bunkerized-nginx

Release v1.3.2

bunkerity

Jun 22, 2022

CONTENTS

1 Introduction **1**

2 Integrations **3**
- 2.1 Docker . . . 3
- 2.2 Docker autoconf . . . 5
- 2.3 Docker Swarm . . . 8
- 2.4 Kubernetes . . . 11
- 2.5 Linux . . . 16

3 Quickstart guide **19**
- 3.1 Reverse proxy . . . 19
- 3.2 PHP applications . . . 24
- 3.3 Multisite . . . 29

4 Special folders **37**
- 4.1 Multisite . . . 37
- 4.2 Web files . . . 37
- 4.3 http configurations . . . 37
- 4.4 server configurations . . . 38
- 4.5 ModSecurity configurations . . . 38
- 4.6 CRS configurations . . . 38
- 4.7 Cache . . . 39
- 4.8 Plugins . . . 39
- 4.9 ACME challenge . . . 39

5 Security tuning **41**
- 5.1 Miscellaneous . . . 41
- 5.2 HTTPS . . . 41
- 5.3 Headers . . . 43
- 5.4 ModSecurity . . . 43
- 5.5 Bad behaviors detection . . . 44
- 5.6 Antibot challenge . . . 44
- 5.7 External blacklists . . . 44
- 5.8 Limiting . . . 46
- 5.9 Country . . . 46
- 5.10 Authentication . . . 47
- 5.11 Whitelisting . . . 47
- 5.12 Blacklisting . . . 47
- 5.13 Plugins . . . 48
- 5.14 Container hardening . . . 48

6 Web UI **51**
6.1 Overview . . . 51
6.2 Usage . . . 51

7 List of environment variables **57**
7.1 nginx . . . 57
7.2 HTTPS . . . 65
7.3 ModSecurity . . . 67
7.4 Security headers . . . 68
7.5 Blocking . . . 69
7.6 PHP . . . 75
7.7 Bad behavior . . . 75
7.8 Authelia . . . 76
7.9 misc . . . 77

8 Troubleshooting **79**
8.1 Logs . . . 79
8.2 Permissions . . . 79
8.3 ModSecurity . . . 79
8.4 Bad behavior . . . 80
8.5 Whitelisting . . . 80

9 Plugins **81**
9.1 Official plugins . . . 81
9.2 Community plugins . . . 81
9.3 Use a plugin . . . 81
9.4 Write a plugin . . . 82

CHAPTER

ONE

INTRODUCTION

Make security by default great again !

bunkerized-nginx is a web server based on the notorious nginx and focused on security. It integrates into existing environments (Linux, Docker, Swarm, Kubernetes, ...) to make your web services "secured by default" without any hassle. The security best practices are automatically applied for you while keeping control of every settings to meet your own use case.

Non-exhaustive list of features :

- HTTPS support with transparent Let's Encrypt automation
- State-of-the-art web security : HTTP security headers, prevent leaks, TLS hardening, ...
- Integrated ModSecurity WAF with the OWASP Core Rule Set
- Automatic ban of strange behaviors
- Antibot challenge through cookie, javascript, captcha or recaptcha v3
- Block TOR, proxies, bad user-agents, countries, ...
- Block known bad IP with DNSBL
- Prevent bruteforce attacks with rate limiting
- Plugins system for external security checks (ClamAV, CrowdSec, ...)
- Easy to configure with environment variables or web UI
- Seamless integration into existing environments : Linux, Docker, Swarm, Kubernetes, ...

Fooling automated tools/scanners :

You can find a live demo at https://demo-nginx.bunkerity.com, feel free to do some security tests.

CHAPTER

TWO

INTEGRATIONS

2.1 Docker

You can get official prebuilt Docker images of bunkerized-nginx for x86, x64, armv7 and aarch64/arm64 architectures on Docker Hub :

```
$ docker pull bunkerity/bunkerized-nginx
```

Or you can build it from source if you wish :

```
$ git clone https://github.com/bunkerity/bunkerized-nginx.git
$ cd bunkerized-nginx
$ docker build -t bunkerized-nginx .
```

To use bunkerized-nginx as a Docker container you have to pass specific environment variables, mount volumes and redirect ports to make it accessible from the outside.

To demonstrate the use of the Docker image, we will create a simple "Hello World" static file that will be served by bunkerized-nginx.

One important thing to know is that the container runs as an unprivileged user with UID and GID 101. The reason behind this behavior is the security : in case a vulnerability is exploited the attacker won't have full privileges inside the container. But there is also a downside because bunkerized-nginx (heavily) make use of volumes, you will need to adjust the rights on the host.

First create the environment on the host :

```
$ mkdir bunkerized-hello bunkerized-hello/www bunkerized-hello/certs
$ cd bunkerized-hello
$ chown root:101 www certs
$ chmod 750 www
$ chmod 770 certs
```

The www folder will contain our static files that will be served by bunkerized-nginx. Whereas the certs folder will store the automatically generated Let's Encrypt certificates.

Let's create a dummy static page into the www folder :

```
$ echo "Hello bunkerized World !" > www/index.html
$ chown root:101 www/index.html
$ chmod 740 www/index.html
```

It's time to run the container :

```
$ docker run \
         -p 80:8080 \
         -p 443:8443 \
         -v "${PWD}/www:/www:ro" \
         -v "${PWD}/certs:/etc/letsencrypt" \
         -e SERVER_NAME=www.example.com \
         -e AUTO_LETS_ENCRYPT=yes \
         bunkerity/bunkerized-nginx
```

Or if you prefer docker-compose :

```
version: '3'
services:
  mybunkerized:
    image: bunkerity/bunkerized-nginx
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./www:/www:ro
      - ./certs:/etc/letsencrypt
    environment:
      - SERVER_NAME=www.example.com
      - AUTO_LETS_ENCRYPT=yes
```

Important things to note :

- Replace www.example.com with your own domain (it must points to your server IP address if you want Let's Encrypt to work)
- Automatic Let's Encrypt is enabled thanks to `AUTO_LETS_ENCRYPT=yes` (since the default is `AUTO_LETS_ENCRYPT=no` you can remove the environment variable to disable Let's Encrypt)
- The container is exposing TCP/8080 for HTTP and TCP/8443 for HTTPS
- The /www volume is used to deliver static files and can be mounted as read-only for security reason
- The /etc/letsencrypt volume is used to store certificates and must be mounted as read/write

Inspect the container logs until bunkerized-nginx is started then visit http(s)://www.example.com to confirm that everything is working as expected.

This example is really simple but, as you can see in the list of environment variables, you may get a lot of environment variables depending on your use case. To make things cleanier, you can write the environment variables to a file :

```
$ cat variables.env
SERVER_NAME=www.example.com
AUTO_LETS_ENCRYPT=yes
```

And load the file when creating the container :

```
$ docker run ... --env-file "${PWD}/variables.env" ... bunkerity/bunkerized-nginx
```

Or if you prefer docker-compose :

```
...
services:
```

(continues on next page)

(continued from previous page)

```
  mybunkerized:
    ...
    env_file:
      - ./variables.env
    ...
...
```

2.2 Docker autoconf

The downside of using environment variables is that the container needs to be recreated each time there is an update which is not very convenient. To counter that issue, you can use another image called bunkerized-nginx-autoconf which will listen for Docker events and automatically configure bunkerized-nginx instance in real time without recreating the container. Instead of defining environment variables for the bunkerized-nginx container, you simply add labels to your web services and bunkerized-nginx-autoconf will "automagically" take care of the rest.

First of all, you will need a network to allow communication between bunkerized-nginx and your web services :

```
$ docker network create services-net
```

We will also make use of a named volume to share the configuration between autoconf and bunkerized-nginx :

```
$ docker volume create bunkerized-vol
```

You can now create the bunkerized-nginx container :

```
$ docker run \
         --name mybunkerized \
         -l bunkerized-nginx.AUTOCONF \
         --network services-net \
         -p 80:8080 \
         -p 443:8443 \
         -v "${PWD}/www:/www:ro" \
         -v "${PWD}/certs:/etc/letsencrypt" \
         -v bunkerized-vol:/etc/nginx \
         -e MULTISITE=yes \
         -e SERVER_NAME= \
         -e AUTO_LETS_ENCRYPT=yes \
         bunkerity/bunkerized-nginx
```

The autoconf one can now be started :

```
$ docker run \
         --name myautoconf \
         --volumes-from mybunkerized:rw \
         -v /var/run/docker.sock:/var/run/docker.sock:ro \
         bunkerity/bunkerized-nginx-autoconf
```

Here is the docker-compose equivalent :

```
version: '3'
```

(continues on next page)

(continued from previous page)

```
services:

  mybunkerized:
    image: bunkerity/bunkerized-nginx
    restart: always
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./certs:/etc/letsencrypt
      - ./www:/www:ro
      - bunkerized-vol:/etc/nginx
    environment:
      - SERVER_NAME=
      - MULTISITE=yes
      - AUTO_LETS_ENCRYPT=yes
    labels:
      - "bunkerized-nginx.AUTOCONF"
    networks:
      - services-net

  myautoconf:
    image: bunkerity/bunkerized-nginx-autoconf
    restart: always
    volumes_from:
      - mybunkerized
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    depends_on:
      - mybunkerized

volumes:
  bunkerized-vol:

networks:
  services-net:
    name: services-net
```

Important things to note :

- autoconf is generating config files and other artefacts for the bunkerized-nginx, they need to share the same volumes
- autoconf must have access to the Docker socket in order to get events, access to labels and send SIGHUP signal (reload order) to bunkerized-nginx
- bunkerized-nginx must have the bunkerized-nginx.AUTOCONF label
- bunkerized-nginx must be started in multisite mode with the `MULTISITE=yes` environment variable
- When setting the `SERVER_NAME` environment variable to an empty value, bunkerized-nginx won't generate any web service configuration at startup
- The `AUTO_LETS_ENCRYPT=yes` will be applied to all subsequent web service configuration, unless overriden by the web service labels

Check the logs of both autoconf and bunkerized-nginx to see if everything is working as expected.

You can now create a new web service and add environment variables as labels with the `bunkerized-nginx.` prefix to let the autoconf service "automagically" do the configuration for you :

```
$ docker run \
         --name myservice \
         --network services-net \
         -l bunkerized-nginx.SERVER_NAME=www.example.com \
         -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
         -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
         -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice \
         tutum/hello-world
```

docker-compose equivalent :

```
version: "3"

services:

  myservice:
    image: tutum/hello-world
    networks:
      services-net:
        aliases:
          - myservice
    labels:
      - "bunkerized-nginx.SERVER_NAME=www.example.com"
      - "bunkerized-nginx.USE_REVERSE_PROXY=yes"
      - "bunkerized-nginx.REVERSE_PROXY_URL=/"
      - "bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice"

networks:
  services-net:
    external:
      name: services-net
```

Please note that if you want to override the `AUTO_LETS_ENCRYPT=yes` previously defined in the bunkerized-nginx container, you simply need to add the `bunkerized-nginx.AUTO_LETS_ENCRYPT=no` label.

Look at the logs of both autoconf and bunkerized-nginx to check if the configuration has been generated and loaded by bunkerized-nginx. You should now be able to visit http(s)://www.example.com.

When your container is not needed anymore, you can delete it as usual. The autoconf should get the event and generate the configuration again.

2.3 Docker Swarm

The deployment and configuration is very similar to the "Docker autoconf" one but with services instead of containers. A service based on the bunkerized-nginx-autoconf image needs to be scheduled on a manager node (don't worry it doesn't expose any network port for obvious security reasons). This service will listen for Docker Swarm events like service creation or deletion and generate the configuration according to the labels of each service. Once configuration generation is done, the bunkerized-nginx-autoconf service will send the configuration files and a reload order to all the bunkerized-nginx tasks so they can apply the new configuration. If you need to deliver static files (e.g., html, images, css, js, ...) a shared folder accessible from all bunkerized-nginx instances is needed (you can use a storage system like NFS, GlusterFS, CephFS on the host or a Docker volume plugin).

In this setup we will deploy bunkerized-nginx in global mode on all workers and autoconf as a single replica on a manager.

First of all, you will need to create 2 networks, one for the communication between bunkerized-nginx and autoconf and the other one for the communication between bunkerized-nginx and the web services :

```
$ docker network create -d overlay --attachable bunkerized-net
$ docker network create -d overlay --attachable services-net
```

We can now start the bunkerized-nginx as a service :

```
$ docker service create \
         --name mybunkerized \
         --mode global \
         --constraint node.role==worker \
         -l bunkerized-nginx.AUTOCONF \
         --network bunkerized-net \
         -p published=80,target=8080,mode=host \
         -p published=443,target=8443,mode=host \
         -e SWARM_MODE=yes \
         -e USE_API=yes \
         -e API_URI=/ChangeMeToSomethingHardToGuess \
         -e SERVER_NAME= \
         -e MULTISITE=yes \
         -e AUTO_LETS_ENCRYPT=yes \
         bunkerity/bunkerized-nginx
$ docker service update \
         --network-add services-net \
         mybunkerized
```

Once bunkerized-nginx has been started you can start the autoconf as a service :

```
$ docker service create \
         --name myautoconf \
         --replicas 1 \
         --constraint node.role==manager \
         --network bunkerized-net \
         --mount type=bind,source=/var/run/docker.sock,destination=/var/run/docker.sock,
↪ro \
         --mount type=volume,source=cache-vol,destination=/cache \
         --mount type=volume,source=certs-vol,destination=/etc/letsencrypt \
         -e SWARM_MODE=yes \
         -e API_URI=/ChangeMeToSomethingHardToGuess \
```

(continues on next page)

(continued from previous page)

```
        bunkerity/bunkerized-nginx-autoconf
```

Or do the same with docker-compose if you wish :

```
version: '3.8'

services:

  nginx:
    image: bunkerity/bunkerized-nginx
    ports:
      - published: 80
        target: 8080
        mode: host
        protocol: tcp
      - published: 443
        target: 8443
        mode: host
        protocol: tcp
    environment:
      - SWARM_MODE=yes
      - USE_API=yes
      - API_URI=/ChangeMeToSomethingHardToGuess # must match API_URI from autoconf
      - MULTISITE=yes
      - SERVER_NAME=
      - AUTO_LETS_ENCRYPT=yes
    networks:
      - bunkerized-net
      - services-net
    deploy:
      mode: global
      placement:
        constraints:
          - "node.role==worker"
      # mandatory label
      labels:
        - "bunkerized-nginx.AUTOCONF"

  autoconf:
    image: bunkerity/bunkerized-nginx-autoconf
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - cache-vol:/cache
      - certs-vol:/etc/letsencrypt
    environment:
      - SWARM_MODE=yes
      - API_URI=/ChangeMeToSomethingHardToGuess # must match API_URI from nginx
    networks:
      - bunkerized-net
    deploy:
      replicas: 1
      placement:
```

(continues on next page)

(continued from previous page)

```
        constraints:
          - "node.role==manager"

# This will create the networks for you
networks:
  bunkerized-net:
    driver: overlay
    attachable: true
    name: bunkerized-net
  services-net:
    driver: overlay
    attachable: true
    name: services-net
# And the volumes too
volumes:
  cache-vol:
  certs-vol:
```

Check the logs of both autoconf and bunkerized-nginx services to see if everything is working as expected.

You can now create a new service and add environment variables as labels with the `bunkerized-nginx.` prefix to let the autoconf service "automagically" do the configuration for you :

```
$ docker service create \
         --name myservice \
         --constraint node.role==worker \
         --network services-net \
         -l bunkerized-nginx.SERVER_NAME=www.example.com \
         -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
         -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
         -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice \
         tutum/hello-world
```

docker-compose equivalent :

```
version: "3"

services:

  myservice:
    image: tutum/hello-world
    networks:
      - services-net
    deploy:
      placement:
        constraints:
          - "node.role==worker"
      labels:
        - "bunkerized-nginx.SERVER_NAME=www.example.com"
        - "bunkerized-nginx.USE_REVERSE_PROXY=yes"
        - "bunkerized-nginx.REVERSE_PROXY_URL=/"
        - "bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice"
```

(continues on next page)

(continued from previous page)

```
networks:
  services-net:
    external:
      name: services-net
```

Please note that if you want to override the `AUTO_LETS_ENCRYPT=yes` previously defined in the bunkerized-nginx service, you simply need to add the `bunkerized-nginx.AUTO_LETS_ENCRYPT=no` label.

Look at the logs of both autoconf and bunkerized-nginx to check if the configuration has been generated and loaded by bunkerized-nginx. You should now be able to visit http(s)://www.example.com.

When your service is not needed anymore, you can delete it as usual. The autoconf should get the event and generate the configuration again.

2.4 Kubernetes

This integration is still in beta, please fill an issue if you find a bug or have an idea on how to improve it.

The bunkerized-nginx-autoconf acts as an Ingress Controller and connects to the k8s API to get cluster events and generate a new configuration when it's needed. Once the configuration is generated, the Ingress Controller sends the configuration files and a reload order to the bunkerized-nginx instances running in the cluster. If you need to deliver static files (e.g., html, images, css, js, ...) a shared folder accessible from all bunkerized-nginx instances is needed (you can use a storage system like NFS, GlusterFS, CephFS on the host or a Kubernetes Volume that supports ReadOnlyMany access).

The first step to do is to declare the RBAC authorization that will be used by the Ingress Controller to access the Kubernetes API. A ready-to-use declaration is available here :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: bunkerized-nginx-ingress-controller
rules:
- apiGroups: [""]
  resources: ["services", "pods"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["extensions"]
  resources: ["ingresses"]
  verbs: ["get", "watch", "list"]
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bunkerized-nginx-ingress-controller
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: bunkerized-nginx-ingress-controller
subjects:
- kind: ServiceAccount
  name: bunkerized-nginx-ingress-controller
```

(continues on next page)

(continued from previous page)

```
  namespace: default
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: bunkerized-nginx-ingress-controller
  apiGroup: rbac.authorization.k8s.io
```

Next, you can deploy bunkerized-nginx as a DaemonSet :

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bunkerized-nginx
  labels:
    app: bunkerized-nginx
spec:
  selector:
    matchLabels:
      name: bunkerized-nginx
  template:
    metadata:
      labels:
        name: bunkerized-nginx
        # this label is mandatory
        bunkerized-nginx: "yes"
    spec:
      containers:
      - name: bunkerized-nginx
        image: bunkerity/bunkerized-nginx
        ports:
        - containerPort: 8080
          hostPort: 80
        - containerPort: 8443
          hostPort: 443
        env:
        - name: KUBERNETES_MODE
          value: "yes"
        - name: DNS_RESOLVERS
          value: "coredns.kube-system.svc.cluster.local"
        - name: USE_API
          value: "yes"
        - name: API_URI
          value: "/ChangeMeToSomethingHardToGuess"
        - name: SERVER_NAME
          value: ""
        - name: MULTISITE
          value: "yes"
---
apiVersion: v1
kind: Service
metadata:
  name: bunkerized-nginx-service
```

(continues on next page)

(continued from previous page)

```
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  # this annotation is mandatory
  annotations:
    bunkerized-nginx.AUTOCONF: "yes"
spec:
  clusterIP: None
  selector:
    name: bunkerized-nginx
```

You can now deploy the autoconf which will act as the ingress controller :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-nginx
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bunkerized-nginx-ingress-controller
  labels:
    app: bunkerized-nginx-autoconf
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bunkerized-nginx-autoconf
  template:
    metadata:
      labels:
        app: bunkerized-nginx-autoconf
    spec:
      serviceAccountName: bunkerized-nginx-ingress-controller
      volumes:
      - name: vol-nginx
        persistentVolumeClaim:
          claimName: pvc-nginx
      initContainers:
      - name: change-data-dir-permissions
        command:
        - chown
        - -R
        - 101:101
        - /etc/letsencrypt
```

(continues on next page)

(continued from previous page)

```
        - /cache
        image: busybox
        volumeMounts:
        - name: vol-nginx
          mountPath: /etc/letsencrypt
          subPath: letsencrypt
        - name: vol-nginx
          mountPath: /cache
          subPath: cache
        securityContext:
          runAsNonRoot: false
          runAsUser: 0
          runAsGroup: 0
      containers:
      - name: bunkerized-nginx-autoconf
        image: bunkerity/bunkerized-nginx-autoconf
        env:
        - name: KUBERNETES_MODE
          value: "yes"
        - name: API_URI
          value: "/ChangeMeToSomethingHardToGuess"
        volumeMounts:
        - name: vol-nginx
          mountPath: /etc/letsencrypt
          subPath: letsencrypt
        - name: vol-nginx
          mountPath: /cache
          subPath: cache
```

Check the logs of both bunkerized-nginx and autoconf deployments to see if everything is working as expected.

You can now deploy your web service and make it accessible from within the cluster :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    app: myapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: containous/whoami
---
```

(continues on next page)

(continued from previous page)

```
apiVersion: v1
kind: Service
metadata:
  name: myapp
spec:
  type: ClusterIP
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Last but not least, it's time to define your Ingress resource to make your web service publicly available :

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bunkerized-nginx-ingress
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    # add any global and default environment variables here as annotations with the
↪"bunkerized-nginx." prefix
    # examples :
    #bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    #bunkerized-nginx.USE_ANTIBOT: "javascript"
    #bunkerized-nginx.REDIRECT_HTTP_TO_HTTPS: "yes"
    #bunkerized-nginx.www.example.com_REVERSE_PROXY_WS: "yes"
    #bunkerized-nginx.www.example.com_USE_MODSECURITY: "no"
spec:
  tls:
  - hosts:
    - www.example.com
  rules:
  - host: "www.example.com"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: myapp
            port:
              number: 80
```

Check the logs to see if the configuration has been generated and bunkerized-nginx reloaded. You should be able to visit http(s)://www.example.com.

Note that an alternative would be to add annotations directly to your services (a common use-case is for PHP applications because the Ingress resource is only for reverse proxy) without editing the Ingress resource :

```
apiVersion: v1
kind: Service
metadata:
  name: myapp
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    bunkerized-nginx.SERVER_NAME: "www.example.com"
    bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    bunkerized-nginx.USE_REVERSE_PROXY: "yes"
    bunkerized-nginx.REVERSE_PROXY_URL: "/"
    bunkerized-nginx.REVERSE_PROXY_HOST: "http://myapp.default.svc.cluster.local"
spec:
  type: ClusterIP
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2.5 Linux

This integration is still in beta, please fill an issue if you find a bug or have an idea on how to improve it.

List of supported Linux distributions :

- Debian buster (10)
- Ubuntu focal (20.04)
- CentOS 7
- Fedora 34
- Arch Linux

Unlike containers, Linux integration can be tedious because bunkerized-nginx has a bunch of dependencies that need to be installed before we can use it. Fortunately, we provide a helper script to make the process easier and automatic. Once installed, the configuration is really simple, all you have to do is to edit the `/opt/bunkerized-nginx/variables.env` configuration file and run the `bunkerized-nginx` command to apply it.

First of all you will need to install bunkerized-nginx. The recommended way is to use the official installer script :

```
$ curl -fsSL https://github.com/bunkerity/bunkerized-nginx/releases/download/v1.3.2/
↪linux-install.sh -o /tmp/bunkerized-nginx.sh
```

Before executing it, you should also check the signature :

```
$ curl -fsSL https://github.com/bunkerity/bunkerized-nginx/releases/download/v1.3.2/
↪linux-install.sh.asc -o /tmp/bunkerized-nginx.sh.asc
$ gpg --auto-key-locate hkps://keys.openpgp.org --locate-keys contact@bunkerity.com
$ gpg --verify /tmp/bunkerized-nginx.sh.asc /tmp/bunkerized-nginx.sh
```

You can now install bunkerized-nginx (and take a coffee because it may take a while) :

```
$ chmod +x /tmp/bunkerized-nginx.sh
$ /tmp/bunkerized-nginx.sh
```

To demonstrate the configuration on Linux, we will create a simple "Hello World" static file that will be served by bunkerized-nginx.

Static files are stored inside the `/opt/bunkerized-nginx/www` folder and the unprivileged nginx user must have read access on it :

```
$ echo "Hello bunkerized World !" > /opt/bunkerized-nginx/www/index.html
$ chown root:nginx /opt/bunkerized-nginx/www/index.html
$ chmod 740 /opt/bunkerized-nginx/www/index.html
```

Here is the example configuration file that needs to be written at `/opt/bunkerized-nginx/variables.env` :

```
HTTP_PORT=80
HTTPS_PORT=443
DNS_RESOLVERS=8.8.8.8 8.8.4.4
SERVER_NAME=www.example.com
AUTO_LETS_ENCRYPT=yes
```

Important things to note :

- Replace www.example.com with your own domain (it must points to your server IP address if you want Let's Encrypt to work)
- Automatic Let's Encrypt is enabled thanks to `AUTO_LETS_ENCRYPT=yes` (since the default is `AUTO_LETS_ENCRYPT=no` you can remove the environment variable to disable Let's Encrypt)
- The default values for `HTTP_PORT` and `HTTPS_PORT` are `8080` and `8443` hence the explicit declaration with standard ports values
- Replace the `DNS_RESOLVERS` value with your own DNS resolver(s) if you need nginx to resolve internal DNS requests (e.g., reverse proxy to an internal service)

You can now apply the configuration by running the **bunkerized-nginx** command :

```
$ bunkerized-nginx
```

Visit http(s)://www.example.com to confirm that everything is working as expected.

CHAPTER

THREE

QUICKSTART GUIDE

3.1 Reverse proxy

The following environment variables can be used to deploy bunkerized-nginx as a reverse proxy in front of your web services :

- `USE_REVERSE_PROXY` : activate/deactivate the reverse proxy mode
- `REVERSE_PROXY_URL` : public path prefix
- `REVERSE_PROXY_HOST` : full address of the proxied service

Here is a basic example :

```
SERVER_NAME=www.example.com
USE_REVERSE_PROXY=yes
REVERSE_PROXY_URL=/
REVERSE_PROXY_HOST=http://my-service.example.local:8080
```

If you have multiple web services you can configure multiple reverse proxy rules by appending a number to the environment variables names :

```
SERVER_NAME=www.example.com
USE_REVERSE_PROXY=yes
REVERSE_PROXY_URL_1=/app1
REVERSE_PROXY_HOST_1=http://app1.example.local:8080
REVERSE_PROXY_URL_2=/app2
REVERSE_PROXY_HOST_2=http://app2.example.local:8080
```

3.1.1 Docker

When using Docker, the recommended way is to create a network so bunkerized-nginx can communicate with the web service using the container name :

```
$ docker network create services-net
$ docker run -d \
         --name myservice \
         --network services-net \
         tutum/hello-world
$ docker run -d \
         --network services-net \
```

(continues on next page)

(continued from previous page)

```
        -p 80:8080 \
        -p 443:8443 \
        -v "${PWD}/certs:/etc/letsencrypt" \
        -e SERVER_NAME=www.example.com \
        -e AUTO_LETS_ENCRYPT=yes \
        -e USE_REVERSE_PROXY=yes \
        -e REVERSE_PROXY_URL=/ \
        -e REVERSE_PROXY_HOST=http://myservice \
        bunkerity/bunkerized-nginx
```

docker-compose equivalent :

```
version: '3'

services:

  mybunkerized:
    image: bunkerity/bunkerized-nginx
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./certs:/etc/letsencrypt
    environment:
      - SERVER_NAME=www.example.com
      - AUTO_LETS_ENCRYPT=yes
      - USE_REVERSE_PROXY=yes
      - REVERSE_PROXY_URL=/
      - REVERSE_PROXY_HOST=http://myservice
    networks:
      - services-net
    depends_on:
      - myservice

  myservice:
    image: tutum/hello-world
    networks:
      - services-net

networks:
  services-net:
```

3.1.2 Docker autoconf

When the Docker autoconf stack is running, you simply need to start the container hosting your web service and add the environment variables as labels :

```
$ docker run -d \
         --name myservice \
         --network services-net \
         -l bunkerized-nginx.SERVER_NAME=www.example.com \
         -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
         -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
         -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice \
         tutum/hello-world
```

docker-compose equivalent :

```
version: '3'

services:

  myservice:
    image: tutum/hello-world
    networks:
      services-net:
        aliases:
          - myservice
    labels:
      - bunkerized-nginx.SERVER_NAME=www.example.com
      - bunkerized-nginx.USE_REVERSE_PROXY=yes
      - bunkerized-nginx.REVERSE_PROXY_URL=/
      - bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice

networks:
  services-net:
    external:
      name: services-net
```

3.1.3 Docker Swarm

When the Docker Swarm stack is running, you simply need to start the Swarm service hosting your web service and add the environment variables as labels :

```
$ docker service create \
         --name myservice \
         --network services-net \
         --constraint node.role==worker \
         -l bunkerized-nginx.SERVER_NAME=www.example.com \
         -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
         -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
         -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice \
         tutum/hello-world
```

docker-compose equivalent :

```
version: '3'

services:

  myservice:
    image: tutum/hello-world
    networks:
      services-net:
        aliases:
          - myservice
    deploy:
      placement:
        constraints:
          - "node.role==worker"
      labels:
        - bunkerized-nginx.SERVER_NAME=www.example.com
        - bunkerized-nginx.USE_REVERSE_PROXY=yes
        - bunkerized-nginx.REVERSE_PROXY_URL=/
        - bunkerized-nginx.REVERSE_PROXY_HOST=http://myservice

networks:
  services-net:
    external:
      name: services-net
```

3.1.4 Kubernetes

Example deployment and service declaration :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myservice
  labels:
    app: myservice
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myservice
  template:
    metadata:
      labels:
        app: myservice
    spec:
      containers:
      - name: myservice
        image: tutum/hello-world
---
apiVersion: v1
kind: Service
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: myservice
spec:
  type: ClusterIP
  selector:
    app: myservice
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

The most straightforward way to add a reverse proxy in the Kubernetes cluster is to declare it in the Ingress resource :

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: bunkerized-nginx-ingress
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
spec:
  tls:
  - hosts:
    - www.example.com
  rules:
  - host: "www.example.com"
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: myservice
            port:
              number: 80
```

An alternative "hackish" way is to use environment variables as annotations prefixed with "bunkerized-nginx." inside the Service resource of your web service :

```
apiVersion: v1
kind: Service
metadata:
  name: myservice
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    bunkerized-nginx.SERVER_NAME: "www.example.com"
    bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    bunkerized-nginx.USE_REVERSE_PROXY: "yes"
    bunkerized-nginx.REVERSE_PROXY_URL: "/"
    bunkerized-nginx.REVERSE_PROXY_HOST: "http://myservice.default.svc.cluster.local"
```

(continues on next page)

(continued from previous page)

```
spec:
  type: ClusterIP
  selector:
    app: myservice
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

3.1.5 Linux

Example of a basic configuration file :

```
HTTP_PORT=80
HTTPS_PORT=443
DNS_RESOLVERS=8.8.8.8 8.8.4.4
SERVER_NAME=www.example.com
AUTO_LETS_ENCRYPT=yes
USE_REVERSE_PROXY=yes
REVERSE_PROXY_URL=/
# Local proxied application
REVERSE_PROXY_HOST=http://127.0.0.1:8080
# Remote proxied application
#REVERSE_PROXY_HOST=http://service.example.local:8080
```

3.2 PHP applications

The following environment variables can be used to configure bunkerized-nginx in front of PHP-FPM web applications :

- `REMOTE_PHP` : host/ip of a remote PHP-FPM instance
- `REMOTE_PHP_PATH` : absolute path containing the PHP files (from the remote instance perspective)
- `LOCAL_PHP` : absolute path of the local unix socket used by a local PHP-FPM instance
- `LOCAL_PHP_PATH` : absolute path containing the PHP files (when using local instance)

Here is a basic example with a remote instance :

```
SERVER_NAME=www.example.com
REMOTE_PHP=my-php.example.local
REMOTE_PHP_PATH=/var/www/html
```

And another example with a local instance :

```
SERVER_NAME=www.example.com
LOCAL_PHP=/var/run/php7-fpm.sock
LOCAL_PHP_PATH=/opt/bunkerized-nginx/www
```

3.2.1 Docker

When using Docker, the recommended way is to create a network so bunkerized-nginx can communicate with the PHP-FPM instance using the container name :

```
$ docker network create services-net
$ docker run -d \
         --name myservice \
         --network services-net \
         -v "${PWD}/www:/app" \
         php:fpm
$ docker run -d \
         --network services-net \
         -p 80:8080 \
         -p 443:8443 \
         -v "${PWD}/www:/www:ro" \
         -v "${PWD}/certs:/etc/letsencrypt" \
         -e SERVER_NAME=www.example.com \
         -e AUTO_LETS_ENCRYPT=yes \
         -e REMOTE_PHP=myservice \
         -e REMOTE_PHP_PATH=/app \
         bunkerity/bunkerized-nginx
```

docker-compose equivalent :

```
version: '3'

services:

  mybunkerized:
    image: bunkerity/bunkerized-nginx
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./www:/www:ro
      - ./certs:/etc/letsencrypt
    environment:
      - SERVER_NAME=www.example.com
      - AUTO_LETS_ENCRYPT=yes
      - REMOTE_PHP=myservice
      - REMOTE_PHP_PATH=/app
    networks:
      - services-net
    depends_on:
      - myservice

  myservice:
    image: php:fpm
    networks:
      - services-net
    volumes:
      - ./www:/app
```

(continues on next page)

(continued from previous page)

```
networks:
  services-net:
```

3.2.2 Docker autoconf

When the Docker autoconf stack is running, you simply need to start the container hosting your PHP-FPM instance and add the environment variables as labels :

```
$ docker run -d \
         --name myservice \
         --network services-net \
         -v "${PWD}/www/www.example.com:/app" \
         -l bunkerized-nginx.SERVER_NAME=www.example.com \
         -l bunkerized-nginx.REMOTE_PHP=myservice \
         -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
         php:fpm
```

```
version: '3'

services:

  myservice:
    image: php:fpm
    volumes:
      - ./www/www.example.com:/app
    networks:
      services-net:
        aliases:
          - myservice
    labels:
      - bunkerized-nginx.SERVER_NAME=www.example.com
      - bunkerized-nginx.REMOTE_PHP=myservice
      - bunkerized-nginx.REMOTE_PHP_PATH=/app

networks:
  services-net:
    external:
      name: services-net
```

3.2.3 Docker Swarm

When the Docker Swarm stack is running, you simply need to start the Swarm service hosting your PHP-FPM instance and add the environment variables as labels :

```
$ docker service create \
         --name myservice \
         --constraint node.role==worker \
         --network services-net \
         --mount type=bind,source=/shared/www/www.example.com,destination=/app \
```

(continues on next page)

(continued from previous page)

```
        -l bunkerized-nginx.SERVER_NAME=www.example.com \
        -l bunkerized-nginx.REMOTE_PHP=myservice \
        -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
        php:fpm
```

docker-compose equivalent :

```
version: "3"

services:

  myservice:
    image: php:fpm
    networks:
      services-net:
        aliases:
          - myservice
    volumes:
      - /shared/www/www.example.com:/app
    deploy:
      placement:
        constraints:
          - "node.role==worker"
      labels:
        - "bunkerized-nginx.SERVER_NAME=www.example.com"
        - "bunkerized-nginx.REMOTE_PHP=myservice"
        - "bunkerized-nginx.REMOTE_PHP_PATH=/app"

networks:
  services-net:
    external:
      name: services-net
```

3.2.4 Kubernetes

You need to use environment variables as annotations prefixed with `bunkerized-nginx.` inside the Service resource of your PHP-FPM instance :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myservice
  labels:
    app: myservice
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myservice
  template:
    metadata:
```

(continues on next page)

(continued from previous page)

```
      labels:
        app: myservice
    spec:
      containers:
      - name: myservice
        image: php:fpm
        volumeMounts:
        - name: php-files
          mountPath: /app
      volumes:
      - name: php-files
        hostPath:
          path: /shared/www/www.example.com
          type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: myservice
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    bunkerized-nginx.SERVER_NAME: "www.example.com"
    bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    bunkerized-nginx.REMOTE_PHP: "myservice.default.svc.cluster.local"
    bunkerized-nginx.REMOTE_PHP_PATH: "/app"
spec:
  type: ClusterIP
  selector:
    app: myservice
  ports:
    - protocol: TCP
      port: 9000
      targetPort: 9000
```

3.2.5 Linux

Example of a basic configuration file :

```
HTTP_PORT=80
HTTPS_PORT=443
DNS_RESOLVERS=8.8.8.8 8.8.4.4
SERVER_NAME=www.example.com
AUTO_LETS_ENCRYPT=yes
# Case 1 : the PHP-FPM instance is on the same machine
# you just need to adjust the socket path
LOCAL_PHP=/run/php/php7.3-fpm.sock
LOCAL_PHP_PATH=/opt/bunkerized-nginx/www
# Case 2 : the PHP-FPM instance is on another machine
#REMOTE_PHP=myapp.example.local
```

(continues on next page)

(continued from previous page)

```
#REMOTE_PHP_PATH=/app
```

Don't forget that bunkerized-nginx runs as an unprivileged user/group both named `nginx`. When using a local PHP-FPM instance, you will need to take care of the rights and permissions of the socket and web files.

For example, if your PHP-FPM is running as the `www-data` user, you can create a new group called `web-users` and add `nginx` and `www-data` into it :

```
$ groupadd web-users
$ usermod -a -G web-users nginx
$ usermod -a -G web-users www-data
```

Once it's done, you will need to tweak your PHP-FPM configuration file (e.g., `/etc/php/7.3/fpm/pool.d/www.conf`) to edit the default group of the processes and the permissions of the socket file :

```
[www]
...
user = www-data
group = web-users
...
listen = /run/php/php7.3-fpm.sock
listen.owner = www-data
listen.group = web-users
listen.mode = 0660
...
```

Last but not least, you will need to edit the permissions of `/opt/bunkerized-nginx/www` to make sure that nginx can read and PHP-FPM can write (in case your PHP app needs it) :

```
$ chown root:web-users /opt/bunkerized-nginx/www
$ chmod 750 /opt/bunkerized-nginx/www
$ find /opt/bunkerized-nginx/www/* -exec chown www-data:nginx {} \;
$ find /opt/bunkerized-nginx/www/* -type f -exec chmod 740 {} \;
$ find /opt/bunkerized-nginx/www/* -type d -exec chmod 750 {} \;
```

3.3 Multisite

If you have multiple services to protect, the easiest way to do it is by enabling the "multisite" mode. When using multisite, bunkerized-nginx will create one server block per server defined in the `SERVER_NAME` environment variable. You can configure each servers independently by adding the server name as a prefix.

Here is an example :

```
SERVER_NAME=app1.example.com app2.example.com
MULTISITE=yes
app1.example.com_USE_REVERSE_PROXY=yes
app1.example.com_REVERSE_PROXY_URL=/
app1.example.com_REVERSE_PROXY_HOST=http://app1.example.local:8080
app2.example.com_REMOTE_PHP=app2.example.local
app2.example.com_REMOTE_PHP_PATH=/var/www/html
```

When using the multisite mode, some special folders must have a specific structure with subfolders named the same as the servers defined in the SERVER_NAME environment variable. Let's take the **app2.example.com** as an example : if some static files need to be served by nginx, you need to place them under **www/app2.example.com**.

3.3.1 Docker

When using Docker, the recommended way is to create a network so bunkerized-nginx can communicate with the web services using the container name :

```
$ docker network create services-net
$ docker run -d \
         --name myapp1 \
         --network services-net \
         tutum/hello-world
$ docker run -d \
         --name myapp2 \
         --network services-net \
         -v "${PWD}/www/app2.example.com:/app" \
         php:fpm
$ docker run -d \
         --network services-net \
         -p 80:8080 \
         -p 443:8443 \
         -v "${PWD}/www:/www:ro" \
         -v "${PWD}/certs:/etc/letsencrypt" \
         -e "SERVER_NAME=app1.example.com app2.example.com" \
         -e MULTISITE=yes \
         -e AUTO_LETS_ENCRYPT=yes \
         -e app1.example.com_USE_REVERSE_PROXY=yes \
         -e app1.example.com_REVERSE_PROXY_URL=/ \
         -e app1.example.com_REVERSE_PROXY_HOST=http://myapp1 \
         -e app2.example.com_REMOTE_PHP=myapp2 \
         -e app2.example.com_REMOTE_PHP_PATH=/app \
         bunkerity/bunkerized-nginx
```

docker-compose equivalent :

```
version: '3'

services:

  mybunkerized:
    image: bunkerity/bunkerized-nginx
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./www:/www:ro
      - ./certs:/etc/letsencrypt
    environment:
      - SERVER_NAME=app1.example.com app2.example.com
      - MULTISITE=yes
      - AUTO_LETS_ENCRYPT=yes
```

(continues on next page)

(continued from previous page)

```
      - app1.example.com_USE_REVERSE_PROXY=yes
      - app1.example.com_REVERSE_PROXY_URL=/
      - app1.example.com_REVERSE_PROXY_HOST=http://myapp1
      - app2.example.com_REMOTE_PHP=myapp2
      - app2.example.com_REMOTE_PHP_PATH=/app
    networks:
      - services-net
    depends_on:
      - myapp1
      - myapp2

  myapp1:
    image: tutum/hello-world
    networks:
      - services-net

  myapp2:
    image: php:fpm
    volumes:
      - ./www/app2.example.com:/app
    networks:
      - services-net

networks:
  services-net:
```

3.3.2 Docker autoconf

The multisite feature must be activated when using the Docker autoconf integration.

When the Docker autoconf stack is running, you simply need to start the containers hosting your web services and add the environment variables as labels :

```
$ docker run -d \
         --name myapp1 \
         --network services-net \
         -l bunkerized-nginx.SERVER_NAME=app1.example.com \
         -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
         -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
         -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myapp1 \
         tutum/hello-world
$ docker run -d \
         --name myapp2 \
         --network services-net \
         -v "${PWD}/www/app2.example.com:/app" \
         -l bunkerized-nginx.SERVER_NAME=app2.example.com \
         -l bunkerized-nginx.REMOTE_PHP=myapp2 \
         -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
         php:fpm
```

docker-compose equivalent :

```
version: '3'

services:

  myapp1:
    image: tutum/hello-world
    networks:
      services-net:
        aliases:
          - myapp1
    labels:
      - bunkerized-nginx.SERVER_NAME=app1.example.com
      - bunkerized-nginx.USE_REVERSE_PROXY=yes
      - bunkerized-nginx.REVERSE_PROXY_URL=/
      - bunkerized-nginx.REVERSE_PROXY_HOST=http://myapp1

  myapp2:
    image: php:fpm
    networks:
      services-net:
        aliases:
          - myapp2
    volumes:
      - ./www/app2.example.com:/app
    labels:
      - bunkerized-nginx.SERVER_NAME=app2.example.com
      - bunkerized-nginx.REMOTE_PHP=myapp2
      - bunkerized-nginx.REMOTE_PHP_PATH=/app

networks:
  services-net:
    external:
      name: services-net
```

3.3.3 Docker Swarm

The multisite feature must be activated when using the Docker Swarm integration.

When the Docker Swarm stack is running, you simply need to start the Swarm service hosting your web services and add the environment variables as labels :

```
$ docker service create \
        --name myapp1 \
        --network services-net \
        --constraint node.role==worker \
        -l bunkerized-nginx.SERVER_NAME=app1.example.com \
        -l bunkerized-nginx.USE_REVERSE_PROXY=yes \
        -l bunkerized-nginx.REVERSE_PROXY_URL=/ \
        -l bunkerized-nginx.REVERSE_PROXY_HOST=http://myapp1 \
        tutum/hello-world
$ docker service create \
        --name myapp2 \
```

(continues on next page)

(continued from previous page)

```
        --constraint node.role==worker \
        --network services-net \
        --mount type=bind,source=/shared/www/app2.example.com,destination=/app \
        -l bunkerized-nginx.SERVER_NAME=app2.example.com \
        -l bunkerized-nginx.REMOTE_PHP=myapp2 \
        -l bunkerized-nginx.REMOTE_PHP_PATH=/app \
        php:fpm
```

docker-compose equivalent :

```
version: "3"

services:

  myapp1:
    image: tutum/hello-world
    networks:
      services-net:
        aliases:
          - myapp1
    deploy:
      placement:
        constraints:
          - "node.role==worker"
      labels:
        - bunkerized-nginx.SERVER_NAME=app1.example.com
        - bunkerized-nginx.USE_REVERSE_PROXY=yes
        - bunkerized-nginx.REVERSE_PROXY_URL=/
        - bunkerized-nginx.REVERSE_PROXY_HOST=http://myapp1

  myapp2:
    image: php:fpm
    networks:
      services-net:
        aliases:
          - myapp2
    volumes:
      - /shared/www/app2.example.com:/app
    deploy:
      placement:
        constraints:
          - "node.role==worker"
      labels:
        - "bunkerized-nginx.SERVER_NAME=app2.example.com"
        - "bunkerized-nginx.REMOTE_PHP=myapp2"
        - "bunkerized-nginx.REMOTE_PHP_PATH=/app"

networks:
  services-net:
    external:
      name: services-net
```

3.3.4 Kubernetes

The multisite feature must be activated when using the Kubernetes integration.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp1
  labels:
    app: myapp1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp1
  template:
    metadata:
      labels:
        app: myapp1
    spec:
      containers:
      - name: myapp1
        image: tutum/hello-world
---
apiVersion: v1
kind: Service
metadata:
  name: myapp1
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    bunkerized-nginx.SERVER_NAME: "app1.example.com"
    bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    bunkerized-nginx.USE_REVERSE_PROXY: "yes"
    bunkerized-nginx.REVERSE_PROXY_URL: "/"
    bunkerized-nginx.REVERSE_PROXY_HOST: "http://myapp1.default.svc.cluster.local"
spec:
  type: ClusterIP
  selector:
    app: myapp1
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp2
  labels:
    app: myapp2
spec:
```

(continues on next page)

(continued from previous page)

```
  replicas: 1
  selector:
    matchLabels:
      app: myapp2
  template:
    metadata:
      labels:
        app: myapp2
    spec:
      containers:
      - name: myapp2
        image: php:fpm
        volumeMounts:
        - name: php-files
          mountPath: /app
      volumes:
      - name: php-files
        hostPath:
          path: /shared/www/app2.example.com
          type: Directory
---
apiVersion: v1
kind: Service
metadata:
  name: myapp2
  # this label is mandatory
  labels:
    bunkerized-nginx: "yes"
  annotations:
    bunkerized-nginx.SERVER_NAME: "app2.example.com"
    bunkerized-nginx.AUTO_LETS_ENCRYPT: "yes"
    bunkerized-nginx.REMOTE_PHP: "myapp2.default.svc.cluster.local"
    bunkerized-nginx.REMOTE_PHP_PATH: "/app"
spec:
  type: ClusterIP
  selector:
    app: myapp2
  ports:
    - protocol: TCP
      port: 9000
      targetPort: 9000
```

3.3.5 Linux

Example of a basic configuration file :

```
HTTP_PORT=80
HTTPS_PORT=443
DNS_RESOLVERS=8.8.8.8 8.8.4.4
SERVER_NAME=app1.example.com app2.example.com
MULTISITE=yes
AUTO_LETS_ENCRYPT=yes
app1.example.com_USE_REVERSE_PROXY=yes
app1.example.com_REVERSE_PROXY_URL=/
# Local proxied application
app1.example.com_REVERSE_PROXY_HOST=http://127.0.0.1:8080
# Remote proxied application
#app1.example.com_REVERSE_PROXY_HOST=http://service.example.local:8080
# If the PHP-FPM instance is on the same machine
# you just need to adjust the socket path
app2.example.com_LOCAL_PHP=/run/php/php7.3-fpm.sock
app2.example.com_LOCAL_PHP_PATH=/opt/bunkerized-nginx/www/app2.example.com
# Else if the PHP-FPM instance is on another machine
#app2.example.com_REMOTE_PHP=myapp.example.local
#app2.example.com_REMOTE_PHP_PATH=/app
```

Don't forget that bunkerized-nginx runs as an unprivileged user/group both named `nginx`. When using a local PHP-FPM instance, you will need to take care of the rights and permissions of the socket and web files.

See the Linux section of PHP application for more information.

CHAPTER

FOUR

SPECIAL FOLDERS

Please note that bunkerized-nginx runs as an unprivileged user (UID/GID 101 when using the Docker image) and you should set the rights on the host accordingly to the files and folders on your host.

4.1 Multisite

When the special folder "supports" the multisite mode, you can create subfolders named as the server names used in the configuration. When doing it only the subfolder files will be "used" by the corresponding web service.

4.2 Web files

This special folder is used by bunkerized-nginx to deliver static files. The typical use case is when you have a PHP application that also contains static assets like CSS, JS and images.

Location (container) : `/www`
Location (Linux) : `/opt/bunkerized-nginx/www`
Multisite : `yes`
Read-only : `yes`

Examples :

- Basic website with PHP
- Multisite basic

4.3 http configurations

This special folder contains .conf files that will be loaded by nginx at http context. The typical use case is when you need to add custom directives into the `http { }` block of nginx.

Location (container) : `/http-confs`
Location (Linux) : `/opt/bunkerized-nginx/http-confs`
Multisite : `no`
Read-only : `yes`

Examples :

- Load balancer

4.4 server configurations

This special folder contains .conf files that will be loaded by nginx at server context. The typical use case is when you need to add custom directives into the `server { }` block of nginx.

Location (container) : `/server-confs`
Location (Linux) : `/opt/bunkerized-nginx/server-confs`
Multisite : `yes`
Read-only : `yes`

Examples :

- Wordpress
- Multisite custom confs

4.5 ModSecurity configurations

This special folder contains .conf files that will be loaded by ModSecurity after the OWASP Core Rule Set is loaded. The typical use case is to edit loaded CRS rules to avoid false positives.

Location (container) : `/modsec-confs`
Location (Linux) : `/opt/bunkerized-nginx/modsec-confs`
Multisite : `yes`
Read-only : `yes`

Examples :

- Wordpress
- Multisite custom confs

4.6 CRS configurations

This special folder contains .conf file that will be loaded by ModSecurity before the OWASP Core Rule Set is loaded. The typical use case is when you want to specify exclusions for the CRS.

Location (container) : `/modsec-crs-confs`
Location (Linux) : `/opt/bunkerized-nginx/modsec-crs-confs`
Multisite : `yes`
Read-only : `yes`

Examples :

- Wordpress
- Multisite custom confs

4.7 Cache

This special folder is used to cache some data like blacklists and avoid downloading them again if it is not necessary. The typical use case is to avoid the overhead when you are testing bunkerized-nginx in a container and you have to recreate it multiple times.

Location (container) : `/cache`
Location (Linux) : `/opt/bunkerized-nginx/cache`
Multisite : `no`
Read-only : `no`

4.8 Plugins

This special folder is the placeholder for the plugins loaded by bunkerized-nginx. See the plugins section for more information.

Location (container) : `/plugins`
Location (Linux) : `/opt/bunkerized-nginx/plugins`
Multisite : `no`
Read-only : `no`

4.9 ACME challenge

This special folder is used as the web root for Let's Encrypt challenges. The typical use case is to share the same folder when you are using bunkerized-nginx in a clustered environment like Docker Swarm or Kubernetes.

Location (container) : `/acme-challenge`
Location (Linux) : `/opt/bunkerized-nginx/acme-challenge`
Multisite : `no`
Read-only : `no`

CHAPTER

FIVE

SECURITY TUNING

bunkerized-nginx comes with a set of predefined security settings that you can (and you should) tune to meet your own use case.

5.1 Miscellaneous

Here is a list of miscellaneous environment variables related more or less to security :

- `MAX_CLIENT_SIZE=10m` : maximum size of client body
- `ALLOWED_METHODS=GET|POST|HEAD` : list of HTTP methods that clients are allowed to use
- `DISABLE_DEFAULT_SERVER=no` : enable/disable the default server (i.e. : should your server respond to unknown Host header ?)
- `SERVER_TOKENS=off` : enable/disable sending the version number of nginx

5.2 HTTPS

5.2.1 Settings

Here is a list of environment variables and the corresponding default value related to HTTPS :

- `LISTEN_HTTP=yes` : you can set it to `no` if you want to disable HTTP access
- `REDIRECT_HTTP_TO_HTTPS=no` : enable/disable HTTP to HTTPS redirection
- `HTTPS_PROTOCOLS=TLSv1.2 TLSv1.3` : list of TLS versions to use
- `HTTP2=yes` : enable/disable HTTP2 when HTTPS is enabled
- `COOKIE_AUTO_SECURE_FLAG=yes` : enable/disable adding Secure flag when HTTPS is enabled
- `STRICT_TRANSPORT_SECURITY=max-age=31536000` : force users to visit the website in HTTPS (more info here)

5.2.2 Let's Encrypt

Using Let's Encrypt with the `AUTO_LETS_ENCRYPT=yes` environment variable is the easiest way to add HTTPS supports to your web services if they are connected to internet and you have public DNS A record(s).

You can also set the `EMAIL_LETS_ENCRYPT` environment variable if you want to receive notifications from Let's Encrypt like expiration alerts.

5.2.3 Custom certificate(s)

If you have security constraints (e.g., local network, custom PKI, ...) you can use custom certificates of your choice and tell bunkerized-nginx to use them with the following environment variables :

- `USE_CUSTOM_HTTPS=yes`
- `CUSTOM_HTTPS_CERT=/path/inside/container/to/cert.pem`
- `CUSTOM_HTTPS_KEY=/path/inside/container/to/key.pem`

Here is a an example on how to use custom certificates :

```
$ ls /etc/ssl/my-web-app
cert.pem key.pem

$ docker run -p 80:8080 \
             -p 443:8443 \
             -v /etc/ssl/my-web-app:/certs:ro \
             -e USE_CUSTOM_HTTPS=yes \
             -e CUSTOM_HTTPS_CERT=/certs/cert.pem \
             -e CUSTOM_HTTPS_KEY=/certs/key.pem \
             ...
             bunkerity/bunkerized-nginx
```

Please note that if you have one or more intermediate certificate(s) in your chain of trust, you will need to provide the bundle to `CUSTOM_HTTPS_CERT` (more info here).

You can reload the certificate(s) (i.e., in case of a renewal) by sending a reload order to bunkerized-nginx.

Docker reload :

```
docker kill --signal=SIGHUP my-container
```

Swarm and Kubernetes reload (repeat for each node) :

```
$ curl http://node-local-ip:80/api-uri/reload
```

Linux reload :

```
$ /usr/sbin/nginx -s reload
```

5.2.4 Self-signed certificate

This method is not recommended in production but can be used to quickly deploy HTTPS for testing purposes. Just use the `GENERATE_SELF_SIGNED_SSL=yes` environment variable and bunkerized-nginx will generate a self-signed certificate for you :

```
$ docker run -p 80:8080 \
             -p 443:8443 \
             -e GENERATE_SELF_SIGNED_SSL=yes \
             ...
             bunkerity/bunkerized-nginx
```

5.3 Headers

Some important HTTP headers related to client security are sent with a default value. Sometimes it can break a web application or can be tuned to provide even more security. The complete list is available here.

You can also remove headers (e.g., too verbose ones) by using the `REMOVE_HEADERS` environment variable which takes a list of header name separated with space (default value `= Server X-Powered-By X-AspNet-Version X-AspNetMvc-Version`).

If you want to keep your application headers and tell bunkerized-nginx to not override it, just set the corresponding environment variable to an empty value (e.g., `CONTENT_SECURITY_POLICY=`, `PERMISSIONS_POLICY=`, ...).

5.4 ModSecurity

ModSecurity is integrated and enabled by default alongside the OWASP Core Rule Set within bunkerized-nginx. To change this behaviour you can use the `USE_MODSECURITY=no` or `USE_MODSECURITY_CRS=no` environment variables.

We strongly recommend to keep both ModSecurity and the OWASP Core Rule Set enabled. The only downsides are the false positives that may occur. But they can be fixed easily and the CRS team maintains a list of exclusions for common application (e.g., wordpress, nextcloud, drupal, cpanel, ...).

Tuning the CRS with bunkerized-nginx is pretty simple : you can add configuration before and after the rules are loaded. You just need to mount your .conf files into the `/modsec-crs-confs` (before CRS is loaded) and `/modsec-confs` (after CRS is loaded) volumes. If you are using Linux integration the special folders are `/opt/bunkerized-nginx/modsec-confs` and `/opt/bunkerized-nginx/modsec-crs-confs`.

Here is a Docker example to illustrate it :

```
$ cat /data/exclusions-crs/wordpress.conf
SecAction \
 "id:900130,\
  phase:1,\
  nolog,\
  pass,\
  t:none,\
  setvar:tx.crs_exclusions_wordpress=1"

$ cat /data/tuning-crs/remove-false-positives.conf
SecRule REQUEST_FILENAME "/wp-admin/admin-ajax.php" "id:1,ctl:ruleRemoveByTag=attack-xss,
↪ctl:ruleRemoveByTag=attack-rce"
```

(continues on next page)

(continued from previous page)

```
SecRule REQUEST_FILENAME "/wp-admin/options.php" "id:2,ctl:ruleRemoveByTag=attack-xss"
SecRule REQUEST_FILENAME "^/wp-json/yoast" "id:3,ctl:ruleRemoveById=930120"

$ docker run -p 80:8080 \
             -p 443:8443 \
             -v /data/exclusions-crs:/modsec-crs-confs:ro \
             -v /data/tuning-crs:/modsec-confs:ro \
             ...
             bunkerity/bunkerized-nginx
```

5.5 Bad behaviors detection

When attackers search for and/or exploit vulnerabilities they might generate some suspicious HTTP status codes that a "regular" user won't generate within a period of time. If we detect that kind of behavior we can ban the offending IP address and force the attacker to come with a new one.

That kind of security measure is implemented and enabled by default in bunkerized-nginx. Here is the list of the related environment variables and their default value :

- `USE_BAD_BEHAVIOR=yes` : enable/disable "bad behavior" detection and automatic ban of IP
- `BAD_BEHAVIOR_STATUS_CODES=400 401 403 404 405 429 444` : the list of HTTP status codes considered as "suspicious"
- `BAD_BEHAVIOR_THRESHOLD=10` : the number of "suspicious" HTTP status codes required before we ban the corresponding IP address
- `BAD_BEHAVIOR_BAN_TIME=86400` : the duration time (in seconds) of the ban
- `BAD_BEHAVIOR_COUNT_TIME=60` : the duration time (in seconds) to wait before resetting the counter of "suspicious" HTTP status codes for a given IP

5.6 Antibot challenge

Attackers will certainly use automated tools to exploit/find some vulnerabilities on your web services. One countermeasure is to challenge the users to detect if they look like a bot. It might be effective against script kiddies or "lazy" attackers.

You can use the `USE_ANTIBOT` environment variable to add that kind of checks whenever a new client is connecting. The available challenges are : `cookie`, `javascript`, `captcha` and `recaptcha`. More info here.

5.7 External blacklists

5.7.1 Distributed

This feature is in beta and will be improved regularly.

You can benefit from a distributed blacklist shared among all of the bunkerized-nginx users.

Each time a bunkerized-nginx instance detect a bad request, the offending IP is sent to a remote API and will enrich a database. An extract of the top malicious IP is downloaded on a periodic basis and integrated into bunkerized-nginx as a blacklist.

This feature is controlled with the `USE_REMOTE_API=yes` environment variable.

To avoid poisoning, in addition to the various security checks made by the API we only mark IP as bad in the database if it has been seen by one of our honeypots under our control.

5.7.2 DNSBL

Automatic checks on external DNS BlackLists are enabled by default with the `USE_DNSBL=yes` environment variable. The list of DNSBL zones is also configurable, you just need to edit the `DNSBL_LIST` environment variable which contains the following value by default `bl.blocklist.de problems.dnsbl.sorbs.net sbl.spamhaus.org xbl.spamhaus.org`.

5.7.3 User-Agents

Sometimes script kiddies or lazy attackers don't put a "legitimate" value inside the **User-Agent** HTTP header so we can block them. This is controlled with the `BLOCK_USER_AGENT=yes` environment variable. The blacklist is composed of two files from here and here.

If a legitimate User-Agent is blacklisted, you can use the `WHITELIST_USER_AGENT` while still keeping the `BLOCK_USER_AGENT=yes` (more info here).

5.7.4 TOR exit nodes

Blocking TOR exit nodes might not be a good decision depending on your use case. We decided to enable it by default with the `BLOCK_TOR_EXIT_NODE=yes` environment variable. If privacy is a concern for you and/or your clients, you can override the default value (i.e : `BLOCK_TOR_EXIT_NODE=no`).

Please note that you have a concrete example on how to use bunkerized-nginx with a .onion hidden service here.

5.7.5 Proxies

This list contains IP addresses and networks known to be open proxies (downloaded from here). Unless privacy is important for you and/or your clients, you should keep the default environment variable `BLOCK_PROXIES=yes`.

5.7.6 Abusers

This list contains IP addresses and networks known to be abusing (downloaded from here). You can control this feature with the `BLOCK_ABUSERS` environment variable (default : `yes`).

5.7.7 Referrers

This list contains bad referrers domains known for spamming (downloaded from here). If one value is found inside the **Referer** HTTP header, request will be blocked. You can control this feature with the `BLOCK_REFERRER` environment variable (default = `yes`).

5.8 Limiting

5.8.1 Requests

To limit bruteforce attacks or rate limit access to your API you can use the "request limit" feature so attackers will be limited to X request(s) within a period of time for the same resource. That kind of protection might be useful against other attacks too (e.g., blind SQL injection).

Here is the list of related environment variables and their default value :

- `USE_LIMIT_REQ=yes` : enable/disable request limiting
- `LIMIT_REQ_URL=` : the URL you want to protect, use `/` to apply the limit for all URL
- `LIMIT_REQ_RATE=1r/s` : the rate to apply for the resource, valid period are : `s` (second), `m` (minute), `h` (hour) and `d` (day)
- `LIMIT_REQ_BURST=5 : the number of request tu put in a queue before effectively rejecting requests
- `LIMIT_REQ_DELAY=1` : the number of seconds to wait before we proceed requests in queue

Please note that you can apply different rate to different URL by appending a number as suffix (more info here).

5.8.2 Connections

Opening too many connections from the same IP address might be considered as suspicious (unless it's a shared IP and everyone is sending requests to your web service). It can be a dos/ddos attempt too. Bunkerized-nginx levarages the ngx_http_conn_module from nginx to prevent users opening too many connections.

Here is the list of related environment variables and their default value :

- `USE_LIMIT_CONN=yes` : enable disable connection limiting
- `LIMIT_CONN_MAX=50` : maximum number of connections per IP

5.9 Country

If the location of your clients is known, you may want to add another security layer by whitelisting or blacklisting some countries. You can use the `BLACKLIST_COUNTRY` or `WHITELIST_COUNTRY` environment variables depending on your approach. They both take a list of 2 letters country code separated with space.

5.10 Authentication

You can quickly protect sensitive resources (e.g. : admin panels) by requiring HTTP authentication. Here is the list of related environment variables and their default value :

- `USE_AUTH_BASIC=no` : enable/disable auth basic
- `AUTH_BASIC_LOCATION=sitewide` : location of the sensitive resource (e.g. `/admin`) or `sitewide` to force authentication on the whole service
- `AUTH_BASIC_USER=changeme` : the username required
- `AUTH_BASIC_PASSWORD=changeme` : the password required
- `AUTH_BASIC_TEXT=Restricted area` : the text that will be displayed to the user

Please note that bunkerized-nginx also supports Authelia for authentication (see the corresponding environment variables and a full example).

5.11 Whitelisting

Adding extra security can sometimes trigger false positives. Also, it might be not useful to do the security checks for specific clients because we decided to trust them. Bunkerized-nginx supports two types of whitelist : by IP address and by reverse DNS.

Here is the list of related environment variables and their default value :

- `USE_WHITELIST_IP=yes` : enable/disable whitelisting by IP address
- `WHITELIST_IP_LIST=23.21.227.69 40.88.21.235 50.16.241.113 50.16.241.114 50.16.241.117 50.16.247.234 52.204.97.54 52.5.190.19 54.197.234.188 54.208.100.253 54.208.102.37 107.21.1.8` : list of IP addresses and/or network CIDR blocks to whitelist (default contains the IP addresses of the DuckDuckGo crawler)
- `USE_WHITELIST_REVERSE=yes` : enable/disable whitelisting by reverse DNS
- `WHITELIST_REVERSE_LIST=.googlebot.com .google.com .search.msn.com .crawl.yahoot.net .crawl.baidu.jp .crawl.baidu.com .yandex.com .yandex.ru .yandex.net` : the list of reverse DNS suffixes to trust (default contains the list of major search engines crawlers)

5.12 Blacklisting

Sometimes it isn't necessary to spend some resources for a particular client because we know for sure that he is malicious. Bunkerized-nginx nginx supports two types of blacklisting : by IP address and by reverse DNS.

Here is the list of related environment variables and their default value :

- `USE_BLACKLIST_IP=yes` : enable/disable blacklisting by IP address
- `BLACKLIST_IP_LIST=` : list of IP addresses and/or network CIDR blocks to blacklist
- `USE_BLACKLIST_REVERSE=yes` : enable/disable blacklisting by reverse DNS
- `BLACKLIST_REVERSE_LIST=.shodan.io` : the list of reverse DNS suffixes to never trust

5.13 Plugins

Some security features can be added through the plugins system (e.g., ClamAV, CrowdSec, ...). You will find more info in the plugins section.

5.14 Container hardening

You will find a ready to use docker-compose.yml file focused on container hardening here.

5.14.1 Drop capabilities

By default, bunkerized-nginx runs as non-root user inside the container and should not use any of the default capabilities allowed by Docker. You can safely remove all capabilities to harden the container :

```
docker run ... --drop-cap=all ... bunkerity/bunkerized-nginx
```

5.14.2 No new privileges

Bunkerized-nginx should never tries to gain additional privileges through setuid/setgid executables. You can safely add the **no-new-privileges** security configuration when creating the container :

```
docker run ... --security-opt no-new-privileges ... bunkerity/bunkerized-nginx
```

5.14.3 Read-only

Since the locations where bunkerized-nginx needs to write are known, we can run the container with a read-only root file system and only allow writes to specific locations by adding volumes and a tmpfs mount :

```
docker run ... --read-only --tmpfs /tmp -v cache-vol:/cache -v conf-vol:/etc/nginx -v /
↪path/to/web/files:/www:ro -v /where/to/store/certificates:/etc/letsencrypt bunkerity/
↪bunkerized-nginx
```

5.14.4 User namespace remap

Another hardening trick is user namespace remapping : it allows you to map the UID/GID of users inside a container to another UID/GID on the host. For example, you can map the user nginx with UID/GID 101 inside the container to a non-existent user with UID/GID 100101 on the host.

Let's assume you have the /etc/subuid and /etc/subgid files like this :

```
user:100000:65536
```

It means that everything done inside the container will be remapped to UID/GID 100101 (100000 + 101) on the host.

Please note that you must set the rights on the volumes (e.g. : /etc/letsencrypt, /www, ...) according to the remapped UID/GID :

```
$ chown root:100101 /path/to/letsencrypt
$ chmod 770 /path/to/letsencrypt
$ docker run ... -v /path/to/letsencrypt:/etc/letsencrypt ... bunkerity/bunkerized-nginx
```

CHAPTER

SIX

WEB UI

6.1 Overview

6.2 Usage

The web UI has its own set of environment variables to configure it :

- `ADMIN_USERNAME` and `ADMIN_PASSWORD` : credentials for accessing the web UI
- `ABSOLUTE_URI` : the full public URI that points to the web UI
- `API_URI` : path of the bunkerized-nginx API (must match the corresponding `API_URI` of the bunkerized-nginx instance)
- `DOCKER_HOST` : Docker API endpoint address (default = `unix:///var/run/docker.sock`)

Since the web UI is a web service itself, we can use bunkerized-nginx as a reverse proxy in front of it.

Using the web UI in a Docker environment exposes a security risk because you need to mount the Docker API socket into the web UI container. It's highly recommended to use a middleware like tecnativa/docker-socket-proxy to reduce the risk as much as possible.

You need to apply the security best practices because the web UI contains code and that code might be vulnerable : complex admin password, hard to guess public URI, network isolation from others services, HTTPS only, ...

6.2.1 Docker

First of all, we will need to setup two networks one for ui communication and the other one for the services :

```
$ docker network create ui-net
$ docker network create services-net
```

We also need a volume to shared the generated configuration from the web UI to the bunkerized-nginx instances :

```
$ docker volume create bunkerized-vol
```

Next we will create the "Docker API proxy" container that will be in the front of the Docker socket and deny access to sensitive things :

```
$ docker run -d \
         --name my-docker-proxy \
         --network ui-net \
         -v /var/run/docker.sock:/var/run/docker.sock:ro \
```

(continues on next page)

(continued from previous page)

```
        -e CONTAINERS=1 \
        -e SWARM=1 \
        -e SERVICES=1 \
        tecnativa/docker-socket-proxy
```

We can now create the web UI container based on bunkerized-nginx-ui image :

```
$ docker run -d \
        --name my-bunkerized-ui \
        --network ui-net \
        -v bunkerized-vol:/etc/nginx \
        -e ABSOLUTE_URI=https://admin.example.com/admin-changeme/ \
        -e DOCKER_HOST=tcp://my-docker-proxy:2375 \
        -e API_URI=/ChangeMeToSomethingHardToGuess \
        -e ADMIN_USERNAME=admin \
        -e ADMIN_PASSWORD=changeme \
        bunkerity/bunkerized-nginx-ui
```

Last but not least, you need to start the bunkerized-nginx and configure it as a reverse proxy for the web UI web service :

```
$ docker create \
        --name my-bunkerized \
        --network ui-net \
        -p 80:8080 \
        -p 443:8443 \
        -v bunkerized-vol:/etc/nginx \
        -v "${PWD}/certs:/etc/letsencrypt" \
        -e SERVER_NAME=admin.example.com \
        -e MULTISITE=yes \
        -e USE_API=yes \
        -e API_URI=/ChangeMeToSomethingHardToGuess \
        -e AUTO_LETS_ENCRYPT=yes \
        -e REDIRECT_HTTP_TO_HTTPS=yes \
        -e admin.example.com_USE_REVERSE_PROXY=yes \
        -e admin.example.com_REVERSE_PROXY_URL=/admin-changeme/ \
        -e admin.example.com_REVERSE_PROXY_HOST=http://my-bunkerized-ui:5000 \
        -e "admin.example.com_REVERSE_PROXY_HEADERS=X-Script-Name /admin-changeme" \
        -e admin.example.com_USE_MODSECURITY=no \
        -l bunkerized-nginx.UI \
        bunkerity/bunkerized-nginx
$ docker network connect services-net my-bunkerized
$ docker start my-bunkerized
```

The web UI should now be accessible at https://admin.example.com/admin-changeme/.

docker-compose equivalent :

```
version: '3'

services:

  my-bunkerized:
```

(continues on next page)

(continued from previous page)

```
    image: bunkerity/bunkerized-nginx
    restart: always
    depends_on:
      - my-bunkerized-ui
    networks:
      - services-net
      - ui-net
    ports:
      - 80:8080
      - 443:8443
    volumes:
      - ./letsencrypt:/etc/letsencrypt
      - bunkerized-vol:/etc/nginx
    environment:
      - SERVER_NAME=admin.example.com                                    # replace␣
↪with your domain
      - MULTISITE=yes
      - USE_API=yes
      - API_URI=/ChangeMeToSomethingHardToGuess                          # change␣
↪it to something hard to guess + must match API_URI from myui service
      - AUTO_LETS_ENCRYPT=yes
      - REDIRECT_HTTP_TO_HTTPS=yes
      - admin.example.com_USE_REVERSE_PROXY=yes
      - admin.example.com_REVERSE_PROXY_URL=/admin-changeme/             # change␣
↪it to something hard to guess
      - admin.example.com_REVERSE_PROXY_HOST=http://my-bunkerized-ui:5000
      - admin.example.com_REVERSE_PROXY_HEADERS=X-Script-Name /admin-changeme # must␣
↪match REVERSE_PROXY_URL
      - admin.example.com_USE_MODSECURITY=no
    labels:
      - "bunkerized-nginx.UI"

  my-bunkerized-ui:
    image: bunkerity/bunkerized-nginx-ui
    restart: always
    depends_on:
      - my-docker-proxy
    networks:
      - ui-net
    volumes:
      - bunkerized-vol:/etc/nginx
    environment:
      - ABSOLUTE_URI=https://admin.example.com/admin-changeme/ # change it to your full␣
↪URI
      - DOCKER_HOST=tcp://my-docker-proxy:2375
      - API_URI=/ChangeMeToSomethingHardToGuess                # must match API_URI from␣
↪bunkerized-nginx
      - ADMIN_USERNAME=admin                                   # change it to something␣
↪hard to guess
      - ADMIN_PASSWORD=changeme                                # change it to a good␣
↪password
```

(continues on next page)

(continued from previous page)

```
  my-docker-proxy:
    image: tecnativa/docker-socket-proxy
    restart: always
    networks:
      - ui-net
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    environment:
      - CONTAINERS=1
      - SWARM=1
      - SERVICES=1

networks:
  ui-net:
  services-net:
    name: services-net

volumes:
  bunkerized-vol:
```

6.2.2 Linux

First of all, you need to edit the web UI configuration file located at `/opt/bunkerized-nginx/ui/variables.env` :

```
ABSOLUTE_URI=https://admin.example.com/admin-changeme/
DOCKER_HOST=
ADMIN_USERNAME=admin
ADMIN_PASSWORD=changeme
```

Make sure that the web UI service is automatically started on boot :

```
$ systemctl enable bunkerized-nginx-ui
```

Now you can start the web UI service :

```
$ systemctl start bunkerized-nginx-ui
```

Edit the bunkerized-nginx configurations located at `/opt/bunkerized-nginx/variables.env` :

```
HTTP_PORT=80
HTTPS_PORT=443
DNS_RESOLVERS=8.8.8.8 8.8.4.4
SERVER_NAME=admin.example.com
MULTISITE=yes
AUTO_LETS_ENCRYPT=yes
REDIRECT_HTTP_TO_HTTPS=yes
admin.example.com_USE_REVERSE_PROXY=yes
admin.example.com_REVERSE_PROXY_URL=/admin-changeme/
# Local bunkerized-nginx-ui
admin.example.com_REVERSE_PROXY_HOST=http://127.0.0.1:5000
```

(continues on next page)

(continued from previous page)

```
# Remote bunkerized-nginx-ui
#REVERSE_PROXY_HOST=http://service.example.local:5000
admin.example.com_REVERSE_PROXY_HEADERS=X-Script-Name /admin-changeme
admin.example.com_USE_MODSECURITY=no
```

And run the `bunkerized-nginx` command to apply changes :

```
$ bunkerized-nginx
```

The web UI should now be accessible at https://admin.example.com/admin-changeme/.

CHAPTER

SEVEN

LIST OF ENVIRONMENT VARIABLES

7.1 nginx

7.1.1 Misc

`MULTISITE`
Values : *yes* | *no*
Default value : *no*
Context : *global*
When set to *no*, only one server block will be generated. Otherwise one server per host defined in the `SERVER_NAME` environment variable will be generated.
Any environment variable tagged as *multisite* context can be used for a specific server block with the following format : *host_VARIABLE=value*. If the variable is used without the host prefix it will be applied to all the server blocks (but still can be overriden).

`SERVER_NAME`
Values : <*first name*> <*second name*> ...
Default value : *www.bunkerity.com*
Context : *global*, *multisite*
Sets the host names of the webserver separated with spaces. This must match the Host header sent by clients.
Useful when used with `MULTISITE=yes` and/or `AUTO_LETSENCRYPT=yes` and/or `DISABLE_DEFAULT_SERVER=yes`.

`MAX_CLIENT_SIZE`
Values : *0* | *Xm*
Default value : *10m*
Context : *global*, *multisite*
Sets the maximum body size before nginx returns a 413 error code.
Setting to 0 means "infinite" body size.

`ALLOWED_METHODS`
Values : *allowed HTTP methods separated with | char*
Default value : *GET|POST|HEAD*
Context : *global*, *multisite*
Only the HTTP methods listed here will be accepted by nginx. If not listed, nginx will close the connection.

`DISABLE_DEFAULT_SERVER`
Values : *yes* | *no*
Default value : *no*
Context : *global*
If set to yes, nginx will only respond to HTTP request when the Host header match a FQDN specified in the `SERVER_NAME` environment variable.
For example, it will close the connection if a bot access the site with direct ip.

SERVE_FILES
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, nginx will serve files from /www directory within the container.
A use case to not serving files is when you setup bunkerized-nginx as a reverse proxy.

DNS_RESOLVERS
Values : *<two IP addresses separated with a space>*
Default value : *127.0.0.11*
Context : *global*
The IP addresses of the DNS resolvers to use when performing DNS lookups.

ROOT_FOLDER
Values : *<any valid path to web files>*
Default value : */www*
Context : *global*
The default folder where nginx will search for web files. Don't change it unless you know what you are doing.

ROOT_SITE_SUBFOLDER
Values : *<any valid directory name>*
Default value :
Context : *global*, *multisite*
The subfolder where nginx will search for site web files.

LOG_FORMAT
Values : *<any values accepted by the log_format directive>*
Default value : *$host $remote_addr - $remote_user [$time_local] "$request" $status $body_bytes_sent "$http_referer" "$http_user_agent"*
Context : *global*
The log format used by nginx to generate logs. More info here.

LOG_LEVEL
Values : *debug, info, notice, warn, error, crit, alert, or emerg*
Default value : *info*
Context : *global*
The level of logging : *debug* means more logs and *emerg* means less logs. More info here.

HTTP_PORT
Values : *<any valid port greater than 1024>*
Default value : *8080*
Context : *global*
The HTTP port number used by nginx inside the container.

HTTPS_PORT
Values : *<any valid port greater than 1024>*
Default value : *8443*
Context : *global*
The HTTPS port number used by nginx inside the container.

WORKER_CONNECTIONS
Values : *<any positive integer>*
Default value : 1024
Context : *global*
Sets the value of the worker_connections directive.

WORKER_RLIMIT_NOFILE
Values : *<any positive integer>*

Default value : 2048
Context : *global*
Sets the value of the worker_rlimit_nofile directive.

`WORKER_PROCESSES`
Values : *<any positive integer or auto>*
Default value : auto
Context : *global*
Sets the value of the worker_processes directive.

`INJECT_BODY`
Values : *<any HTML code>*
Default value :
Context : *global*, *multisite*
Use this variable to inject any HTML code you want before the </body> tag (e.g. : `\<script src="https://... "\>`)

`REDIRECT_TO`
Values : *<any valid absolute URI>*
Default value :
Context : *global*, *multisite*
Use this variable if you want to redirect one server to another (e.g., redirect apex to www : `REDIRECT_TO=https://www.example.com`).

`REDIRECT_TO_REQUEST_URI`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to yes and `REDIRECT_TO` is set it will append the requested path to the redirection (e.g., https://example.com/something redirects to https://www.example.com/something).

`CUSTOM_HEADER`
Values : *<HeaderName: HeaderValue>*
Default value :
Context : *global*, *multisite*
Add custom HTTP header of your choice to clients. You can add multiple headers by appending a number as a suffix of the environment variable : `CUSTOM_HEADER_1`, `CUSTOM_HEADER_2`, `CUSTOM_HEADER_3`, ...

7.1.2 Information leak

`SERVER_TOKENS`
Values : *on* | *off*
Default value : *off*
Context : *global*
If set to on, nginx will display server version in Server header and default error pages.

`REMOVE_HEADERS`
Values : *<list of headers separated with space>*
Default value : *Server X-Powered-By X-AspNet-Version X-AspNetMvc-Version*
Context : *global*, *multisite*
List of header to remove when sending responses to clients.

7.1.3 Custom error pages

ERRORS
Values : *<error1=/page1 error2=/page2>*
Default value :
Context : *global*, *multisite*
Use this kind of environment variable to define custom error page depending on the HTTP error code. Replace errorX with HTTP code.
Example : `ERRORS=404=/404.html 403=/403.html` the /404.html page will be displayed when 404 code is generated (same for 403 and /403.html page). The path is relative to the root web folder.

7.1.4 HTTP basic authentication

USE_AUTH_BASIC
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to yes, enables HTTP basic authentication at the location AUTH_BASIC_LOCATION with user AUTH_BASIC_USER and password AUTH_BASIC_PASSWORD.

AUTH_BASIC_LOCATION
Values : *sitewide* | */somedir* | *<any valid location>*
Default value : *sitewide*
Context : *global*, *multisite*
The location to restrict when USE_AUTH_BASIC is set to *yes*. If the special value *sitewide* is used then auth basic will be set at server level outside any location context.

AUTH_BASIC_USER
Values : *<any valid username>*
Default value : *changeme*
Context : *global*, *multisite*
The username allowed to access AUTH_BASIC_LOCATION when USE_AUTH_BASIC is set to yes.

AUTH_BASIC_PASSWORD
Values : *<any valid password>*
Default value : *changeme*
Context : *global*, *multisite*
The password of AUTH_BASIC_USER when USE_AUTH_BASIC is set to yes.

AUTH_BASIC_TEXT
Values : *<any valid text>*
Default value : *Restricted area*
Context : *global*, *multisite*
The text displayed inside the login prompt when USE_AUTH_BASIC is set to yes.

7.1.5 Reverse proxy

USE_REVERSE_PROXY
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
Set this environment variable to *yes* if you want to use bunkerized-nginx as a reverse proxy.

REVERSE_PROXY_URL
Values : <*any valid location path*>
Default value :
Context : *global*, *multisite*
Only valid when USE_REVERSE_PROXY is set to *yes*. Let's you define the location path to match when acting as a reverse proxy.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_URL_1, REVERSE_PROXY_URL_2, REVERSE_PROXY_URL_3, ...

REVERSE_PROXY_HOST
Values : <*any valid proxy_pass value*>
Default value :
Context : *global*, *multisite*
Only valid when USE_REVERSE_PROXY is set to *yes*. Let's you define the proxy_pass destination to use when acting as a reverse proxy.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_HOST_1, REVERSE_PROXY_HOST_2, REVERSE_PROXY_HOST_3, ...

REVERSE_PROXY_WS
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
Only valid when USE_REVERSE_PROXY is set to *yes*. Set it to *yes* when the corresponding REVERSE_PROXY_HOST is a WebSocket server.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_WS_1, REVERSE_PROXY_WS_2, REVERSE_PROXY_WS_3, ...

REVERSE_PROXY_BUFFERING
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Only valid when USE_REVERSE_PROXY is set to *yes*. Set it to *yes* then the proxy_buffering directive will be set to on or off otherwise.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_BUFFERING_1, REVERSE_PROXY_BUFFERING_2, REVERSE_PROXY_BUFFERING_3, ...

REVERSE_PROXY_KEEPALIVE
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Only valid when USE_REVERSE_PROXY is set to *yes*. Set it to *yes* to enable keepalive connections with the backend (needs a HTTP 1.1 backend) or *no* otherwise.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_KEEPALIVE_1, REVERSE_PROXY_KEEPALIVE_2, REVERSE_PROXY_KEEPALIVE_3, ...

REVERSE_PROXY_HEADERS
Values : <*list of custom headers separated with a semicolon like this : header1 value1;header2 value2...*> Default value :
Context : *global*, *multisite*

Only valid when USE_REVERSE_PROXY is set to *yes*.
You can set multiple url/host by adding a suffix number to the variable name like this : REVERSE_PROXY_HEADERS_1, REVERSE_PROXY_HEADERS_2, REVERSE_PROXY_HEADERS_3, ...

PROXY_REAL_IP
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
Set this environment variable to *yes* if you're using bunkerized-nginx behind a reverse proxy. This means you will see the real client address instead of the proxy one inside your logs. Security tools will also then work correctly.

PROXY_REAL_IP_FROM
Values : <*list of trusted IP addresses and/or networks separated with spaces*>
Default value : *192.168.0.0/16 172.16.0.0/12 10.0.0.0/8*
Context : *global*, *multisite*
When PROXY_REAL_IP is set to *yes*, lets you define the trusted IPs/networks allowed to send the correct client address.

PROXY_REAL_IP_HEADER
Values : *X-Forwarded-For* | *X-Real-IP* | *custom header*
Default value : *X-Forwarded-For*
Context : *global*, *multisite*
When PROXY_REAL_IP is set to *yes*, lets you define the header that contains the real client IP address.

PROXY_REAL_IP_RECURSIVE
Values : *on* | *off*
Default value : *on*
Context : *global*, *multisite*
When PROXY_REAL_IP is set to *yes*, setting this to *on* avoid spoofing attacks using the header defined in PROXY_REAL_IP_HEADER.

7.1.6 Compression

USE_GZIP
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to *yes*, nginx will use the gzip algorithm to compress responses sent to clients.

GZIP_COMP_LEVEL
Values : <*any integer between 1 and 9*>
Default value : *5*
Context : *global*, *multisite*
The gzip compression level to use when USE_GZIP is set to *yes*.

GZIP_MIN_LENGTH
Values : <*any positive integer*>
Default value : *1000*
Context : *global*, *multisite*
The minimum size (in bytes) of a response required to compress when USE_GZIP is set to *yes*.

GZIP_TYPES
Values : <*list of mime types separated with space*>
Default value : *application/atom+xml application/javascript application/json application/rss+xml application/vnd.ms-fontobject application/x-font-opentype application/x-font-truetype application/x-font-ttf application/x-javascript application/xhtml+xml application/xml font/eot font/opentype font/otf font/truetype image/svg+xml image/vnd.microsoft.icon image/x-icon image/x-win-bitmap text/css text/javascript text/plain text/xml*

Context : *global*, *multisite*
List of response MIME type required to compress when `USE_GZIP` is set to *yes*.

`USE_BROTLI`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to *yes*, nginx will use the brotli algorithm to compress responses sent to clients.

`BROTLI_COMP_LEVEL`
Values : <*any integer between 1 and 9*>
Default value : *5*
Context : *global*, *multisite*
The brotli compression level to use when `USE_BROTLI` is set to *yes*.

`BROTLI_MIN_LENGTH`
Values : <*any positive integer*>
Default value : *1000*
Context : *global*, *multisite*
The minimum size (in bytes) of a response required to compress when `USE_BROTLI` is set to *yes*.

`BROTLI_TYPES`
Values : <*list of mime types separated with space*>
Default value : *application/atom+xml application/javascript application/json application/rss+xml application/vnd.ms-fontobject application/x-font-opentype application/x-font-truetype application/x-font-ttf application/x-javascript application/xhtml+xml application/xml font/eot font/opentype font/otf font/truetype image/svg+xml image/vnd.microsoft.icon image/x-icon image/x-win-bitmap text/css text/javascript text/plain text/xml*
Context : *global*, *multisite*
List of response MIME type required to compress when `USE_BROTLI` is set to *yes*.

7.1.7 Cache

`USE_CLIENT_CACHE`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to *yes*, clients will be told to cache some files locally.

`CLIENT_CACHE_EXTENSIONS`
Values : <*list of extensions separated with* |>
Default value : *jpg|jpeg|png|bmp|ico|svg|tif|css|js|otf|ttf|eot|woff|woff2*
Context : *global*, *multisite*
List of file extensions that clients should cache when `USE_CLIENT_CACHE` is set to *yes*.

`CLIENT_CACHE_CONTROL`
Values : <*Cache-Control header value*>
Default value : *public, max-age=15552000*
Context : *global*, *multisite*
Content of the Cache-Control header to send when `USE_CLIENT_CACHE` is set to *yes*.

`CLIENT_CACHE_ETAG`
Values : *on* | *off*
Default value : *on*
Context : *global*, *multisite*
Whether or not nginx will send the ETag header when `USE_CLIENT_CACHE` is set to *yes*.

USE_OPEN_FILE_CACHE
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to *yes*, nginx will cache open fd, existence of directories, ... See open_file_cache.

OPEN_FILE_CACHE
Values : <*any valid open_file_cache parameters*>
Default value : *max=1000 inactive=20s*
Context : *global*, *multisite*
Parameters to use with open_file_cache when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_ERRORS
Values : *on* | *off*
Default value : *on*
Context : *global*, *multisite*
Whether or not nginx should cache file lookup errors when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_MIN_USES
Values : <*any valid integer *>
Default value : *2*
Context : *global*, *multisite*
The minimum number of file accesses required to cache the fd when USE_OPEN_FILE_CACHE is set to *yes*.

OPEN_FILE_CACHE_VALID
Values : <*any time value like Xs, Xm, Xh, ...*>
Default value : *30s*
Context : *global*, *multisite*
The time after which cached elements should be validated when USE_OPEN_FILE_CACHE is set to *yes*.

USE_PROXY_CACHE
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to *yes*, nginx will cache responses from proxied applications. See proxy_cache.

PROXY_CACHE_PATH_ZONE_SIZE
Values : <*any valid size like Xk, Xm, Xg, ...*>
Default value : *10m*
Context : *global*, *multisite*
Maximum size of cached metadata when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_PATH_PARAMS
Values : <*any valid parameters to proxy_cache_path directive*>
Default value : *max_size=100m*
Context : *global*, *multisite*
Parameters to use for proxy_cache_path directive when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_METHODS
Values : <*list of HTTP methods separated with space*>
Default value : *GET HEAD*
Context : *global*, *multisite*
The HTTP methods that should trigger a cache operation when USE_PROXY_CACHE is set to *yes*.

PROXY_CACHE_MIN_USES
Values : <*any positive integer*>
Default value : *2*
Context : *global*, *multisite*

The minimum number of requests before the response is cached when `USE_PROXY_CACHE` is set to *yes*.

`PROXY_CACHE_KEY`
Values : <*list of variables*>
Default value : *$scheme$host$request_uri*
Context : *global*, *multisite*
The key used to uniquely identify a cached response when `USE_PROXY_CACHE` is set to *yes*.

`PROXY_CACHE_VALID`
Values : <*status=time list separated with space*>
Default value : *200=10m 301=10m 302=1h*
Context : *global*, *multisite*
Define the caching time depending on the HTTP status code (list of status=time separated with space) when `USE_PROXY_CACHE` is set to *yes*.

`PROXY_NO_CACHE`
Values : <*list of variables*>
Default value : *$http_authorization*
Context : *global*, *multisite*
Conditions that must be met to disable caching of the response when `USE_PROXY_CACHE` is set to *yes*.

`PROXY_CACHE_BYPASS`
Values : <*list of variables*> Default value : *$http_authorization*
Context : *global*, *multisite* Conditions that must be met to bypass the cache when `USE_PROXY_CACHE` is set to *yes*.

7.2 HTTPS

7.2.1 Let's Encrypt

`AUTO_LETS_ENCRYPT`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to yes, automatic certificate generation and renewal will be setup through Let's Encrypt. This will enable HTTPS on your website for free.
You will need to redirect the 80 port to 8080 port inside container and also set the `SERVER_NAME` environment variable.

`EMAIL_LETS_ENCRYPT`
Values : *contact@yourdomain.com*
Default value : *contact@first-domain-in-server-name*
Context : *global*, *multisite*
Define the contact email address declare in the certificate.

`USE_LETS_ENCRYPT_STAGING`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
When set to yes, it tells certbot to use the staging environment for Let's Encrypt certificate generation. Useful when you are testing your deployments to avoid being rate limited in the production environment.

7.2.2 HTTP

`LISTEN_HTTP`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to no, nginx will not in listen on HTTP (port 80).
Useful if you only want HTTPS access to your website.

`REDIRECT_HTTP_TO_HTTPS`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to yes, nginx will redirect all HTTP requests to HTTPS.

7.2.3 Custom certificate

`USE_CUSTOM_HTTPS`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to yes, HTTPS will be enabled with certificate/key of your choice.

`CUSTOM_HTTPS_CERT`
Values : <*any valid path inside the container*>
Default value :
Context : *global*, *multisite*
Full path of the certificate or bundle file to use when `USE_CUSTOM_HTTPS` is set to yes. If your chain of trust contains one or more intermediate certificate(s), you will need to bundle them into a single file (more info here).

`CUSTOM_HTTPS_KEY`
Values : <*any valid path inside the container*>
Default value :
Context : *global*, *multisite*
Full path of the key file to use when `USE_CUSTOM_HTTPS` is set to yes.

7.2.4 Self-signed certificate

`GENERATE_SELF_SIGNED_SSL`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to yes, HTTPS will be enabled with a container generated self-signed certificate.

`SELF_SIGNED_SSL_EXPIRY`
Values : *integer*
Default value : *365* (1 year)
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the expiry date for the self generated certificate.

`SELF_SIGNED_SSL_COUNTRY`
Values : *text*
Default value : *Switzerland*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the country for the self generated certificate.

`SELF_SIGNED_SSL_STATE`
Values : *text*, *multisite*
Default value : *Switzerland*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the state for the self generated certificate.

`SELF_SIGNED_SSL_CITY`
Values : *text*
Default value : *Bern*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the city for the self generated certificate.

`SELF_SIGNED_SSL_ORG`
Values : *text*
Default value : *AcmeInc*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the organisation name for the self generated certificate.

`SELF_SIGNED_SSL_OU`
Values : *text*
Default value : *IT*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the organisitional unit for the self generated certificate.

`SELF_SIGNED_SSL_CN`
Values : *text*
Default value : *bunkerity-nginx*
Context : *global*, *multisite*
Needs `GENERATE_SELF_SIGNED_SSL` to work. Sets the CN server name for the self generated certificate.

7.2.5 Misc

`HTTP2`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, nginx will use HTTP2 protocol when HTTPS is enabled.

`HTTPS_PROTOCOLS`
Values : *TLSv1.2* | *TLSv1.3* | *TLSv1.2 TLSv1.3*
Default value : *TLSv1.2 TLSv1.3*
Context : *global*, *multisite*
The supported version of TLS. We recommend the default value *TLSv1.2 TLSv1.3* for compatibility reasons.

7.3 ModSecurity

`USE_MODSECURITY`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, the ModSecurity WAF will be enabled.
You can include custom rules by adding .conf files into the /modsec-confs/ directory inside the container (i.e : through a volume).

`USE_MODSECURITY_CRS`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, the OWASP ModSecurity Core Rule Set will be used. It provides generic rules to detect common web attacks.
You can customize the CRS (i.e. : add WordPress exclusions) by adding custom .conf files into the /modsec-crs-confs/ directory inside the container (i.e : through a volume). Files inside this directory are included before the CRS rules. If you need to tweak (i.e. : SecRuleUpdateTargetById) put .conf files inside the /modsec-confs/ which is included after the CRS rules.

`MODSECURITY_SEC_AUDIT_ENGINE`
Values : *On* | *Off* | *RelevantOnly*
Default value : *RelevantOnly*
Context : *global*, *multisite*
Sets the value of the SecAuditEngine directive of ModSecurity.

7.4 Security headers

If you want to keep your application headers and tell bunkerized-nginx to not override it, just set the corresponding environment variable to an empty value (e.g., `CONTENT_SECURITY_POLICY=`, `PERMISSIONS_POLICY=`, ...).

`X_FRAME_OPTIONS`
Values : *DENY* | *SAMEORIGIN* | *ALLOW-FROM https://www.website.net* Default value : *DENY*
Context : *global*, *multisite*
Policy to be used when the site is displayed through iframe. Can be used to mitigate clickjacking attacks. More info here.

`X_XSS_PROTECTION`
Values : *0* | *1* | *1; mode=block*
Default value : *1; mode=block*
Context : *global*, *multisite*
Policy to be used when XSS is detected by the browser. Only works with Internet Explorer.
More info here.

`X_CONTENT_TYPE_OPTIONS`
Values : *nosniff*
Default value : *nosniff*
Context : *global*, *multisite*
Tells the browser to be strict about MIME type.
More info here.

`REFERRER_POLICY`
Values : *no-referrer* | *no-referrer-when-downgrade* | *origin* | *origin-when-cross-origin* | *same-origin* | *strict-origin* | *strict-origin-when-cross-origin* | *unsafe-url*
Default value : *no-referrer*
Context : *global*, *multisite*
Policy to be used for the Referer header.
More info here.

`FEATURE_POLICY`
Values : *<directive> <allow list>*
Default value : *accelerometer 'none'; ambient-light-sensor 'none'; autoplay 'none'; battery 'none'; camera 'none'; display-capture 'none'; document-domain 'none'; encrypted-media 'none'; fullscreen 'none'; geolocation 'none'; gyroscope 'none'; magnetometer 'none'; microphone 'none'; midi 'none'; payment 'none'; picture-in-picture 'none';*

publickey-credentials-get 'none'; sync-xhr 'none'; usb 'none'; wake-lock 'none'; web-share 'none'; xr-spatial-tracking 'none"
Context : *global*, *multisite*
Tells the browser which features can be used on the website.
More info here.

`PERMISSIONS_POLICY`
Values : *feature=(allow list)*
Default value : *accelerometer=(), ambient-light-sensor=(), autoplay=(), battery=(), camera=(), display-capture=(), document-domain=(), encrypted-media=(), fullscreen=(), geolocation=(), gyroscope=(), interest-cohort=(), magnetometer=(), microphone=(), midi=(), payment=(), picture-in-picture=(), publickey-credentials-get=(), screen-wake-lock=(), sync-xhr=(), usb=(), web-share=(), xr-spatial-tracking=()*
Context : *global*, *multisite*
Tells the browser which features can be used on the website.
More info here.

`COOKIE_FLAGS`
Values : ** HttpOnly | MyCookie secure SameSite=Lax | ...*
Default value : ** HttpOnly SameSite=Lax*
Context : *global*, *multisite*
Adds some security to the cookies set by the server.
Accepted value can be found here.

`COOKIE_AUTO_SECURE_FLAG`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
When set to *yes*, the *secure* will be automatically added to cookies when using HTTPS.

`STRICT_TRANSPORT_SECURITY`
Values : *max-age=expireTime [; includeSubDomains] [; preload]*
Default value : *max-age=31536000*
Context : *global*, *multisite*
Tells the browser to use exclusively HTTPS instead of HTTP when communicating with the server.
More info here.

`CONTENT_SECURITY_POLICY`
Values : *<directive 1>; <directive 2>; ...*
Default value : *object-src 'none'; frame-ancestors 'self'; form-action 'self'; block-all-mixed-content; sandbox allow-forms allow-same-origin allow-scripts allow-popups allow-downloads; base-uri 'self';*
Context : *global*, *multisite*
Policy to be used when loading resources (scripts, forms, frames, ...).
More info here.

7.5 Blocking

7.5.1 Antibot

`USE_ANTIBOT`
Values : *no* | *cookie* | *javascript* | *captcha* | *recaptcha*
Default value : *no*
Context : *global*, *multisite*
If set to another allowed value than *no*, users must complete a "challenge" before accessing the pages on your website :

- *cookie* : asks the users to set a cookie

- *javascript* : users must execute a javascript code
- *captcha* : a text captcha must be resolved by the users
- *recaptcha* : use Google reCAPTCHA v3 score to allow/deny users

`ANTIBOT_URI`
Values : *<any valid uri>*
Default value : */challenge*
Context : *global*, *multisite*
A valid and unused URI to redirect users when `USE_ANTIBOT` is used. Be sure that it doesn't exist on your website.

`ANTIBOT_SESSION_SECRET`
Values : *random* | *<32 chars of your choice>*
Default value : *random*
Context : *global*, *multisite*
A secret used to generate sessions when `USE_ANTIBOT` is set. Using the special *random* value will generate a random one. Be sure to use the same value when you are in a multi-server environment (so sessions are valid in all the servers).

`ANTIBOT_RECAPTCHA_SCORE`
Values : *<0.0 to 1.0>*
Default value : *0.7*
Context : *global*, *multisite*
The minimum score required when `USE_ANTIBOT` is set to *recaptcha*.

`ANTIBOT_RECAPTCHA_SITEKEY`
Values : *<public key given by Google>*
Default value :
Context : *global*, *multisite*
The sitekey given by Google when `USE_ANTIBOT` is set to *recaptcha*.

`ANTIBOT_RECAPTCHA_SECRET`
Values : *<private key given by Google>*
Default value :
Context : *global*, *multisite*
The secret given by Google when `USE_ANTIBOT` is set to *recaptcha*.

7.5.2 Distributed blacklist

`USE_REMOTE_API`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, the instance will participate into the distributed blacklist shared among all other instances. The blacklist will be automaticaly downloaded on a periodic basis.

`REMOTE_API_SERVER`
Values : *<any valid full URL>*
Default value :
Context : *global*
Full URL of the remote API used for the distributed blacklist.

7.5.3 External blacklists

`BLOCK_USER_AGENT`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, block clients with "bad" user agent.
Blacklist can be found here and here.

`BLOCK_TOR_EXIT_NODE`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Is set to yes, will block known TOR exit nodes.
Blacklist can be found here.

`BLOCK_PROXIES`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Is set to yes, will block known proxies.
Blacklist can be found here.

`BLOCK_ABUSERS`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Is set to yes, will block known abusers.
Blacklist can be found here.

`BLOCK_REFERRER`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
Is set to yes, will block known bad referrer header.
Blacklist can be found here.

7.5.4 DNSBL

`USE_DNSBL`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to *yes*, DNSBL checks will be performed to the servers specified in the `DNSBL_LIST` environment variable.

`DNSBL_LIST`
Values : *<list of DNS zones separated with spaces>*
Default value : *bl.blocklist.de problems.dnsbl.sorbs.net sbl.spamhaus.org xbl.spamhaus.org*
Context : *global*, *multisite*
The list of DNSBL zones to query when `USE_DNSBL` is set to *yes*.

7.5.5 CrowdSec

`USE_CROWDSEC`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
If set to *yes*, CrowdSec will be enabled. Please note that you need a CrowdSec instance running see example here.

`CROWDSEC_HOST`
Values : *<full URL to the CrowdSec instance API>*
Default value :
Context : *global*
The full URL to the CrowdSec API.

`CROWDSEC_KEY`
Values : *<CrowdSec bouncer key>*
Default value :
Context : *global*
The CrowdSec key given by *cscli bouncer add BouncerName*.

7.5.6 Custom whitelisting

`USE_WHITELIST_IP`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to *yes*, lets you define custom IP addresses to be whitelisted through the `WHITELIST_IP_LIST` environment variable.

`WHITELIST_IP_LIST`
Values : *<list of IP addresses and/or network CIDR blocks separated with spaces>*
Default value : *23.21.227.69 40.88.21.235 50.16.241.113 50.16.241.114 50.16.241.117 50.16.247.234 52.204.97.54 52.5.190.19 54.197.234.188 54.208.100.253 54.208.102.37 107.21.1.8*
Context : *global*, *multisite*
The list of IP addresses and/or network CIDR blocks to whitelist when `USE_WHITELIST_IP` is set to *yes*. The default list contains IP addresses of the DuckDuckGo crawler.

`USE_WHITELIST_REVERSE`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to *yes*, lets you define custom reverse DNS suffixes to be whitelisted through the `WHITELIST_REVERSE_LIST` environment variable.

`WHITELIST_REVERSE_LIST`
Values : *<list of reverse DNS suffixes separated with spaces>*
Default value : *.googlebot.com .google.com .search.msn.com .crawl.yahoot.net .crawl.baidu.jp .crawl.baidu.com .yandex.com .yandex.ru .yandex.net*
Context : *global*, *multisite*
The list of reverse DNS suffixes to whitelist when `USE_WHITELIST_REVERSE` is set to *yes*. The default list contains suffixes of major search engines.

`WHITELIST_USER_AGENT`
Values : *<list of regexes separated with spaces>*
Default value :

Context : *global*, *multisite*
Whitelist user agent from being blocked by BLOCK_USER_AGENT.

WHITELIST_URI
Values : *<list of URI separated with spaces>*
Default value :
Context : *global*, *multisite*
URI listed here have security checks like bad user-agents, bad IP, ... disabled. Useful when using callbacks for example.

7.5.7 Custom blacklisting

USE_BLACKLIST_IP
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to *yes*, lets you define custom IP addresses to be blacklisted through the BLACKLIST_IP_LIST environment variable.

BLACKLIST_IP_LIST
Values : *<list of IP addresses and/or network CIDR blocks separated with spaces>*
Default value :
Context : *global*, *multisite*
The list of IP addresses and/or network CIDR blocks to blacklist when USE_BLACKLIST_IP is set to *yes*.

USE_BLACKLIST_REVERSE
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to *yes*, lets you define custom reverse DNS suffixes to be blacklisted through the BLACKLIST_REVERSE_LIST environment variable.

BLACKLIST_REVERSE_LIST
Values : *<list of reverse DNS suffixes separated with spaces>*
Default value : *.shodan.io*
Context : *global*, *multisite*
The list of reverse DNS suffixes to blacklist when USE_BLACKLIST_REVERSE is set to *yes*.

7.5.8 Requests limiting

USE_LIMIT_REQ
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, the amount of HTTP requests made by a user for a given resource will be limited during a period of time.

LIMIT_REQ_URL
Values : *<any valid url>*
Default value :
Context : *global*, *multisite*
The URL where you want to apply the request limiting. Use special value of / to apply it globally for all URL.
You can set multiple rules by adding a suffix number to the variable name like this : LIMIT_REQ_URL_1, LIMIT_REQ_URL_2, LIMIT_REQ_URL_3, ...

LIMIT_REQ_RATE
Values : *Xr/s* | *Xr/m* | *Xr/h* | *Xr/d*

Default value : *1r/s*
Context : *global*, *multisite*
The rate limit to apply when `USE_LIMIT_REQ` is set to *yes*. Default is 1 request to the same URI and from the same IP per second. Possible value are : `s` (second), `m` (minute), `h` (hour) and `d` (day)).
You can set multiple rules by adding a suffix number to the variable name like this : `LIMIT_REQ_RATE_1`, `LIMIT_REQ_RATE_2`, `LIMIT_REQ_RATE_3`, ...

`LIMIT_REQ_BURST`
Values : *<any valid integer>*
Default value : *5*
Context : *global*, *multisite*
The number of requests to put in queue before rejecting requests.
You can set multiple rules by adding a suffix number to the variable name like this : `LIMIT_REQ_BURST_1`, `LIMIT_REQ_BURST_2`, `LIMIT_REQ_BURST_3`, ...

`LIMIT_REQ_DELAY`
Values : *<any valid float>*
Default value : *1*
Context : *global*, *multisite*
The number of seconds to wait before requests in queue are processed. Values like `0.1`, `0.01` or `0.001` are also accepted.
You can set multiple rules by adding a suffix number to the variable name like this : `LIMIT_REQ_DELAY_1`, `LIMIT_REQ_DELAY_2`, `LIMIT_REQ_DELAY_3`, ...

`LIMIT_REQ_CACHE`
Values : *Xm* | *Xk*
Default value : *10m*
Context : *global*
The size of the cache to store information about request limiting.

7.5.9 Connections limiting

`USE_LIMIT_CONN`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, the number of connections made by an ip will be limited during a period of time. (ie. very small/weak ddos protection)
More info connections limiting here.

`LIMIT_CONN_MAX`
Values : *<any valid integer>*
Default value : *50*
Context : *global*, *multisite*
The maximum number of connections per ip to put in queue before rejecting requests.

`LIMIT_CONN_CACHE`
Values : *Xm* | *Xk*
Default value : *10m*
Context : *global*
The size of the cache to store information about connection limiting.

7.5.10 Countries

`BLACKLIST_COUNTRY`
Values : *<country code 1> <country code 2> ...*
Default value :
Context : *global*, *multisite*
Block some countries from accessing your website. Use 2 letters country code separated with space.

`WHITELIST_COUNTRY`
Values : *<country code 1> <country code 2> ...*
Default value :
Context : *global*, *multisite*
Only allow specific countries accessing your website. Use 2 letters country code separated with space.

7.6 PHP

`REMOTE_PHP`
Values : *<any valid IP/hostname>*
Default value :
Context : *global*, *multisite*
Set the IP/hostname address of a remote PHP-FPM to execute .php files.

`REMOTE_PHP_PATH`
Values : *<any valid absolute path>*
Default value : */app*
Context : *global*, *multisite*
The path where the PHP files are located inside the server specified in `REMOTE_PHP`.

`LOCAL_PHP`
Values : *<any valid absolute path>*
Default value :
Context : *global*, *multisite*
Set the absolute path of the unix socket file of a local PHP-FPM instance to execute .php files.

`LOCAL_PHP_PATH`
Values : *<any valid absolute path>*
Default value : */app*
Context : *global*, *multisite*
The path where the PHP files are located inside the server specified in `LOCAL_PHP`.

7.7 Bad behavior

`USE_BAD_BEHAVIOR`
Values : *yes* | *no*
Default value : *yes*
Context : *global*, *multisite*
If set to yes, bunkerized-nginx will block users getting too much "suspicious" HTTP codes in a period of time.

`BAD_BEHAVIOR_STATUS_CODES`
Values : *<HTTP status codes separated with space>*
Default value : *400 401 403 404 405 429 444*
Context : *global*, *multisite*
List of HTTP status codes considered as "suspicious".

`BAD_BEHAVIOR_THRESHOLD`
Values : *<any positive integer>*
Default value : *10*
Context : *global*, *multisite*
The number of "suspicious" HTTP status code before the corresponding IP is banned.

`BAD_BEHAVIOR_BAN_TIME`
Values : *<any positive integer>*
Default value : *86400*
Context : *global*, *multisite*
The duration time (in seconds) of a ban when the corresponding IP has reached the `BAD_BEHAVIOR_THRESHOLD`.

`BAD_BEHAVIOR_COUNT_TIME`
Values : *<any positive integer>*
Default value : *60*
Context : *global*, *multisite*
The duration time (in seconds) before the counter of "suspicious" HTTP is reset.

7.8 Authelia

`USE_AUTHELIA`
Values : *yes* | *no*
Default value : *no*
Context : *global*, *multisite*
Enable or disable Authelia support. See the authelia example for more information on how to setup Authelia with bunkerized-nginx.

`AUTHELIA_BACKEND`
Values : *<any valid http(s) address>*
Default value :
Context : *global*, *multisite*
The public Authelia address that users will be redirect to when they will be asked to login (e.g. : `https://auth.example.com`).

`AUTHELIA_UPSTREAM`
Values : *<any valid http(s) address>*
Default value :
Context : *global*, *multisite*
The private Authelia address when doing requests from nginx (e.g. : http://my-authelia.local:9091).

`AUTHELIA_MODE`
Values : *portal* | *auth-basic*
Default value : *portal*
Context : *global*, *multisite*
Choose authentication mode : show a web page (`portal`) or a simple auth basic prompt (`auth-basic`).

7.9 misc

`SWARM_MODE`
Values : *yes* | *no*
Default value : *no*
Context : *global*
Only set to *yes* when you use *bunkerized-nginx* with Docker Swarm integration.

`KUBERNETES_MODE`
Values : *yes* | *no*
Default value : *no*
Context : *global*
Only set to *yes* when you use bunkerized-nginx with Kubernetes integration.

`USE_API`
Values : *yes* | *no*
Default value : *no*
Context : *global*
Only set to *yes* when you use bunkerized-nginx with Swarm/Kubernetes integration or with the web UI.

`API_URI`
Values : *random* | *<any valid URI path>*
Default value : *random*
Context : *global*
Only set to *yes* when you use bunkerized-nginx with Swarm/Kubernetes integration or with the web UI.

`API_WHITELIST_IP`
Values : *<list of IP/CIDR separated with space>*
Default value : *192.168.0.0/16 172.16.0.0/12 10.0.0.0/8*
Context : *global*
List of IP/CIDR block allowed to send API order using the `API_URI` uri.

`USE_REDIS`
Undocumented. Reserved for future use.

`REDIS_HOST`
Undocumented. Reserved for future use.

CHAPTER

EIGHT

TROUBLESHOOTING

8.1 Logs

When troubleshooting, the logs are your best friends. We try our best to provide user-friendly logs to help you understand what happened.

If you are using container based integrations, you can get the logs using your manager/orchestrator (e.g., docker logs, docker service logs, kubectl logs, ...). For Linux integration, everything is stored inside the `/var/log` folder.

You can edit the `LOG_LEVEL` environment variable to increase or decrease the verbosity of logs with the following values : `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` or `emerg` (with `debug` being the most verbose level).

8.2 Permissions

Don't forget that bunkerized-nginx runs as an unprivileged user with UID/GID 101 when using container based integrations or simply the `nginx` user on Linux. Double check the permissions of files and folders for each special folders (see the volumes list).

8.3 ModSecurity

The OWASP Core Rule Set can sometimes leads to false positives. Here is what you can do :

- Check if your application has exclusions rules (e.g : wordpress, nextcloud, drupal, ...)
- Edit the matched rules to exclude some parameters, URIs, ...
- Remove the matched rules if editing it is too much a hassle

Some additional resources :

- Wordpress example
- Handling false positive
- Adding exceptions and tuning

8.4 Bad behavior

The bad behavior feature comes with a set of status codes considered as "suspicious". You may need to tweak the corresponding list to avoid false positives within your application.

8.5 Whitelisting

It's a common case that a bot gets flagged as suspicious and can't access your website. Instead of disabling the corresponding security feature(s) we recommend a whitelisting approach. Here is a list of environment variables you can use :

- `WHITELIST_IP_LIST`
- `WHITELIST_REVERSE_LIST`
- `WHITELIST_URI`
- `WHITELIST_USER_AGENT`

More information here.

CHAPTER

NINE

PLUGINS

Bunkerized-nginx comes with a plugin system that lets you extend the core with extra security features.

9.1 Official plugins

- ClamAV : automatically scan uploaded files and deny access if a virus is detected
- CrowdSec : CrowdSec bouncer integration within bunkerized-nginx

9.2 Community plugins

If you have made a plugin and want it to be listed here, feel free to create a pull request and edit that section.

9.3 Use a plugin

The generic way of using a plugin consists of :

- Download the plugin into your local drive (e.g., git clone)
- Edit the settings inside the plugin.json files (e.g., myplugin/plugin.json)
- If you are using a container based integration, you need to mount it to the plugins special folder inside the container (e.g., /where/is/myplugin:/plugins/myplugin)
- If you are using Linux integration, copy the downloaded plugin folder to the plugins special folder (e.g., cp -r myplugin /plugins)

To check if the plugin is loaded you should see log entries like that :

```
2021/06/05 09:19:47 [error] 104#104: [PLUGINS] *NOT AN ERROR* plugin MyPlugin/1.0 has␣
↪been loaded
```

9.4 Write a plugin

A plugin is composed of a plugin.json which contains metadata (e.g. : name, settings, ...) and a set of LUA files for the plugin code.

9.4.1 plugin.json

```
{
        "id": "myplugin",
        "name": "My Plugin",
        "description": "Short description of my plugin.",
        "version": "1.0",
        "settings": {
                "MY_SETTING": "value1",
                "ANOTHER_SETTING": "value2",
        }
}
```

The `id` value is really important because it must match the subfolder name inside the `plugins` volume. Choose one which isn't already used to avoid conflicts.

Settings names and default values can be choosen freely. There will be no conflict when you retrieve them because they will be prefixed with your plugin id (e.g. : `myplugin_MY_SETTING`).

9.4.2 Main code

```
local M                = {}
local logger        = require "logger"

-- this function will be called at startup
-- the name MUST be init without any argument
function M.init ()
        -- the logger.log function lets you write into the logs
        -- only ERROR level is available in init()
        logger.log(ngx.ERR, "MyPlugin", "*NOT AN ERROR* init called")

        -- here is how to retrieve a setting
        local my_setting = ngx.shared.plugins_data:get("pluginid_MY_SETTING")
        logger.log(ngx.ERR, "MyPlugin", "*NOT AN ERROR* my_setting = " .. my_setting)

        return true
end

-- this function will be called for each request
-- the name MUST be check without any argument
function M.check ()

        -- the logger.log function lets you write into the logs
        logger.log(ngx.NOTICE, "MyPlugin", "check called")

        -- here is how to retrieve a setting
```

(continues on next page)

(continued from previous page)

```
        local my_setting = ngx.shared.plugins_data:get("pluginid_MY_SETTING")

        -- a dummy example to show how to block a request
        if my_setting == "block" then
                ngx.exit(ngx.HTTP_FORBIDDEN)
        end
end

return M
```

That file must have the same name as the `id` defined in the plugin.json with a .lua suffix (e.g. : `myplugin.lua`).

Under the hood, bunkerized-nginx uses the lua nginx module therefore you should be able to access to the whole **ngx.*** functions.

9.4.3 Dependencies

Since the core already uses some external libraries you can use it in your own plugins too (see the compile.sh file and the core lua files).

In case you need to add dependencies, you can do it by placing the corresponding files into the same folder of your main plugin code. Here is an example with a file named **dependency.lua** :

```
local M = {}

function M.my_function ()
        return "42"
end

return M
```

To include it from you main code you will need to prefix it with your plugin id like that :

```
...
local my_dependency = require "pluginid.dependency"

function M.check ()
        ...
        local my_value = my_dependency.my_function()
        ...
end
...
```