

---

# **vespa Documentation**

*Release 1.0*

**Aurélien Wailly**

December 16, 2014



<b>1</b>	<b>vespa package</b>	<b>3</b>
1.1	Submodules	3
1.2	vespa.aes_gcm module	3
1.3	vespa.agent module	3
1.4	vespa.agent_av module	3
1.5	vespa.agent_bandwidth module	4
1.6	vespa.agent_connections module	4
1.7	vespa.agent_controller module	5
1.8	vespa.agent_controller_floodlight module	5
1.9	vespa.agent_controller_pox module	6
1.10	vespa.agent_libvirt module	7
1.11	vespa.controller module	8
1.12	vespa.ho module	8
1.13	vespa.ho_hy module	8
1.14	vespa.ho_ph module	9
1.15	vespa.ho_vm module	9
1.16	vespa.log_pipe module	10
1.17	vespa.model module	10
1.18	vespa.node module	11
1.19	vespa.starter module	12
1.20	vespa.view module	12
1.21	vespa.vo module	12
1.22	Module contents	13
<b>2</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



Contents:



## 1.1 Submodules

## 1.2 vespa.aes\_gcm module

```
class vespa.aes_gcm.AES_GCM(master_key)

    change_key(master_key)
    decrypt(init_value, ciphertext, auth_tag, auth_data='')
    encrypt(init_value, plaintext, auth_data='')

exception vespa.aes_gcm.InvalidInputException(msg)
    Bases: exceptions.Exception

exception vespa.aes_gcm.InvalidTagException
    Bases: exceptions.Exception

vespa.aes_gcm.gf_2_128_mul(x, y)
```

## 1.3 vespa.agent module

Agent representation

```
class vespa.agent.Agent(name, host, port, master, run=True)
    Bases: vespa.node.Node
```

## 1.4 vespa.agent\_av module

Agent representation

```
class vespa.agent_av.Agent_AV(name, host, port, master, vm)
    Bases: vespa.agent.Agent
```

Create an Agent able to communicate with the ClamAV backend (need a driver).

**Returns** The Agent instance to offer the ClamAV support

**Return type** Node

**connect\_warning** ()

Set up the agent for interactions with the VM

**dump\_analyzed\_file\_list** ()

Gather list of files analyzed bi the ClamAV antivirus

**Returns** The list of analyzed files

**Return type** list

**isolate\_warning** (*vm*)

Set up the agent for interactions with the hypervisor

**Parameters** **vm** (*str*) – The tuple (name, host, port) describing the backend

**send** (*msg*)

Overload the internal send to capture and send messages to the backend

**Parameters** **msg** (*str*) – The message to process and to send

**Returns** The backend response

**Return type** str

## 1.5 vespa.agent\_bandwidth module

Agent wrapper around /proc/dev/net to filter an interface statistics. The interface *eth0* is used as default.

**class** vespa.agent\_bandwidth.**Agent\_Bandwidth** (*name, host, port, master, run=True*)

Bases: vespa.agent.Agent

Provide a wrapper around Linux interfaces /proc files. The Agent can extract information of specific interfaces, i.e. eth0 or lo.

**Returns** The agent to grab informations

**Return type** Node

**get\_mac** ()

Grab the mac address of the class defined `_self.iface_`

**Returns** The string containing the mac address, colon separated

**Return type** str

**launch** ()

Send `_recv_bytes_` and `_trans_bytes_` back to the master every second

## 1.6 vespa.agent\_connections module

**Counting connections as suggested into:** <http://www.linuxjournal.com/content/back-dead-simple-bash-complex-ddos>

SynFlood <https://raw.githubusercontent.com/arthurnn/SynFlood/master/synflood>

**class** vespa.agent\_connections.**Agent\_Connections** (*name, host, port, master, run=True*)

Bases: vespa.agent.Agent

An agent gathering network links through psutil python module or system lsof command

**Returns** The wrapper



**Return type** Node

**launch** ()

Return network connections to orchestrator layer every second using either psutil or lsof

## 1.7 vespa.agent\_controller module

Agent to wrap Gandalf's controller

**class** `vespa.agent_controller.Agent_Controller` (*name, host, port, master, run=False*)

Bases: `vespa.agent.Agent`

Create an Agent to send a mac address to an OpenFlow controller

**Returns** The Agent instance to offer the OpenFlow alert\_ip function

**Return type** Node

**alert\_ip** (*ip, mac*)

Block the mac address on the network

**Parameters**

- **ip** (*str*) – The IP address or domain of the controller
- **mac** (*str*) – The mac address to block on the network

**Returns** The “Ok” string

**Return type** str

## 1.8 vespa.agent\_controller\_floodlight module

Agent to wrap Gandalf's controller. Based on floodlight, it can be a nice start for a full API against floodlight.

**class** `vespa.agent_controller_floodlight.Agent_Controller_Floodlight` (*name, host, port, master, run=False*)

Bases: `vespa.agent_controller.Agent_Controller`

Flag a mac address as suspicious and gather statistics for local links

**Returns** The wrapper to the OMN controller

**Return type** Node

**alert\_ip** (*ip, mac*)

Block a tuple (ip,mac) with SDN

**Parameters**

- **IP** (*str*) – The IP to block (for future)
- **mac** (*str*) – The associated MAC address (needed)

**Returns** The controller response

**Return type** str

**block\_hackers** (*mac*)

Block a MAC address with SDN

**Parameters** `mac` (*str*) – The associated MAC address

**Returns** The controller response

**Return type** `str`

**get\_link\_stats** (*cmd*='wm/topology/links/json')

Get links statistics over the floodlight controller

**Parameters** `cmd` (*str*) – The floodlight URL to grab the links statistics

**Returns** The controller response

**Return type** `str`

**get\_topology** (*cmd*='wm/topology/switchclusters/json')

Get the current topology of the SDN network

**Parameters** `cmd` (*str*) – The floodlight URL to grab the topology

**Returns** The list of nodes and links detected

**Return type** `dict`

**release\_hackers** ()

Release all tuples (ip,mac) with SDN

**Returns** The controller response

**Return type** `str`

**status\_hackers** ()

Get the status of a tuple (ip,mac) with SDN

**Returns** The controller response

**Return type** `str`

## 1.9 vespa.agent\_controller\_pox module

Agent to wrap the POX python SDN controller. It require some modification on the other side too. You can follow the mac address blocking tutorial on the POX website.

```
class vespa.agent_controller_pox.Agent_Controller_Pox(name, host, port, master,
                                                    run=False)
```

Bases: `vespa.agent_controller.Agent_Controller`

Flag a mac address as suspicious and gather statistics for local links

**Returns** The wrapper to the OMN controller

**Return type** `Node`

**alert\_ip** (*ip, mac*)

Block a tuple (ip,mac) with SDN

**Parameters**

- **IP** (*str*) – The IP to block (for future)
- **mac** (*str*) – The associated MAC address (needed)

**Returns** The controller response

**Return type** `str`

**block\_hackers** ()  
Block a MAC address with SDN

**Parameters** **mac** (*str*) – The associated MAC address

**Returns** The controller response

**Return type** *str*

**get\_link\_stats** (*cmd='get\_link\_stats'*)  
Get links statistics over the pox controller

**Parameters** **cmd** (*str*) – The POX URL to grab the links statistics

**Returns** The controller response

**Return type** *str*

**get\_topology** (*cmd='get\_topology'*)  
Get the current topology of the SDN network

**Parameters** **cmd** (*str*) – The POX URL to grab the topology

**Returns** The list of nodes and links detected

**Return type** *dict*

**release\_hackers** ()  
Release all tuples (ip,mac) with SDN

**Returns** The controller response

**Return type** *str*

**status\_hackers** ()  
Get the status of a tuple (ip,mac) with SDN

**Returns** The controller response

**Return type** *str*

## 1.10 vespa.agent\_libvirt module

**class** `vespa.agent_libvirt.Agent_Libvirt` (*name, host, port, master, run=True*)  
Bases: `vespa.agent.Agent`

**connect\_link** (*nodeName='arch-poc-win'*)

**contains\_vm** (*vm*)

**cut\_link** (*nodeName='arch-poc-win'*)

**launch** ()

**migrate** (*nodeName, quarantine, quarantine\_user*)

**restart** (*vm*)

**restart\_hard** (*vm*)

**send** (*msg*)

**send\_key** (*vm, args*)

## 1.11 vespa.controller module

Controller

```
class vespa.controller.Controller(model, view, testmode=False)
    Bases: object
```

```
    handler(signum, false)
```

```
    start()
```

```
class vespa.controller.HttpServer(name, host, port, handler, c)
```

```
    start()
```

```
    stop()
```

```
class vespa.controller.HttpServerHandler(request, client_address, server)
```

```
    Bases: BaseHTTPServer.BaseHTTPRequestHandler
```

```
    do_GET()
```

```
    log_message(format, *args)
```

```
class vespa.controller.MyHTTPServer(server_address, RequestHandlerClass, handler, control)
```

```
    Bases: BaseHTTPServer.HTTPServer
```

this class is necessary to allow passing custom request handler into the RequestHandlerClass

```
vespa.controller.server_handler(c, request)
```

## 1.12 vespa.ho module

Horizontal orchestrator

```
class vespa.ho.HO(name, host, port, master, run=True)
```

```
    Bases: vespa.node.Node
```

```
    findAgent(name)
```

## 1.13 vespa.ho\_hy module

Horizontal orchestrator

```
class vespa.ho_hy.HO_HY(name, host, port, master, run=True)
```

```
    Bases: vespa.ho.HO
```

Create an horizontal orchestrator to handle agents at the hypervisor level.

**Returns** The HO to gather and react on hypervisor agents.

**Return type** Node

```
ninjaMethod()
```

Empty function for tests

```
send(msg)
```

Overload the internal send() to capture and send messages to the backend

**Parameters** `msg` (*str*) – The message to process and to send

**Returns** The backend response

**Return type** `str`

## 1.14 vespa.ho\_ph module

Horizontal orchestrator

**class** `vespa.ho_ph.HO_PH` (*name, host, port, master, run=True*)

Bases: `vespa.ho.HO`

Create an horizontal orchestrator to handle agents at the physical level.

**Returns** The HO to gather and react on physical agents.

**Return type** `Node`

**ninjaMethod** ()

Empty function for tests

**send** (*msg*)

Overload the internal `send()` to capture and send messages to the backend

**Parameters** `msg` (*str*) – The message to process and to send

**Returns** The backend response

**Return type** `str`

## 1.15 vespa.ho\_vm module

Horizontal orchestrator

**class** `vespa.ho_vm.HO_VM` (*name, host, port, master, run=True*)

Bases: `vespa.ho.HO`

Create an horizontal orchestrator to handle agents at the VM level.

**Returns** The Horizontal Orchestrator to gather and react on VM agents.

**Return type** `Node`

**ninjaMethod** ()

Empty function for tests

**send** (*msg*)

Overload the internal `send()` to capture and send messages to the backend

**Parameters** `msg` (*str*) – The message to process and to send

**Returns** The backend response

**Return type** `str`

## 1.16 vespa.log\_pipe module

log\_pipe

**class** vespa.log\_pipe.bcolors

Bases: object

**ENDC** = '\x1b[0m'

**FAIL** = '\x1b[91m'

**HEADER** = '\x1b[95m'

**OKBLUE** = '\x1b[94m'

**OKGREEN** = '\x1b[92m'

**WARNING** = '\x1b[93m'

**disable** ()

vespa.log\_pipe.debug1 (*str*)

vespa.log\_pipe.debug2 (*str*)

vespa.log\_pipe.debug4 (*str*)

vespa.log\_pipe.debug5 (*str*)

vespa.log\_pipe.debug\_comm (*str*)

vespa.log\_pipe.debug\_comm\_detail (*str*)

Display extended communications information - How sendRemote split RECV\_LENGTH

Notes: offloaded to prevent screen flooding

vespa.log\_pipe.debug\_comm\_len (*s*)

Display maximum sized informations

Notes: offloaded to prevent screen flooding

vespa.log\_pipe.debug\_controller (*str*)

vespa.log\_pipe.debug\_crypto (*str*)

vespa.log\_pipe.debug\_info (*str*)

vespa.log\_pipe.debug\_init (*str*)

vespa.log\_pipe.debug\_thread (*str*)

## 1.17 vespa.model module

Model

**class** vespa.model.Model

Bases: vespa.node.Node

**create\_object\_instance** (*config, obj, master*)

**findNode** (*name*)

Return a tuple if the node “name” is found, raise an Exception otherwise. TODO: Refactor (3x)

**find\_vo** (*config*)  
Return VO object from config file

One and only one VO

**sendRemoteWake** (*remote, msg*)  
Force sending content to a remote host. Loop until it is done

## 1.18 vespa.node module

Most basic inherited class for a simple Node

Default config without backend

Pthread is used as Profiler wrapper

**If you do not need profiling you may replace the PThread class with:** `class Node(Thread):`

INTERNALS:

```
class vespa.node.Node (name, host, port, master, run=True)
    Bases: vespa.node.PThread

    run ()

class vespa.node.PThread (name, host, port, master, run=True)
    Bases: threading.Thread

    desc ()
        Return the tuple representing a node

    destroy ()
        Destroy all slaves

    findNode (name)

    get_backend ()
        Return the backend registered on initialization (i.e. Resource)

    launch ()

    list_slaves ()
        Return self.slaves

    listen_interface (host)

    register (name, host, port)

    register_alert_handler (handler)

    run ()
        Thread listening on node port. It creates a worker thread for each accepted socket.

        It SHOULD NOT accept multiple hosts, but ready for it.

    send (msg)
        Provide an entry to current node functions.

    sendAlert (msg)
        Wrapper for sendRemote with alert formatting.

        See "sendRemote" for arguments description and returns
```

**sendRemote** (*remote, msg, needack=True*)

Send a message to a node (*remote*) using the `node.desc()` string. This function deals with sockets directly.

Default behavior is to wait data as acknowledgement (*needack*). It is only modified for messages needing fast delivery and processing such as alerts.

**sendRemotef** (*remote, msg*)

Wrapper for `sendRemote` with `needack=False`

See “`sendRemote`” for arguments description and returns

**sendSocket** (*s, remote, msg*)

Handle the socket (`s.send`) message and encryption routines.

**wait\_backend** (*max\_tries=0*)

Ping node backend and return when backend is up.

WARNING: Does not timeout if `max_tries = 0`

**worker** (*conn*)

Handle socket reception job.

**class** `vespa.node.ThreadWorker` (*group=None, target=None, name=None, args=(), kwargs=None, verbose=None*)

Bases: `threading.Thread`

**run** ()

## 1.19 vespa.starter module

Starter

## 1.20 vespa.view module

**class** `vespa.view.View` (*model*)

Bases: `object`

## 1.21 vespa.vo module

Vertical orchestrator

**class** `vespa.vo.VO` (*name, host, port, master, run=True*)

Bases: `vespa.node.Node`

Create a Vertical Orchestrator to interconnect all other components. It may be requested by an external controller. All incoming communications go through the `alert()` method, and are parsed there.

**Returns** The VO instance

**Return type** `Node`

**alert** (*msg*)

This is the most important function of the whole framework. The format is `alert|source>...>source>message`. The message is also split in the following format `function#arg1#...#argN`. The two formats are used to create a Finite State Machine, each alert being a state transition.



**Parameters** `msg` (*str*) – The message to process with the current format

`get_alerts()`

`get_ip_connections()`

`get_link_stats()`

`get_next_recv_bytes()`

`get_next_trans_bytes()`

`get_recv_bytes()`

`get_topology()`

`get_trans_bytes()`

## 1.22 Module contents



---

## Indices and tables

---

- *genindex*
- *modindex*
- *search*



## V

vespa, 13  
vespa.aes\_gcm, 3  
vespa.agent, 3  
vespa.agent\_av, 3  
vespa.agent\_bandwidth, 4  
vespa.agent\_connections, 4  
vespa.agent\_controller, 5  
vespa.agent\_controller\_floodlight, 5  
vespa.agent\_controller\_pox, 6  
vespa.agent\_libvirt, 7  
vespa.controller, 8  
vespa.ho, 8  
vespa.ho\_hy, 8  
vespa.ho\_ph, 9  
vespa.ho\_vm, 9  
vespa.log\_pipe, 10  
vespa.model, 10  
vespa.node, 11  
vespa.starter, 12  
vespa.view, 12  
vespa.vo, 12



**A**

AES\_GCM (class in vespa.aes\_gcm), 3  
 Agent (class in vespa.agent), 3  
 Agent\_AV (class in vespa.agent\_av), 3  
 Agent\_Bandwidth (class in vespa.agent\_bandwidth), 4  
 Agent\_Connections (class in vespa.agent\_connections), 4  
 Agent\_Controller (class in vespa.agent\_controller), 5  
 Agent\_Controller\_Floodlight (class in vespa.agent\_controller\_floodlight), 5  
 Agent\_Controller\_Pox (class in vespa.agent\_controller\_pox), 6  
 Agent\_Libvirt (class in vespa.agent\_libvirt), 7  
 alert() (vespa.vo.VO method), 12  
 alert\_ip() (vespa.agent\_controller.Agent\_Controller method), 5  
 alert\_ip() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 5  
 alert\_ip() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 6

**B**

bcolors (class in vespa.log\_pipe), 10  
 block\_hackers() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 5  
 block\_hackers() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 6

**C**

change\_key() (vespa.aes\_gcm.AES\_GCM method), 3  
 connect\_link() (vespa.agent\_libvirt.Agent\_Libvirt method), 7  
 connect\_warning() (vespa.agent\_av.Agent\_AV method), 3  
 contains\_vm() (vespa.agent\_libvirt.Agent\_Libvirt method), 7  
 Controller (class in vespa.controller), 8  
 create\_object\_instance() (vespa.model.Model method), 10  
 cut\_link() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

**D**

debug1() (in module vespa.log\_pipe), 10  
 debug2() (in module vespa.log\_pipe), 10  
 debug4() (in module vespa.log\_pipe), 10  
 debug5() (in module vespa.log\_pipe), 10  
 debug\_comm() (in module vespa.log\_pipe), 10  
 debug\_comm\_detail() (in module vespa.log\_pipe), 10  
 debug\_comm\_len() (in module vespa.log\_pipe), 10  
 debug\_controller() (in module vespa.log\_pipe), 10  
 debug\_crypto() (in module vespa.log\_pipe), 10  
 debug\_info() (in module vespa.log\_pipe), 10  
 debug\_init() (in module vespa.log\_pipe), 10  
 debug\_thread() (in module vespa.log\_pipe), 10  
 decrypt() (vespa.aes\_gcm.AES\_GCM method), 3  
 desc() (vespa.node.PThread method), 11  
 desc() (vespa.node.PThread method), 11  
 disable() (vespa.log\_pipe.bcolors method), 10  
 do\_GET() (vespa.controller.HttpServerHandler method), 8  
 dump\_analyzed\_file\_list() (vespa.agent\_av.Agent\_AV method), 4

**E**

encrypt() (vespa.aes\_gcm.AES\_GCM method), 3  
 ENDC (vespa.log\_pipe.bcolors attribute), 10

**F**

FAIL (vespa.log\_pipe.bcolors attribute), 10  
 find\_vo() (vespa.model.Model method), 10  
 findAgent() (vespa.ho.HO method), 8  
 findNode() (vespa.model.Model method), 10  
 findNode() (vespa.node.PThread method), 11

**G**

get\_alerts() (vespa.vo.VO method), 13  
 get\_backend() (vespa.node.PThread method), 11  
 get\_ip\_connections() (vespa.vo.VO method), 13  
 get\_link\_stats() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 6

get\_link\_stats() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 7

get\_link\_stats() (vespa.vo.VO method), 13

get\_mac() (vespa.agent\_bandwidth.Agent\_Bandwidth method), 4

get\_next\_recv\_bytes() (vespa.vo.VO method), 13

get\_next\_trans\_bytes() (vespa.vo.VO method), 13

get\_recv\_bytes() (vespa.vo.VO method), 13

get\_topology() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 6

get\_topology() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 7

get\_topology() (vespa.vo.VO method), 13

get\_trans\_bytes() (vespa.vo.VO method), 13

gf\_2\_128\_mul() (in module vespa.aes\_gcm), 3

## H

handler() (vespa.controller.Controller method), 8

HEADER (vespa.log\_pipe.bcolors attribute), 10

HO (class in vespa.ho), 8

HO\_HY (class in vespa.ho\_hy), 8

HO\_PH (class in vespa.ho\_ph), 9

HO\_VM (class in vespa.ho\_vm), 9

HttpServer (class in vespa.controller), 8

HttpServerHandler (class in vespa.controller), 8

## I

InvalidInputException, 3

InvalidTagException, 3

isolate\_warning() (vespa.agent\_av.Agent\_AV method), 4

## L

launch() (vespa.agent\_bandwidth.Agent\_Bandwidth method), 4

launch() (vespa.agent\_connections.Agent\_Connections method), 5

launch() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

launch() (vespa.node.PThread method), 11

list\_slaves() (vespa.node.PThread method), 11

listen\_interface() (vespa.node.PThread method), 11

log\_message() (vespa.controller.HttpServerHandler method), 8

## M

migrate() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

Model (class in vespa.model), 10

MyHTTPServer (class in vespa.controller), 8

## N

ninjaMethod() (vespa.ho\_hy.HO\_HY method), 8

ninjaMethod() (vespa.ho\_ph.HO\_PH method), 9

ninjaMethod() (vespa.ho\_vm.HO\_VM method), 9

Node (class in vespa.node), 11

## O

OKBLUE (vespa.log\_pipe.bcolors attribute), 10

OKGREEN (vespa.log\_pipe.bcolors attribute), 10

## P

PThread (class in vespa.node), 11

## R

register() (vespa.node.PThread method), 11

register\_alert\_handler() (vespa.node.PThread method), 11

release\_hackers() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 6

release\_hackers() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 7

restart() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

restart\_hard() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

run() (vespa.node.Node method), 11

run() (vespa.node.PThread method), 11

run() (vespa.node.ThreadWorker method), 12

## S

send() (vespa.agent\_av.Agent\_AV method), 4

send() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

send() (vespa.ho\_hy.HO\_HY method), 8

send() (vespa.ho\_ph.HO\_PH method), 9

send() (vespa.ho\_vm.HO\_VM method), 9

send() (vespa.node.PThread method), 11

send\_key() (vespa.agent\_libvirt.Agent\_Libvirt method), 7

sendAlert() (vespa.node.PThread method), 11

sendRemote() (vespa.node.PThread method), 11

sendRemotef() (vespa.node.PThread method), 12

sendRemoteWake() (vespa.model.Model method), 11

sendSocket() (vespa.node.PThread method), 12

server\_handler() (in module vespa.controller), 8

start() (vespa.controller.Controller method), 8

start() (vespa.controller.HttpServer method), 8

status\_hackers() (vespa.agent\_controller\_floodlight.Agent\_Controller\_Floodlight method), 6

status\_hackers() (vespa.agent\_controller\_pox.Agent\_Controller\_Pox method), 7

stop() (vespa.controller.HttpServer method), 8

## T

ThreadWorker (class in vespa.node), 12

## V

vespa (module), 13

vespa.aes\_gcm (module), 3

vespa.agent (module), 3

vespa.agent\_av (module), 3

vespa.agent\_bandwidth (module), 4



vespa.agent\_connections (module), 4  
vespa.agent\_controller (module), 5  
vespa.agent\_controller\_floodlight (module), 5  
vespa.agent\_controller\_pox (module), 6  
vespa.agent\_libvirt (module), 7  
vespa.controller (module), 8  
vespa.ho (module), 8  
vespa.ho\_hy (module), 8  
vespa.ho\_ph (module), 9  
vespa.ho\_vm (module), 9  
vespa.log\_pipe (module), 10  
vespa.model (module), 10  
vespa.node (module), 11  
vespa.starter (module), 12  
vespa.view (module), 12  
vespa.vo (module), 12  
View (class in vespa.view), 12  
VO (class in vespa.vo), 12

## W

wait\_backend() (vespa.node.PThread method), 12  
WARNING (vespa.log\_pipe.bcolors attribute), 10  
worker() (vespa.node.PThread method), 12