1– Wazuh Components

Wazuh is a complete solution consisting of three (3) main components: the Wazuh Server, the Bungee Stack, and the Wazuh Agents.

1.1 – The wazuh server

The Wazuh server is based on a suite of applications where each application or component is designed to accomplish a certain task. These components work together to:

- Analyze the data received from the various logs,
- Trigger alerts when a log event matches a rule.
- Register new customers/agents, and
- Send data to the Elastic Stack server.

➢ Components:

- The **Wazuh Manager** receives and analyzes agent data using decoders and rules that have been created to trigger security alerts. The handler is also used to distribute configuration files to agents, monitor their status, and send control messages to trigger automatic actions at the agent level.
- The **registration service** uses a secure mechanism to register agents without any server-side intervention.
- The RESTful API provides an interface to manage and monitor manager and agent configuration. It is used to register agents, inspect manager log messages, decoders, and rules, and provide useful information about agents, including agent status, operating system details, and alerts related to file integrity monitoring and root checks.
- **Filebeat** is used to transfer alert data from the Wazuh Manager to Elasticsearch. This component has its own documentation developed by Elastic.

1.2 – Elastic battery

Elastic Stack is used to index, browse, and visualize Wazuh alert data. In addition, the Wazuh app for Kibana is also used to view configuration settings, rules, decoders, and agent status information. Dashboards used for this visualization include, but are not limited to, policy monitoring, compliance, and file integrity.

➢ Components:

- The **Wazuh** app is a Kibana plug-in designed to display information related to Wazuh by providing a RESTful API web interface. This interface makes the administration of the Wazuh Manager and Wazuh agents simple and powerful.
- **Elasticsearch** is a highly scalable, full-text search and analytics engine. It is used to index alert data and historical information about the status of agents. More information can be found in the official Elasticsearch documentation
- **Kibana** is a flexible and intuitive web-based interface for data exploration, analysis, and visualization. In combination with the Wazuh Kibana app, it is used as the Wazuh Web User Interface (WUI). More information can be found in Kibana's official documentation

1.3- Agents Wazuh

The Wazuh agent runs on monitored systems and is responsible for collecting log and event data, performing policy monitoring scans, detecting malware and rootkits, and triggering alerts when monitored files are modified. It communicates with the Wazuh server via an encrypted and authenticated channel.

➢ Components

- **Rootcheck** detects rootkits and malware on every system on which the agent is installed.
- **Log monitoring/analysis** collects and scans system logs for any suspicious activity.
- **Syscheck** runs periodically to check for changes to any configured file (or registry entry on Windows).

1.4 – Architecture of Wazuh

Wazuh has two main components to install: the Wazuh Manager and Elastic Stacker. Due to the type of installation, Wazuh allows two types of architecture: Centralized architecture: The Wazuh server and Elastic Stack run on the same server. Distributed architecture: Wazuh server and Elastic Stack cluster (one or more servers) on different systems.

In Figure 11 we can see the centralized network architecture in a single host and in Figure 12 we can see the distributed network architecture across multiple hosts:

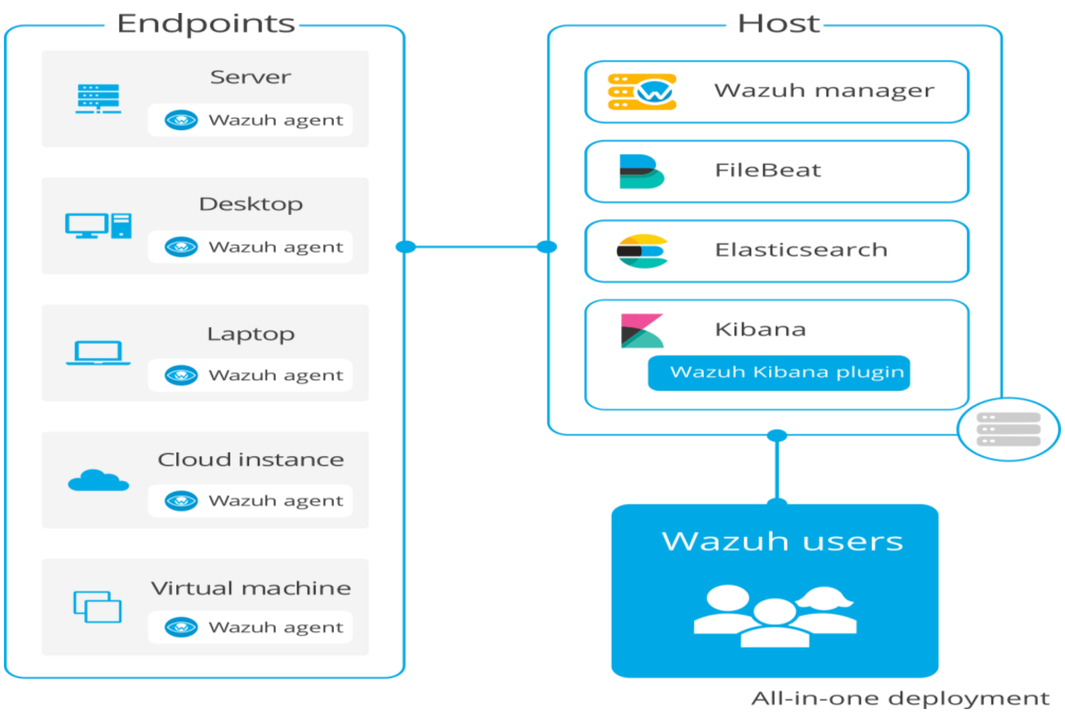

Figure 1: Wazuh's Centralized Network Architecture

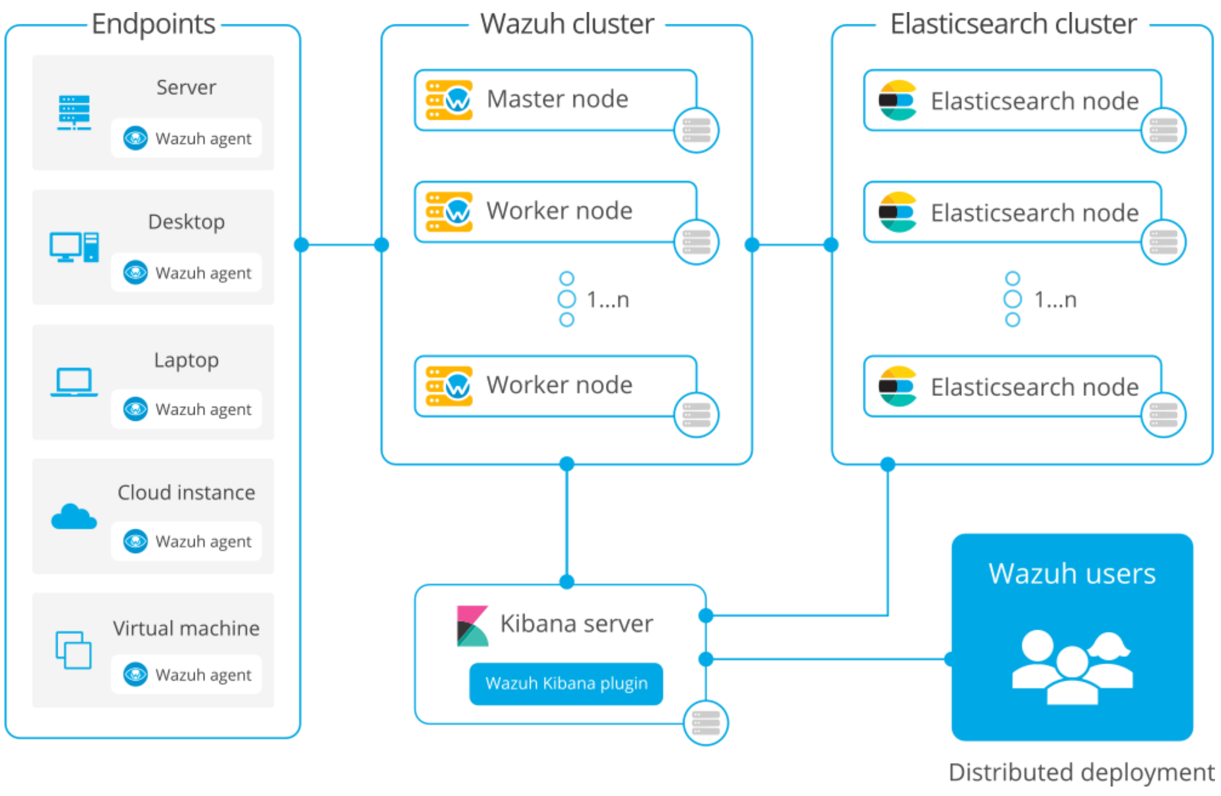

Figure 2: Wazuh's Distributed Network Architecture

1.5 – Scalability and availability: Wazuh cluster

A Wazuh cluster is a group of Wazuh managers who work together to improve the availability and scalability of the service. With a Wazuh cluster configuration, we have the potential to significantly increase the number of agents as long as we add worker nodes whenever needed.

1.6- Reasons for using a cluster

- Horizontal scalability:

It multiplies Wazuh's event processing capacity and allows it to have reports from thousands of agents. Adding a new node to the cluster is very simple (just add the master's address in the configuration) and it can be automated easily, giving the user the ability to implement autoscaling.

- High availability:

Servers eventually fail: the hardware can be broken, a human can turn them off, the system can crash... And while the server is restored, you won't be able to see what's going on in your agents. By using a cluster, you ensure that your agents will always have a manager to report to.

1.7- Types of nodes

- Master:

The master node centralizes and coordinates worker nodes, ensuring that critical and required data is consistent across all nodes. It ensures the centralization of the following elements:

- Agent Registration
- Suppression d’agent
- Configuring CDB Rules, Decoders, and Lists
- Configuring Agent Collections

<u>NB</u>:

- The master does not send his local configuration file to workers. If the configuration is changed in the master node, it must be manually changed in the compute nodes. Be careful not to overwrite the cluster section in each worker's local configuration.
- When rules, set-top boxes, or CDB lists are synchronized, worker nodes are not restarted. They must be manually restarted in order to apply the received configuration.
- All communications between nodes in the cluster are encrypted using the AES algorithm.

- Worker:
  Work nodes are responsible for 3 main tasks:

  - Synchronization of health files from the master node.
  - Sending updates to the agent's status to the master.
  - Redirect agent registration requests to the master.

1.8-Cluster operation

The cluster is managed by a daemon, called **wazuh-clusterd**, which communicates with all nodes following a master-worker architecture. The image below (Figure 13) shows the communications between a compute node and a master node. Each worker-master communication is independent of each other, since the workers are the ones who start the communication with the master.

There are different independent threads running, each framed in the image:

- **Keep alive thread**: Responsible for sending a keep alive to the master from time to time.
- **Agent Feed**: Responsible for sending agent statuses that pertain to this node.
- **Thread of Integrity**: Responsible for synchronizing files sent by the master.

All cluster logs are written to the logs/cluster.log file:

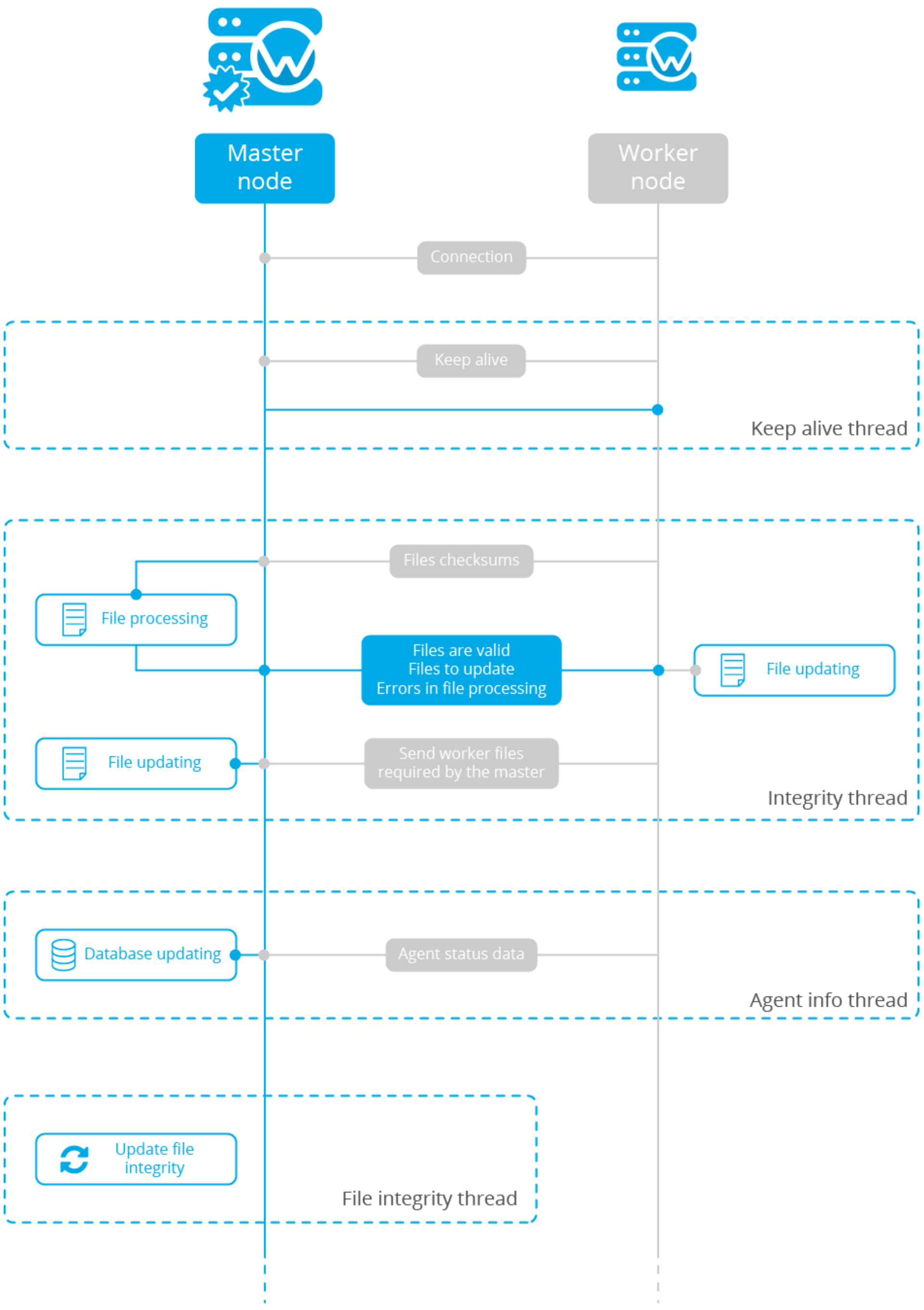

Figure 3: Description of how the Wazuh cluster works

- Keep the thread going:

The *keep alive thread* sends a keep-alive to the master from time to time. It is necessary to keep the connection open between the master and the worker, as the cluster uses persistent connections.

- Agent Feed:

The *agent information feed* sends information about the operating system, configured labels, and statuses of agents that report to the worker node.

The master also checks whether or not the agent exists before saving their status update. This is done to prevent the master from storing unnecessary information. For example, this situation is very common when an agent is deleted but the master has not yet notified the worker nodes.

- Thread of Integrity:

The *integrity thread* is responsible for synchronizing the files sent by the master node to the workers. These files are:

- Wazuh Agent Key File
- User-defined rules, set-top boxes, and CDB lists
- Agent Consolidates Files and Assignments

Usually, the master is responsible for sending the group assignments, but just in case a new agent starts reporting in a work node, the worker will send the new agent's group assignment to the master.

- File Integrity Queue:

  The integrity of each file is calculated using its MD5 checksum and modification time. To avoid compute health with each worker connection, health is computed in a different thread, called the File Health Thread, in the master node from time to time.

  In summary, however, Wazuh allows us to set up a cluster when we need to increase processing capacity because we have thousands of agents in the system that report to Wazuh's manager. In this case, there would be a central node (Master) and several worker nodes (workers) to which the agent load would be distributed. The architecture would be as shown in Figure 11, with a load balancer in front of the Wazuh cluster sending the agent report to either worker. It also gives us availability because if one worker breaks down, another can replace them. What this scheme wouldn't cover is a drop of the master (master) node, although that's a feature (having multiple masters and availability) that the Wazuh developers haven't implemented yet, as you can see in the GitHub thread. As for which architecture to use, seen in the architecture section of Wazuh, the decision of whether or not to implement a Wazuh cluster in our implementation

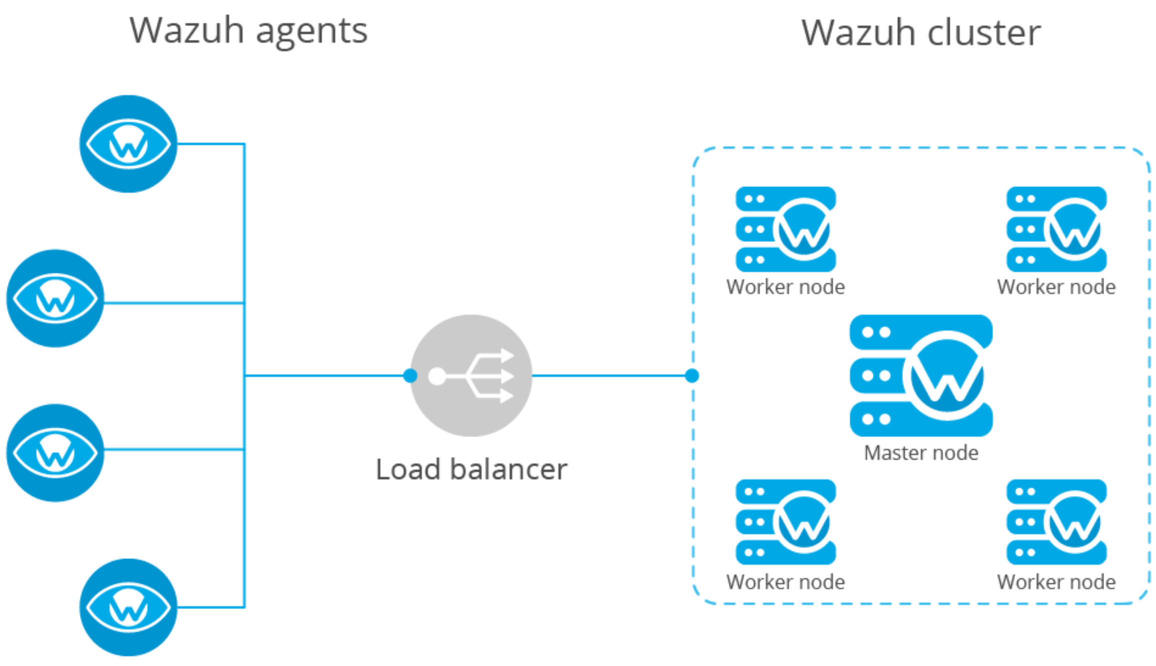

Figure 4: Wazuh's High Availability Architecture