name you wish in place of `caddyDmz`. I used that name since this will be for a Caddy instance on the DMZ network.

```
sudo cscli bouncers add caddy
```

You should see the API key in the console output. Copy/paste this key until it is needed later.

```
API key for 'caddy':

   xxxxxxxxxxxxxxxxxxxxxx

Please keep this key since you will not be able to retrieve it!
```

Build Caddy with the Desired Modules

In order to use DNS challenges with Let's Encrypt and to use a CrowdSec bouncer, you will need to a build a custom Caddy executable to extend the base functionality. Fortunately, the build process is easy with `xcaddy` since you can build a Caddy executable with a single command.

I will be using the Cloudflare module for Let's Encrypt but you may use a different provider. Replace `github.com/caddy-dns/cloudflare` with a provider from the list found on GitHub.

```
xcaddy build \
    --with github.com/caddy-dns/cloudflare \
    --with github.com/hslatman/caddy-crowdsec-bouncer/crowdsec
```

Delete the old file and move the file to the `/usr/bin/` folder:

```
sudo rm /usr/bin/caddy
sudo mv caddy /usr/bin/
```

You should be able to run the `caddy` executable to ensure it can be found on the path.

```
caddy version
```

Modify Caddyfile

In the configuration file, you will need to enter the Cloudflare API key used for editing DNS zones for the `acme_dns cloudflare` option.

In addition to the DNS API key, newer versions of Caddy (v2.8.0+) require you to enter an email address for ZeroSSL (Caddy uses both Let's Encrypt and ZeroSSL for issuing certificates).

Then in a `crowdsec` block, you will need to enter the API key what was generated from OPNsense earlier. The URL is for the CrowdSec LAPI on OPNsense which is `192.168.1.1:8080`.

These first two settings should be contained in the global settings block as shown below.

```
{
        email nicolas.lespinasse@vainsta.fr
        acme_dns cloudflare xxxxxxxxxxxxxxxxxxxxxxxx
        admin :2019
        metrics
        crowdsec {
                api_key xxxxxxxxxxxxxxx
                api_url http://192.168.1.1:8080/
        }
```

```
}

panel.vainsta.fr {
        log {
                output file /var/log/caddy/panel-access.log {
                        roll_size 100mb
                        roll_keep 20
                        roll_keep_for 720h
                }
                format json
                level INFO
        }
        reverse_proxy https://192.168.1.4:444 {
                transport http {
                        tls_insecure_skip_verify
                }
        }
}
```

There are two additional CrowdSec bouncer options you may include.

If you wish for the bouncer to check the LAPI each time instead of caching the decisions in memory and polling the LAPI every 10 seconds by default, you can disable streaming by adding the `disable_streaming` option to the `crowdsec` block. Streaming decision information is more efficient if you have a lot of requests, but it is possible there will be a slight increase in delay when decisions on the LAPI have changed.

```
...
        crowdsec {
            ...
            disable_streaming
...
```

There is an option to enable “hard fails” if there is an issue connecting to the CrowdSec LAPI. This feature might be nice if you wish to prevent your services from being accessed if something is wrong with the LAPI since they will be unprotected by CrowdSec. Of course, that would negatively affect uptime, but it would make it very apparent something bad has happened.

I noticed it takes about 30 seconds to fail after the CrowdSec service is stopped in OPNsense. I was starting to wonder if this option was working properly, but I simply was not patient enough during my testing.

```
...
        crowdsec {
            ...
            enable_hard_fails
...
```

Press “Ctrl + O”, “Enter”, and “Ctrl + X” to save and close the `Caddyfile`.

Install CrowdSec Agent on Caddy Server

The CrowdSec bouncer is what will block connections to services behind the reverse proxy, but the Caddy server will need a CrowdSec agent installed so it can run the parsers and scenarios on the server. The agent sends the information to the LAPI on OPNsense to make decisions on blocking content. For a single CrowdSec instance, this usually occurs on the same system, but in a multi-server configuration, the CrowdSec agent and bouncer communicates with the LAPI on a different server (in this case, OPNsense).

Install CrowdSec using the following commands. Basically the next step is following the CrowdSec installation guide. It is very simple to install.

```
curl -s https://packagecloud.io/install/repositories/crowdsec/crowdsec/script.deb.sh | sudo bash
sudo apt install crowdsec
```

The CrowdSec agent needs to be registered with OPNsense CrowdSec LAPI.

```
sudo cscli lapi register -u http://192.168.1.1:8080
```

Copy the default CrowdSec `systemd` file from `/lib/systemd/system` to `/etc/systemd/system` so customizations can be made to the service file.

```
sudo cp /lib/systemd/system/crowdsec.service /etc/systemd/system/crowdsec.service
```

Edit the `/etc/systemd/system/crowdsec.service` to add the `-no-api` flag to the end of the `ExecStart` command. This disables the LAPI on the Caddy server since it is not needed because the LAPI on OPNsense will be used instead (see CrowdSec’s multi-server configuration example).

```
[Unit]
Description=Crowdsec agent
After=syslog.target network.target remote-fs.target nss-lookup.target
```

```
[Service]
Type=notify
Environment=LC_ALL=C LANG=C
ExecStartPre=/usr/bin/crowdsec -c /etc/crowdsec/config.yaml -t -error
ExecStart=/usr/bin/crowdsec -c /etc/crowdsec/config.yaml -no-api
#ExecStartPost=/bin/sleep 0.1
ExecReload=/bin/kill -HUP $MAINPID
Restart=always
RestartSec=60

[Install]
WantedBy=multi-user.target
```

You will need to reload the `systemd` service since changes to the service was made.

```
sudo systemctl daemon-reload
```

Do not reload the CrowdSec service just yet because with the LAPI on the Caddy server disabled, CrowdSec will error on startup until you have validated the Caddy machine on OPNsense. CrowdSec will be unable to connect to the LAPI on OPNsense until validation occurs. When there is no reachable LAPI, the CrowdSec agent will fail to load.

Validate the Caddy Machine in OPNsense

While logged into OPNsense via SSH or the console, list the machines which have been registered or requesting to be registered:

```
sudo cscli machines list
```

You should see similar output to below. Notice the status of the machine `02a3nfadce4ez4b19zh582e0f68f72a4CX4EzFJ2Th4PNkj1` shows the “No” symbol under “Status”.

```
────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────
 Name                                                IP Address      Last Update
Status   Version                      Auth Type   Last Heartbeat
────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────────────────────────────────
 localhost                                           192.168.1.1      2024-03-11T14:11:50Z   ✔️
       v1.6.0-freebsd-4b8e6cd7   password    21s
 02a3nfadce4ez4b19zh582e0f68f72a4CX4EzFJ2Th4PNkj1   192.168.1.2    2024-03-11T13:55:29Z   
```

```
                                password     ⚠ 16m42s
──────────────────────────────────────────────────────────────────────────────
──────────────────────────────────────────────────────────────
```

To validate the machine, you can enter the following command.

```
sudo cscli machines validate 02a3nfadce4ez4b19zh582e0f68f72a4CX4EzFJ2Th4PNkj1
```

Add Collection(s) to CrowdSec Agent on the Caddy Server

To add extra Caddy-specific parsers, you can add the following collection to your CrowdSec installation on your Caddy server. The Caddy collection includes a Caddy log parser and basic HTTP protections.

While the Caddy log parser may only be beneficial if you are using Caddy as a web server instead of a reverse proxy, the included basic HTTP protections should be helpful to protect web apps that are behind the reverse proxy.

```
sudo cscli collections install crowdsecurity/caddy
```

If you are hosting a public service (which great care must be taken to address security), it may not be a bad idea to also include the HTTP DoS collection to help detect denial of service attacks.

Of course, you should test this does not interfere with the normal operation of your app/service. I also do not know if this would be beneficial if you are using a Cloudflare proxy which includes DDoS protection.

```
sudo cscli collections install crowdsecurity/http-dos
```

You may now finally restart the CrowdSec agent on the Caddy server.

```
sudo systemctl reload crowdsec
```

After reloading CrowdSec, you should check if it is running properly.

```
sudo systemctl status crowdsec
```