Whalebone admin guide

Release 3.2.1-12

Jun 04, 2019

Contents:

Whalebone is a service for security filtering of DNS traffic. It uses logic on top of own DNS resolvers. Such resolvers could be either cloud ones maintained directly by Whalebone, or on-premise software resolvers using cloud just for threat intelligence updates and reporting. For threat prevention Whalebone relies on external intelligence sources as well as on own methods. More information about the product and company is available on the official Whalebone website

CHAPTER 1

Deployment options

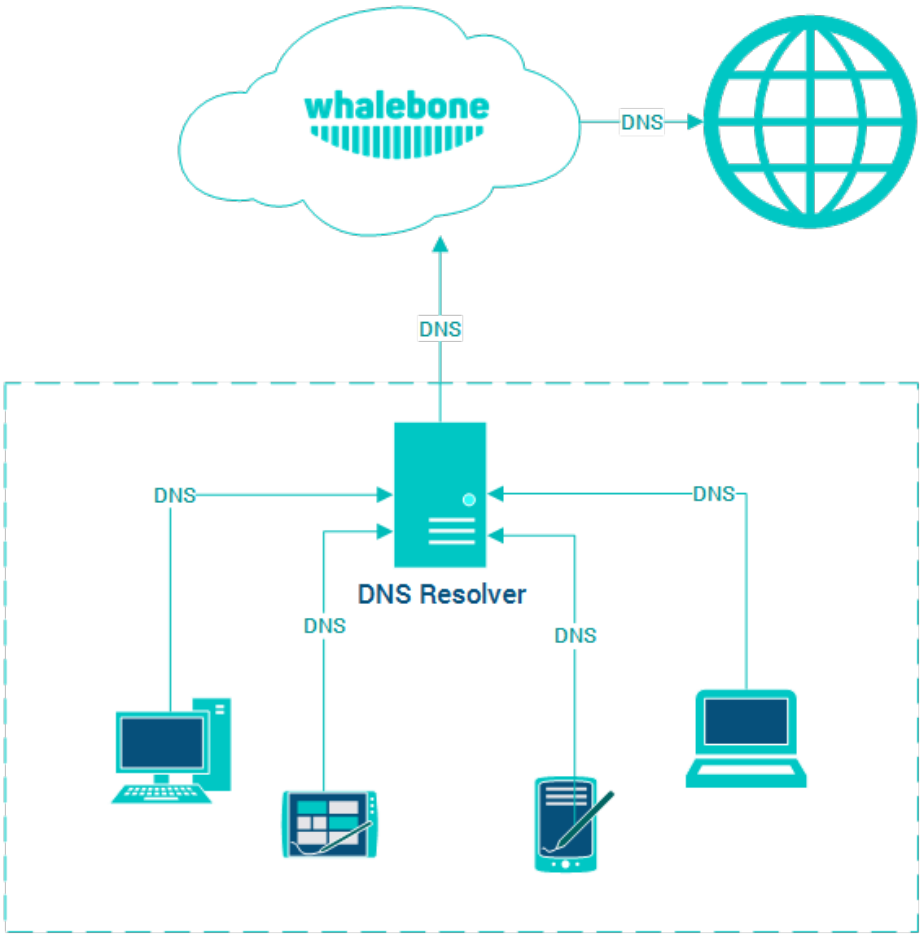
Whalebone could be deployed in several scenarios which can be even combined to satisfy requirements of particular networks. Combination of cloud and local DNS resolver with single management console will satisfy even complex and distributed networks.

Tip: All of the options below could be combined together. Various network segments and zones could have different requirements and possibilities.

Tip: Should you believe none of the scenarios below is applicable to your use case, please contact Whalebone and we will help you with architecture that will suite your needs.

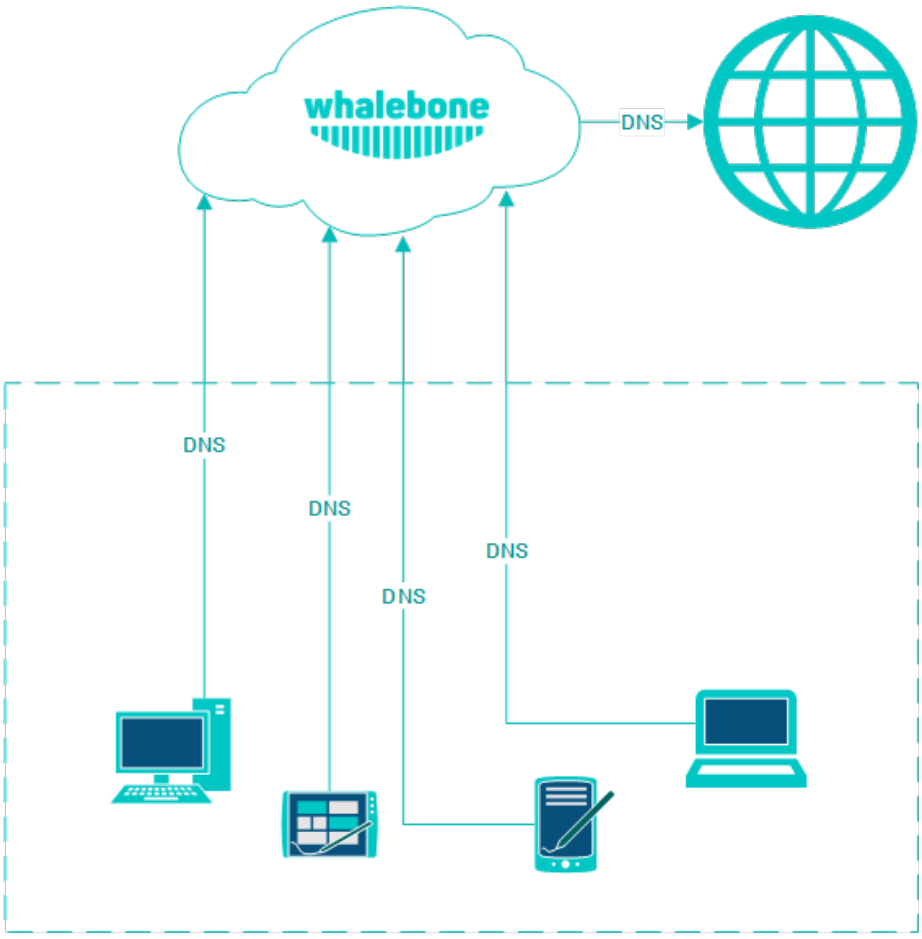
1.1 Cloud DNS

This is the simplest method o deployment. To use Whalebone filtering, just change the configuration of your recent DNS resolvers and point them to Whalebone cloud resolvers. The downside of this deployment is that all of the incidents will be visible with source IP of the DNS forwarder instead of the original source IP. Still this deployment could come in handy if the priority is to prevent the threats with as low effort and infrastructure changes as possible.



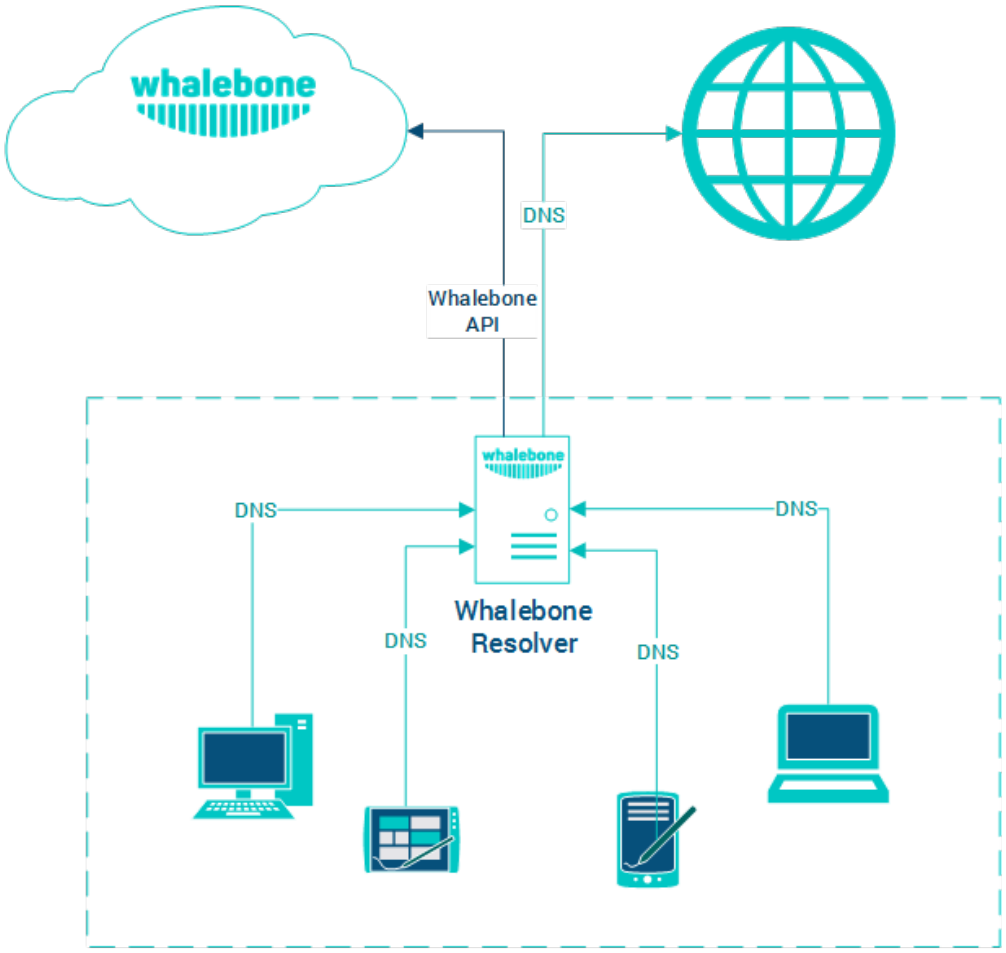
1.2 Cloud DNS (direct connection)

This deployment is similar to forwarding the requests to Whalebone cloud resolvers, but the requests are sent directly to the cloud without local DNS cache. This could be usually set for all endpoints through DHCP. However not using local DNS cache means increased latency introduced by the network communication between the client and cloud resolver. If the individual machines are not hidden behind a NAT, their IP addresses will be directly visible in the Whalebone reporting and the clients could be easilly distinguished.



1.3 Local resolver

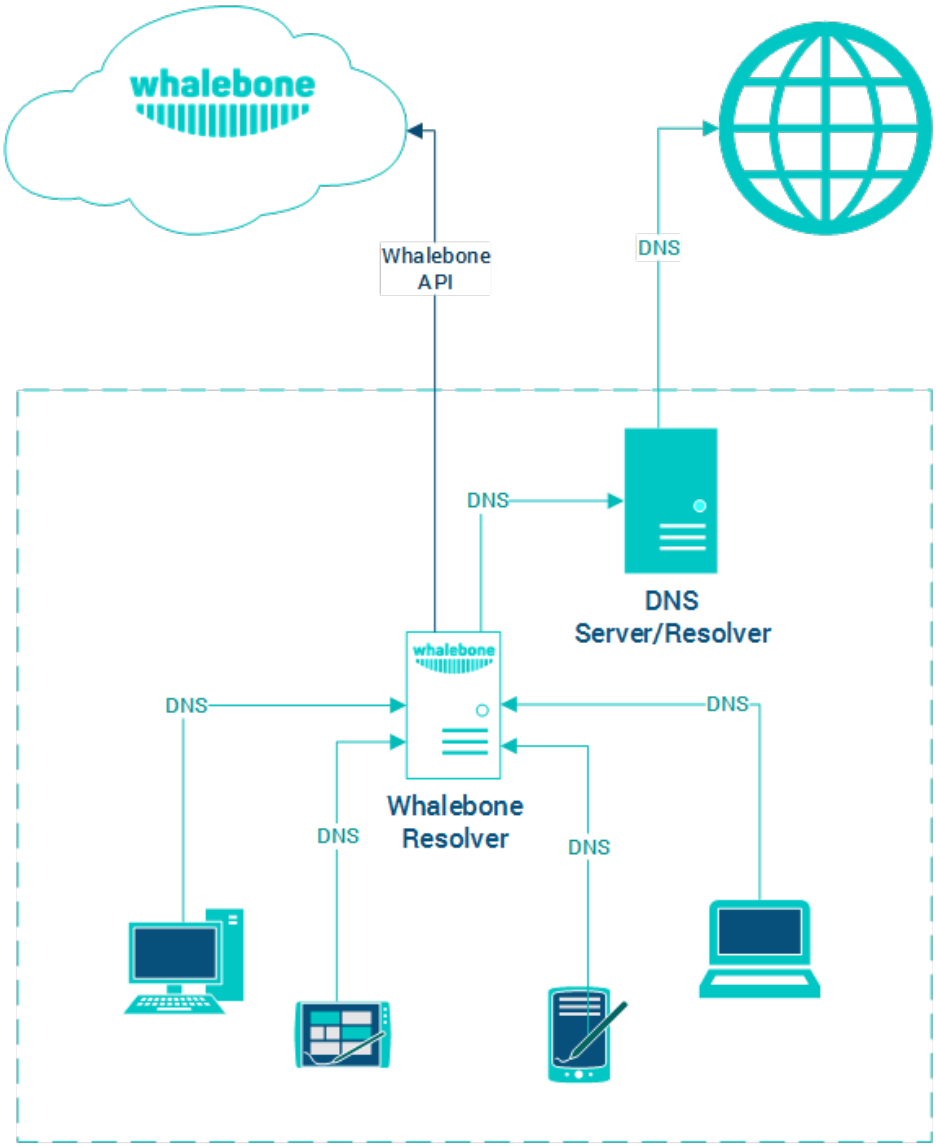
This deployment scenario uses local Whalebone resolver, that communicates with Whalebone cloud through API. The DNS resolution takes place directly on the resolver and is completely independent on the cloud availability. Should the resolver not be able to reach the cloud service, it won't be able to update the threat intelligence and to reports any incidents. The main advantage of this deployment is visibility into local network and individual IP addresses and native DNS resolver latency.



1.4 Local forwarder

Very similar deployment scenario as the local resolver, however Whalebone just forwards the requests to preconfigured resolvers. This scenario is very useful in case there are local DNS zones that has to be available for the clients (e.g. Active Directory) or cases when the recent resolver configuration is very specific and has to be preserved. This deployment has also lower hardware requirements, roughly half of the CPU and RAM recommended.

> **Warning:** We don't recommend to forward the requests to Whalebone cloud resolvers. Such configuration would result in duplicit incident detection, no added security and unnecesary latency for the clients.

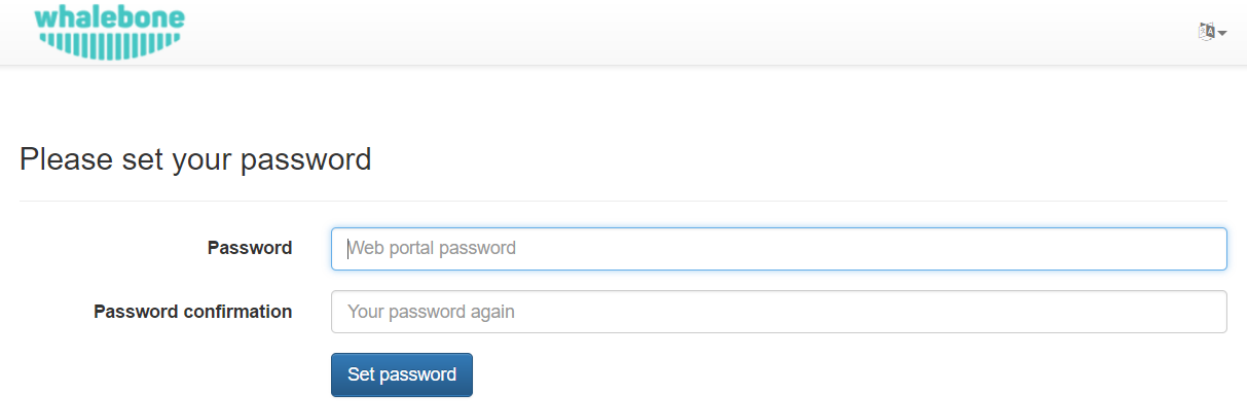

whalebone
Whalebone
API
DNS
DNS
DNS
Server/Resolver
whalebone
DNS
DNS
Whalebone
Resolver
DNS
DNS

CHAPTER 2

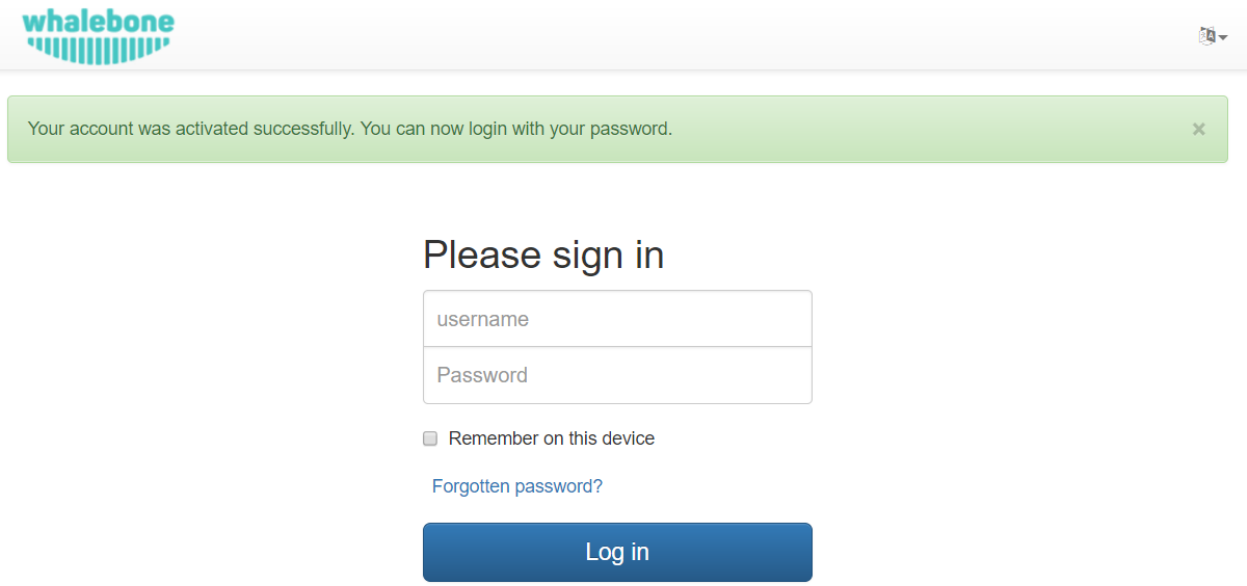
Quickstart

2.1 Creating the portal account

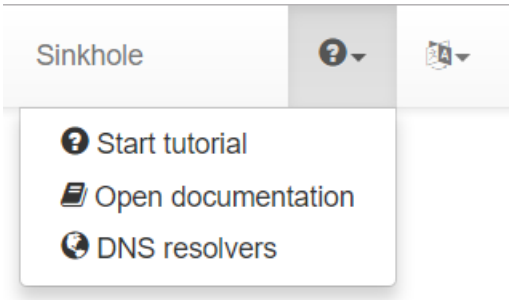
After accessing the URL from your activation email, you will be asked to setup the password for your account. We don't enforce any password complexity, but we recommend using unique and non-trivial password. An unauthorized access would be a threat to users privacy and could misuse the configuration to harm your network.



After the password setup you will be asked to login using your username and newly created password.

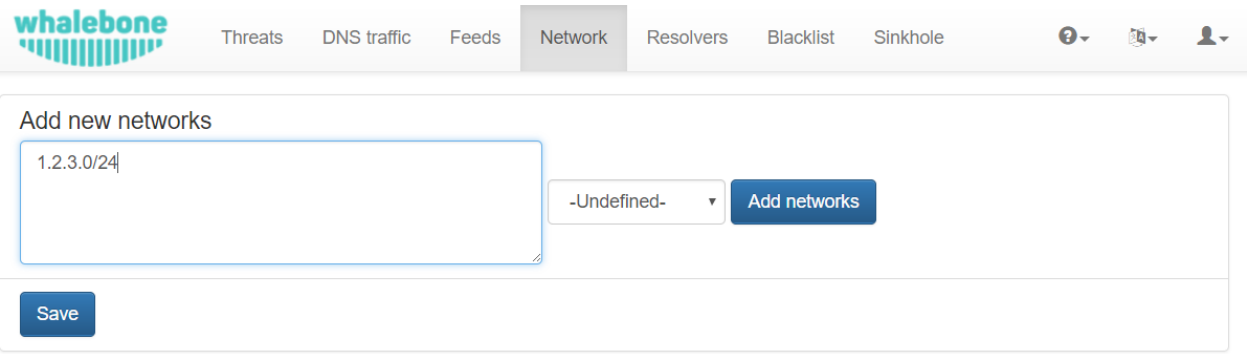


Right after the first login the tutorial will pop up and will guide you through the very basics of the Whalebone portal. You can skip the tutorial any time and get back to it later from the main menu (the question mark icon) option `Start tutorial`.



2.2 Public network ranges

Public network range definition serves to distinguish individual customers and their users. It is necessary to include all the public network ranges that will be used by DNS resolver as well as the users browsing the internet. The definition is used to customize the block page appearance (described later). Single customer can manage more network ranges, such ranges can be assigned to localities to easily distinguish between logical network zones in DNS traffic audit and incidents.



Warning: Should you not fill in your public network ranges, cloud resolvers will serve as a simple DNS resolvers without any filtering. If you use local resolvers, you still have to input your network ranges to display fully customized blocking page (sinkhole) to the blocked users.

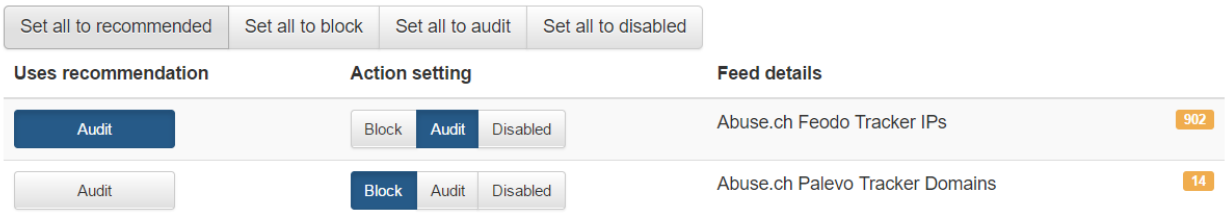
- Into the field `Add new network` insert one or more network ranges using notation <network address>/<mask>, e.g.: `198.51.100.0/24`
- Press button `Add networks` to add changes
- Don't forget to save your new setup through the `Save` button

Tip: While testing Whalebone (e.g. through adding a testing domain into blacklist) don't forget that many DNS records could be recently in the DNS cache anywhere between the resolver and the user (including the browser, operating system or forwarders). Testing right after the configuration change could therefore fail and the timespan before the protection becomes active could vary based on the TTL of the particular DNS record (should all the caches along the way actually honor the TTL value).

2.3 Feed filtering options

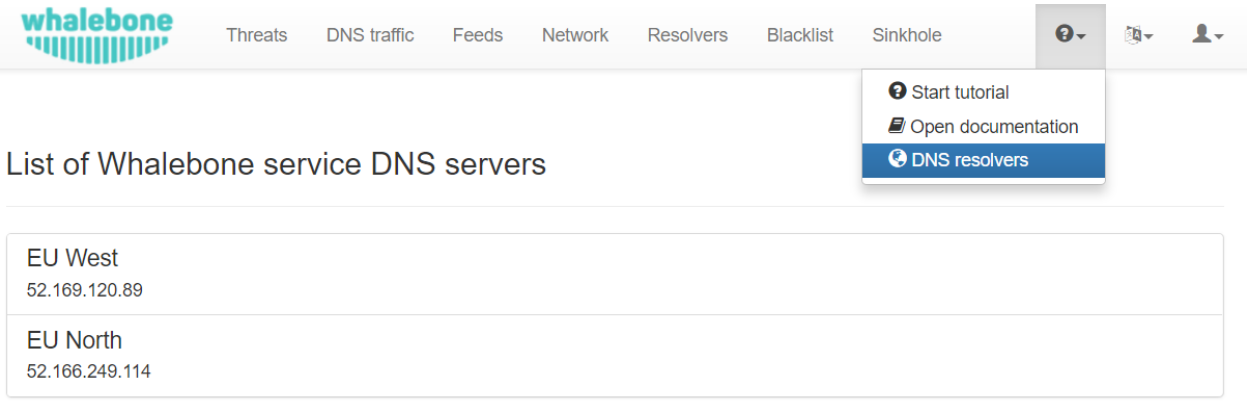
Every Threat Intelligence Feed could be setup in a different manner. As long as the button `Uses recommendation` is pressed the configuration is left to the Whalebone specialist and it honors the Whalebone best practice. If you prefer your own configuration you can choose from these three options:

- **Block**
- **Audit**
- **Disabled**



2.4 Cloud DNS resolvers

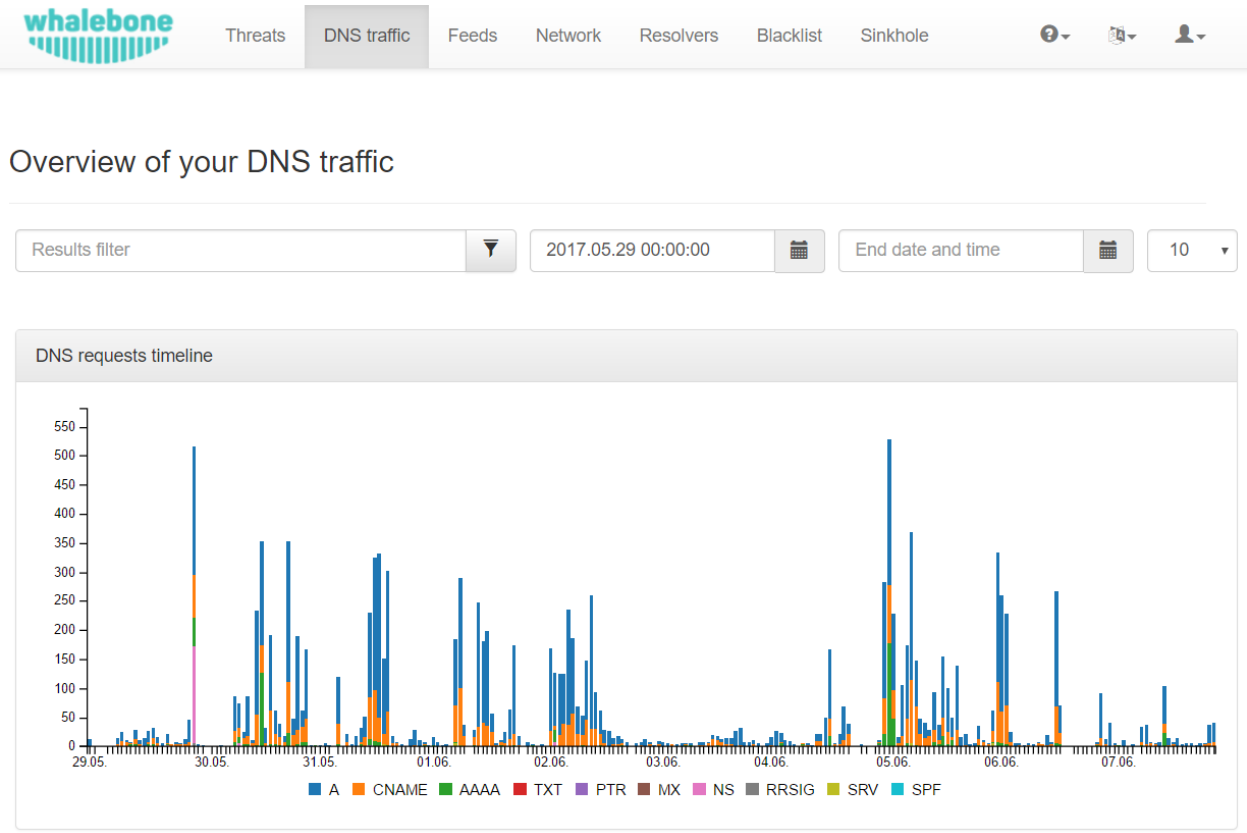
You should forward your DNS traffic towards Whalebone cloud resolvers if this is your preferred deployment option. Cloud resolver are available on two independent IP addresses: `52.169.120.89` `52.166.249.114`



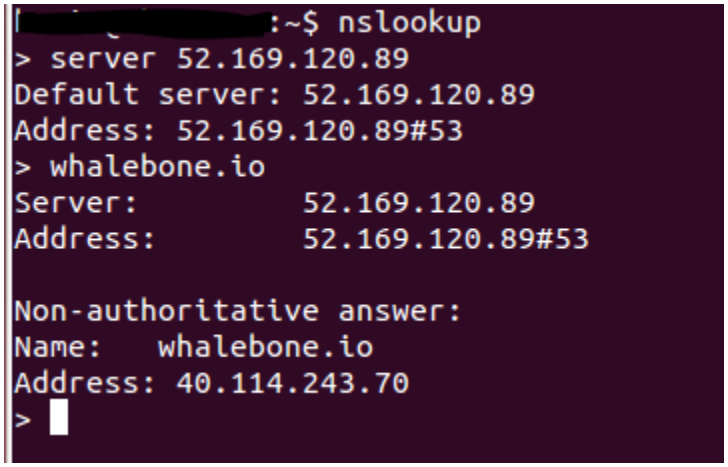
Always use both IP addresses in your configuration The guarantee of availability of the DNS resolution service is applied only in cases where both of the IP addresses are configured to use primary and secondary resolvers automatically.

2.5 DNS traffic

Should the traffic be properly forwarded on Whalebone DNS resolvers (cloud or local) the DNS traffic will be visible under the menu option `DNS traffic`, where the individual request and responses are available for further investigation. The traffic should be visible in several minutes after everything has been properly setup. If there is no traffic recorded even in several hours don't hesitate to contact Whalebone support to help you doublecheck the configuration or any sort of network issues.



The DNS resolution check could be also done manually on Windows or Linux machines through `nslookup` tool. Set the Whalebone resolver IP and try to resolve an existing domain name.



CHAPTER 3

Local resolver

Whalebone local resolver brings the advantage of visibility of local IP addresses that send the actual requests. Whalebone resolver is based on the implementation of Knot Resolver developed in the CZ.NIC labs.

3.1 System requirements

Local resolver is supported on dedicated (hardware or virtual) machine running a supported operating system.

- **Supported operating system** (64-bit, server editions of following distributions):
  - Red Hat Enterprise Linux 7
  - CentOS 7
  - Debian 9
  - Ubuntu 16.04, 18.04
- **Supported filesystems**
  - ext4
  - xfs only with d_type support (ftype=1)
- **Recommended hardware sizing** for usual traffic (physical or virtual):
  - 2 CPU cores
  - 4 GB RAM
  - 40 GB HDD (at least 30 GB in /var partition)
- **Network setup requirements** (local resolver needs the following ports opened):
  - `TCP+UDP/53` into the internet destinations if responsible for the resolution
  - `TCP/8443` to `resolverapi.whalebone.io`

  - `TCP/443` to `logger.whalebone.io, agentapi.whalebone.io, portal.whalebone.io, index.docker.io, registry-1.docker.io, data.iana.org`
  - Reachability of software repositories for the operating system

Warning: Without communication on port 8443 and 443 to the domains listed above the resolver won't be installed at all (the installation script will abort).

With recommended hardware resources the resolver will provide stable and fast DNS resolution and filtering. Resolver can be run with significantly lower resources, but that is recommended just for low volume testing environments.

Note: Should you need sizing estimation for large ISP or Enterprise network contact Whalebone. Whalebone local resolver will need approx. twice the RAM and CPU than usual resolver (BIND, Unbound).

3.2 Installation of a new resolver

In menu **Resolvers** press the button **Create new**. Choose a name (identifier) for your new resolver. The input is purely informative and won't affect the functionality. Once you've entered the name, click **Add resolver** button After clicking the button an informative window will pop up with list of supported platforms and the one-line command for the installation. Copy the command and run on the machine dedicated for the local resolver. The command will run the installation script and will pass the one time token used for the resolver activation (the same command can not be used repeatedly).

Once the command is run the operating system is being checked and requirements installed. Script will inform you about the progress and it creates a detailed log named `wb_install.log` in current directory. Successul run of the installation script is ended with the notification `` `Final tuning of the OS` `` with value `[ OK ]`. Right after the installation also the initialization takes place and it could take several minutes before the resolver starts the services.

Warning: Local resolver is configured as an open resolver. It will respond to any request sent. This is quite comfortable in terms of availability of the services, but also could be a risk if the service is available from the outside networks. Please make sure you limit the access to the local resolver on port 53 (UDP and TCP) from the trusted networks only, otherwise it can be misused for various DoS attacks.

3.3 Security policies

The behavior of DNS filtering on the resolvers could be defined in the menu item **Configuration** and tab **Security poicies**. In the default state there is only the **Default policy**, which is automatically assigned to any new resolver. In any policy there are several options to be defined:

- **Malicious domains filtering**
  - Allows to apply actions Audit (logging) or Block (redirect to blocking page) on resolution of malicious domains
  - Individual actions could be turned off - e.g. turn off the blocking for testing purposes

  - The slider values define the probability that the particular domain is malicious on the scale from 0 to 100 (0 is a safe domain, 100 is malicious)

Tip: The default threshold for blocking is set to `80` which is safe even for larger network with liberal policy towards the users. For more restrictive policy we suggest setting threshold for blocking to `70-75`, in very restrictive networks even down to `60`. Audit is purely informative, however setting the threshold too low can result in too many logged incidents.

- **Lists of blocked domains**
  - Lists of domains, that has to be blocked
  - Such domains do not have to be malicious, it could be just domains blokced based on legal requirements
  - These lists are regularly updated by Whalebone
- **Whitelist**
  - Domains that won't be blocked at any time
  - The whitelist is applied to the domain and all of the subdomains, e.g.: whitelisted domain `whalebone.io` will also whitelist `docs.whalebone.io`, but not vice versa
- **Blacklist**
  - Domains that will be blocked at all times (higher priority has only **Whitelist**)
  - The blacklist is applied to the domain and all of the subdomains, e.g.: whitelisted domain `malware.ninja` will also blacklist `super.malware.ninja`, but not vice versa

---

Note: Changes will be applied to the resolvers in approx. 30 minutes. Saved configuration is used during preparation of the threat data package for the resolvers that download and apply those packages at regular intervals.

3.4 DNS resolution configuration

You can find the options to configure the resolver in the menu **Configuration** and tab **DNS resolution**. This page allows you to do the basic configuration without the knowledge of configuration syntax. Furthermore there is a text area allowing you to define any configuration to the underlying Knot Resolver.

Available configuration options:

- **Enable IPv6**
  - Should the system has the IPv6 properly configured and working, it is possible to enable it. Otherwise the activation of IPv6 could have negative effects on the performance and latency of the resolver.
- **Forward queries to**
  - This option allows to redirect all or chosen queries to upstream resolvers or authoritative DNS servers (suitable e.g. for forwarding to domain controllers of Active Directory)
  - **Disable DNSSEC**
    - If checked, the answers from the forwarded queries won't be DNSSEC validated. We recommend to check this option should the upstream server have not DNSSEC configured properly.
  - **All queries to**

        * Option to forward all queries to one or more resolver
    - **Following domains**
        * Option to choose particular domains that should be forwarded to on more resolvers
        * Different resolvers could be defined for different domains
- **Static records**
    - Predefined answers that should be returned for particular domains
    - Could serve for special purposes such as monitoring or very simple substition of records on authoritative server
- **Advanced DNS configuration**
    - Text area for complete Knot Resolver configuration
    - Supports Lua scripting
    - Faulty configuration can impact stability, performance or security functions of the resolver

3.5 Resolver management

On the **Resolvers** page there is an overview of created resolvers. Administrator can adjust the configuration, deploy updates and install new resolvers.

3.5.1 Resolvers overview

In the main resolver overview there are tiles with resolver details and configuration options. The overview includes information about operating system and resources as CPU, Memory and HDD usage. Therre is also the state of services running on the resolvers (should state "Running" if everything is OK) and the status of the communication channel between the resolver and the cloud (it is expected to be "Active").

3.5.2 Deploy configuration

Should you change any configuration related to the DNS resolution, you have to deploy the configuration afterwards. If there are any configuration changes available to be deployed, there will be a red icon with down right arrow visible on the resolver card. Once clicked, the webpage will ask for confirmation and the successful deployment will be notified in the top right corner.

Note: If the result is an deployment error, try to repeat the action. The reason for the error could be a short term communication outage between the cloud and the resolver.

3.6 Resolver agent

3.6.1 Command line interface

Agent's actions can be invoked using a proxy bash script present at path **/var/whalebone/cli**. This script calls a python script which handles the execution of the following agent actions:

- **sysinfo - returns the system status data in JSON format.**
  - Parameters: None
  - Output: tested categories on tested key can have two values ‘ok’ and ‘fail’


```
{
   "hostname":"hostname",
   "system":"Linux",
   "platform":"CentOS Linux 7 (Core)",
   "cpu":{
      "count":4,
      "usage":28.6
   },
   "memory":{
      "total":7.6,
      "available":3.9,
      "usage":49.2
   },
   "hdd":{
      "total":50.0,
      "free":14.4,
      "usage":71.1
   },
   "swap":{
      "total":0.0,
      "free":0.0,
      "usage":0
   },
   "resolver":{
      "answer.nxdomain":3284,
      "answer.tc":35,
      "answer.ad":849,
      "answer.100ms":3983,
      "answer.cd":6,
      "answer.1500ms":74,
      "answer.slow":215,
      "answer.rd":224337,
      "answer.1ms":104683,
      "answer.servfail":215,
      "predict.epoch":24,
      "query.dnssec":6,
      "answer.250ms":14941,
      "query.edns":35498,
      "answer.cached":86713,
      "answer.nodata":3622,
      "answer.aa":2362,
      "answer.do":6,
      "answer.edns0":35498,
      "answer.ra":224337,
      "predict.queue":0,
      "answer.total":224337,
      "answer.10ms":35351,
      "answer.noerror":217216,
      "answer.50ms":59766,
      "answer.500ms":4642,
      "answer.1000ms":653,
      "predict.learned":80
   },
```


(continues on next page)

(continued from previous page)


```
   "docker":{
      "Platform":{
         "Name":""
      },
      "Components":[
         {
            "Name":"Engine",
            "Version":"17.12.1-ce",
            "Details":{
               "ApiVersion":"1.35",
               "Arch":"amd64",
               "BuildTime":"2022-02-27T22:17:54.000000000+00:00",
               "Experimental":"false",
               "GitCommit":"88888fc6",
               "GoVersion":"go1.999.999",
               "KernelVersion":"3.22.66-693.21.1.el7.x86_64",
               "MinAPIVersion":"1.99",
               "Os":"linux"
            }
         }
      ],
      "Version":"19.32.1-ce",
      "ApiVersion":"1.98",
      "MinAPIVersion":"1.12",
      "GitCommit":"7390fc6",
      "GoVersion":"go1.9.4",
      "Os":"linux",
      "Arch":"amd64",
      "KernelVersion":"3.10.0-693.21.1.el7.x86_64",
      "BuildTime":"2018-02-27T22:17:54.000000000+00:00"
   },
   "check":{
      "resolve":"ok",
      "port":"ok"
   },
   "containers":{
      "lr-agent":"running",
      "passivedns":"running",
      "resolver":"running",
      "kresman":"running",
      "pcpy":"running",
      "logrotate":"running",
      "logstream":"running"
   },
   "images":{
      "lr-agent":"whalebone/agent:1.1.1",
      "passivedns":"whalebone/passivedns:1.1.1",
      "resolver":"whalebone/kres:1.1.1",
      "kresman":"whalebone/kresman:1.1.1",
      "logrotate":"whalebone/logrotate:1.1.1",
      "logstream":"whalebone/logstream:1.1.1"
   },
   "error_messages":{
   },
   "interfaces":[
      {
         "name":"lo",
```


(continues on next page)

(continued from previous page)


```
            "addresses":[
               "127.0.0.1",
               "::1",
               "00:00:00:00:00:00"
            ]
         },
         {
            "name":"eth0",
            "addresses":[
               "1.1.1.1",
               "::c8",
               "fe80::",
               "00:00:00:00:00:00"
            ]
         },
         {
            "name":"docker0",
            "addresses":[
               "198.1.1.1",
               "00:00:00:00:00:00"
            ]
         }
      ]
}
```


- **stop - stops up to three containers**
  - Parameters: containers to stop (up to 3), Example: ./cli.sh stop resolver lr-agent kresman
  - Output:


```
{
        'resolver': {'status': 'success'},
        'lr-agent': {'status': 'success'},
        'kresman': {'status': 'success'}
}
```


- **remove - removes up to three containers**
  - Parameters: containers to remove (up to 3), Example: ./cli.sh remove resolver lr-agent kresman
  - Output:


```
{
        'resolver': {'status': 'success'},
        'lr-agent': {'status': 'success'},
        'kresman': {'status': 'success'}
}
```


- **upgrade - upgrades up to three containers, the container's configuration is specified by a docker-compose in agent contain**
  - Parameters: containers to upgrade (up to 3), Example: ./cli.sh upgrade resolver lr-agent kresman
  - Output:


```
{
        'resolver': {'status': 'success'},
        'lr-agent': {'status': 'success'},
```


(continues on next page)

(continued from previous page)

```
        'kresman': {'status': 'success'}
}
```

- **create - creates containers, the containers are specified by a docker-compose in agent container (can also be found in /etc/v**
  - Parameters: None, Example: ./cli.sh create
  - Output:

```
{'resolver': {'status': 'success'}
```

- **list - lists the awaiting command and the changes that would be made to the containers specified in the awaiting action, thi**
  - Parameters: None, Example: ./cli.sh list
  - Output:

```
--------------------------------
Changes for resolver
New value for label: resolver-1.1.1
        Old value for label: resolver-1.0.0
--------------------------------
```

- **run - executes the awaiting command**
  - Parameters: none, Example: ./cli.sh run

```
{'resolver': {'status': 'success'}
```

- **updatecache - forces the update of resolver's IoC cache (which is used for blocking), this action should be done to manuall**
  - Parameters: None
  - Output:

```
{'status': 'success', 'message': 'Cache update successful'}
```

- **containers - lists the containers and their information which include: labels, image, name and status.**
  - Parameters: None
  - Output:

```
[
   {
      "id":"b8f4489379",
      "image":{
         "id":"c893b4df5ca3",
         "tags":[
            "whalebone/agent:1.1.1"
         ]
      },
      "labels":{
         "lr-agent":"1.1.1"
      },
      "name":"lr-agent",
```

(continues on next page)

(continued from previous page)


```
      "status":"running"
   },
   {
      "id":"e433d58f13",
      "image":{
         "id":"2c4b84a7daee",
         "tags":[
            "whalebone/passivedns:1.1.1"
         ]
      },
      "labels":{
         "passivedns":"1.1.1"
      },
      "name":"passivedns",
      "status":"running"
   },
   {
      "id":"2aeec00121",
      "image":{
         "id":"fc442e625539",
         "tags":[
            "whalebone/kres:1.1.1"
         ]
      },
      "labels":{
         "resolver":"1.1.1"
      },
      "name":"resolver",
      "status":"running"
   },
   {
      "id":"662dac2e6c",
      "image":{
         "id":"b37d0d1bd10b",
         "tags":[
            "whalebone/kresman:1.1.1"
         ]
      },
      "labels":{
         "kresman":"1.1.1"
      },
      "name":"kresman",
      "status":"running"
   },
   {
      "id":"05188ac1df",
      "image":{
         "id":"5b50cdc924fc",
         "tags":[
            "whalebone/logrotate:1.1.1"
         ]
      },
      "labels":{
         "logrotate":"1.1.1"
      },
      "name":"logrotate",
      "status":"running"
```


(continues on next page)

(continued from previous page)


```
    },
    {
        "id":"01e64dd697",
        "image":{
            "id":"fffb52c2dadd",
            "tags":[
                "whalebone/logstream:1.1.1"
            ]
        },
        "labels":{
            "logstream":"1.1.1"
        },
        "name":"logstream",
        "status":"running"
    }
]
```


Each of those actions execute similarly named actions and the status of that action, or output of that action, is printed. The **list** and **run** actions are intended for the scenario when a confirmation of a certain action is required. The action list shows the action that should be executed and the changes that would be done by that action for containers specified in that action. This serves as an example of what would happen if the awaiting action would have been executed. The run action then executes the awaiting action cleans up afterwards.

The actions of upgrade and create use the docker-compose template present in the agent container to create/upgrade the desired container. This template is mounted in the volume **/etc/whalebone/agent** if the user decides to change it. However this change needs to be done also to the template present at **portal.whalebone.io**, if not than the local changes will be overwritten from the cloud during next upgrade.

The bash script should be invoked like this: **./cli.sh action param1 param2 param3**. Action is the action name and parameters are the action parameters. Only actions for container stop, remove and upgrade use these and specify what containers should be affected by the respective action.

3.7 Knot Resolver - Tips & Tricks

Advanced configuration of Whalebone resolver allows to apply any Knot Resolver configuration. In this section we are going to describe the most frequent use cases and examples of such configuration snippets. Views, policies and their actions are evaluated in the sequence as they are defined (except special chain actions that are described in the official Knot Resolver documentation). First match will execute the action, the rest of the policy rules is not evaluated. If you are going to combine different configuration snnippets, you can load the same module just once at the beginning of the configuration.

3.7.1 Allow particular IP ranges

Define a list of IP ranges that will be allowed to use this DNS resolver. Queries from all other ranges will be refused.

```
-- load modules
modules = {'policy', 'view'}

--define list of ranges to allow
allowed = {
  '10.10.20.5/32',
  '10.30.10.0/24'
}
```

(continues on next page)

(continued from previous page)

```
-- allow list of ranges
for i,subnet in ipairs(allowed) do
  view:addr(subnet, policy.all(policy.PASS))
end

-- block all other ranges
view:addr('0.0.0.0/0', policy.all(policy.DENY))
```

3.7.2 Refuse particular IP ranges

Define a list of IP ranges that will be blocked to use this DNS resolver. Queries from all other ranges will be allowed.

```
-- load modules
modules = {'policy', 'view'}

--define list of ranges to block
blocked = {
  '10.10.20.5/32',
  '10.30.10.0/24'
}

-- block list of ranges
for i,subnet in ipairs(blocked) do
  view:addr(subnet, policy.all(policy.REFUSE))
end
```

3.7.3 Allow list of domains

```
-- load modules
modules = {'policy'}

--define list of allowed domains
domains = {
  'example.com',
  'anotherexample.org'
}

-- allow list of domains
for i,domain in ipairs(domains) do
  policy.suffix(policy.PASS, {todname(domain)})
end
```

3.7.4 Disable DNSSEC globally

```
trust_anchors.negative = { '.' }
```

3.7.5 Outgoing IP address

CHAPTER 4

End user dashboard

Whalebone also provides web UI for the end users. This is usually done in an environment, where ISP or Telco allows customers to configure the security service. UI is fully customizable and can be whitelabeled by the service provider. The integration is done by Whalebone specialists directly with cooperation of the service provider team.

4.1 Integration requirements

There are three four main points of integration of the Whalebone security service with the existing infrastructure and services.

- **DNS resolvers user awareness**
  - Usually done through gathering of authentication audit (e.g. RADIUS)
  - Static IP x User assignemnts are supported and are delivere through API integration (see below)
  - Other authentication services and methods are supported on request
- **End user dashboard**
  - Simple UI to provide the user with configuration and reporting options
  - Authentication through Single sign-on from the service provider existing interface
  - Fully customizable to follow the look and feel of the service provider
- **Blocking page**
  - Simple UI to inform the user about the security incident
  - Fully customizable to follow the look and feel of the service provider
  - Optional Bypass button to allow the user to continue to the destination website
- **API integration**
  - Whalebone offers API to receive calls about subscribed and unsubscribed users
  - Whalebone is able to send API calls containing user alerts and reports