SOC-In-A-Box Demo Docker Deployment guide

**Step 1: Set Up the Environment**

1. **Install Virtual Machines**: Set up individual VMs for each major component. Assign resources based on the role of each server:
   * **Elasticsearch & Kibana** (Core data storage and visualization)
   * **Logstash/Kafka** (Data processing)
   * **TheHive & Cortex** (Incident response and SOAR)
   * **Cuckoo Sandbox** (Malware analysis)
   * **Threat Simulation** (Install Infection Monkey, Atomic Red Team on separate testing VMs)
   * **Endpoints** (Windows/Linux for Filebeat, Winlogbeat, and Elastic Agent)
2. **Network Configuration**: Ensure VMs are on the same virtual network for easy communication.
3. **Install and Configure Open Source Tools**: On each VM, install the necessary components (e.g., Elasticsearch, Kibana, MISP, etc.) as per the requirements of each role. Download and install the open-source components for each layer.

**Step 2: Configure the Data Sources Layer**

1. **Install and Configure Beats**:
   * **Filebeat**: Configure Filebeat on all endpoints to send system and application logs to Logstash.
   * **Winlogbeat**: On Windows machines, configure Winlogbeat to forward Windows event logs.
   * **Auditbeat**: On Linux endpoints, set up Auditbeat for system auditing and file integrity monitoring.
   * **Metricbeat**: Install Metricbeat to monitor CPU, memory, and disk usage.
2. **Set Up Elastic Agent and Syslog**:
   * **Elastic Agent**: Deploy on each endpoint to capture telemetry data.
   * **Syslog Integration**: Configure network devices to send Syslog data to Logstash.
3. **Connect Threat Feeds**:
   * Install **MISP** and integrate it with OTX to ingest threat intelligence. Set up MISP to share IOCs with Elasticsearch.

**Step 3: Configure the Data Processing Layer**

1. **Install and Configure Logstash**:
   * Set up **Logstash** to parse and filter data from Beats, Elastic Agent, Syslog, and MISP. Configure it to enrich logs and send them to Elasticsearch.
   * Create separate pipelines in Logstash for each data type (e.g., Filebeat logs, Winlogbeat, Metricbeat, threat intelligence).
2. **Set Up Elasticsearch**:
   * Install **Elasticsearch** on the designated VM. Configure clusters and ensure appropriate sharding for performance.
   * Define index lifecycle policies to manage data retention.
3. **(Optional) Install Kafka**:
   * For large-scale demos, configure **Kafka** as an intermediary to stream data into Logstash.

**Step 4: Configure Incident Response and Automation Layer**

1. **Install and Set Up TheHive**:
   * Install **TheHive** on a dedicated VM, configure it to receive alerts from Elastic SIEM, and create case templates for APT incidents.
   * Set up integrations with Elasticsearch and MISP to enrich incidents.
2. **Configure Cortex XSOAR**:
   * Install **Cortex XSOAR** alongside TheHive or as a connected instance.
   * Create automated playbooks in Cortex for common APT actions, such as account suspension, IOC blocking, and notification triggers.
3. **Configure Elastic SIEM**:
   * In **Kibana**, enable Elastic SIEM and set up detection rules specifically targeting known APT TTPs.
   * Create custom alerts and assign actions (e.g., send to TheHive) based on incident severity.

**Step 5: AI-Driven Analysis and Detection Layer**

1. **Enable Kibana AI and ML Models**:
   * In **Kibana**, enable machine learning jobs. Configure behavioral baselines using UEBA (User and Entity Behavior Analytics) to detect anomalous activities.
   * Set up anomaly detection models on critical metrics (e.g., login frequency, data transfer volumes).
2. **Create Custom Detection Rules**:
   * Use **SIGMA and YARA rules** in conjunction with Elastic Stack to detect patterns related to APT activities.
   * Upload rules in Kibana’s detection section to continuously monitor for APT IOCs and behaviors.
3. **(Optional) Integrate Generative AI**:
   * If using any generative AI models (locally or through APIs), ensure sensitive data is properly obfuscated or restricted.
   * Use it to simulate potential APT scenarios and analyze Elastic’s detection performance.

**Step 6: Threat Simulation and Malware Analysis**

1. **Install Infection Monkey and Atomic Red Team**:
   * Deploy **Infection Monkey** on a dedicated VM to test lateral movement capabilities. Connect it to your network and execute tests to simulate APT movements.
   * Deploy **Atomic Red Team** and run various tactics to validate detection capabilities in Elastic SIEM and TheHive.
2. **Set Up Cuckoo Sandbox**:
   * Install **Cuckoo Sandbox** on a dedicated VM for malware analysis. Configure it to analyze suspicious files and share the analysis results with Elasticsearch.

**Step 7: Threat Intelligence and Collaboration Layer**

1. **Install and Configure MISP**:
   * Set up **MISP** to collect threat intelligence and integrate with Elastic Stack to share IOCs with **Elasticsearch**.
   * Add MISP as an enrichment source in **TheHive** to auto-enrich cases with threat intelligence data.
2. **Set Up Internal and External Threat Exchange**:
   * Connect MISP to trusted threat exchanges. Automate IOC sharing with internal systems for real-time threat intelligence updates.

**Step 8: Monitoring and Visualization Layer**

1. **Create Kibana Dashboards**:
   * Set up custom **Kibana dashboards** for each data source, visualizing endpoint activity, SIEM alerts, and threat intelligence. Ensure clear visual indicators for APT-specific activities.
   * Create specific dashboards for incident response tracking in TheHive, including case statuses and critical incidents.
2. **Enable and Customize Elastic SIEM**:
   * Configure Elastic SIEM’s correlation rules to monitor multi-source data. Enable visualizations for each alert type, showing a timeline of APT-like behavior.
3. **Monitor System Health and Performance**:
   * Use **Metricbeat** and **Elasticsearch** monitoring dashboards to track the health of all SOC components, ensuring optimal performance and quick troubleshooting.

**Step 9: Test and Validate the Setup**

1. **Run Simulations**:
   * Use Infection Monkey and Atomic Red Team to test detection capabilities. Validate alerts generated in Elastic SIEM and their subsequent handling in TheHive.
2. **Verify Threat Intelligence Flow**:
   * Add IOCs in MISP and ensure they propagate to Elastic SIEM and TheHive, validating that threat intelligence enrichment is functioning.
3. **Create Test Cases**:
   * Simulate common APT scenarios to verify detection, analysis, and response workflows. Ensure Cortex XSOAR’s playbooks execute as expected based on alert severity.

Creating an all-in-one Docker solution for your open-source SOC stack on Ubuntu is an efficient way to deploy and manage each component. Below is an installation script that will pull Docker images for each tool, create necessary networks, and configure each container for inter-communication. This script assumes you’re running Ubuntu 20.04 or later and have Docker and Docker Compose installed.

**All-in-One Docker Installation Script for SOC Stack**

#!/bin/bash

# SOC-In-A-Box Introduction display

echo "This is an AI Driven SOC-In-A-Box"

echo "All-In-One-Solution for Security Operation Centers"

echo "Powered by AI and Open Source technologies over container based platform"

# Check if the user is root

if [[ $EUID -ne 0 ]]; then

echo "This script must be run as root. Please run with sudo."

exit 1

fi

# Function to check command success

check\_success() {

if [ $? -ne 0 ]; then

echo "Error: $1 failed. Exiting."

exit 1

fi

}

# Prompt user for required information

read -p "Enter the IP address of the system: " IP\_ADDRESS

read -p "Enter a username for the services: " USERNAME

read -sp "Enter a password for the services: " PASSWORD

echo

echo "Starting the SOC stack installation process..."

# Display progress for the user

echo "Updating system packages..."

sudo apt-get update -y && sudo apt-get upgrade -y

check\_success "System package update and upgrade"

echo "Installing Docker and Docker Compose..."

sudo apt-get install -y docker.io docker-compose

sudo systemctl start docker

sudo systemctl enable docker

check\_success "Docker service start and enable"

# Create a Docker network, if not already created

if ! docker network inspect soc-network >/dev/null 2>&1; then

echo "Creating a Docker network for SOC components..."

docker network create soc-network

check\_success "Docker network creation"

else

echo "Docker network 'soc-network' already exists. Skipping creation."

fi

# Elasticsearch Java options for production (recommended: 2GB minimum heap size)

ES\_JAVA\_OPTS="-Xms2g -Xmx2g" # Set to 2GB. Adjust based on available system memory.

# Start deploying the containers

echo "Pulling and configuring SOC stack containers..."

# 1. Elasticsearch and Kibana (Elastic Stack)

docker pull docker.elastic.co/elasticsearch/elasticsearch:7.17.10

check\_success "Elasticsearch image pull"

docker pull docker.elastic.co/kibana/kibana:7.17.10

check\_success "Kibana image pull"

# Elasticsearch Container

echo "Starting Elasticsearch container..."

docker run -d --restart unless-stopped --name elasticsearch --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "discovery.type=single-node" \

-e "ES\_JAVA\_OPTS=-Xms2g -Xmx2g" \

-e "xpack.security.enabled=true" \

-e "xpack.security.authc.api\_key.enabled=true" \

-e "ELASTIC\_USERNAME=$USERNAME" \

-e "ELASTIC\_PASSWORD=$PASSWORD" \

-p 9200:9200 \

docker.elastic.co/elasticsearch/elasticsearch:7.17.10

check\_success "Elasticsearch container start"

# Kibana Container

echo "Starting Kibana container..."

docker run -d --restart unless-stopped --name kibana --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "ELASTICSEARCH\_HOSTS=http://elasticsearch:9200" \

-e "ELASTICSEARCH\_USERNAME=$USERNAME" \

-e "ELASTICSEARCH\_PASSWORD=$PASSWORD" \

-p 5601:5601 \

docker.elastic.co/kibana/kibana:7.17.10

check\_success "Kibana container start"

# 2. Logstash

docker pull docker.elastic.co/logstash/logstash:7.17.10

echo "Starting Logstash container..."

docker run -d --restart unless-stopped --name logstash --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-v /var/lib/docker\_volumes/logstash:/usr/share/logstash/data \

-p 5044:5044 -p 9600:9600 \

docker.elastic.co/logstash/logstash:7.17.10

check\_success "Logstash container start"

# 3. TheHive (Incident Response and SOAR)

docker pull thehiveproject/thehive:4.1.11-1

echo "Starting TheHive container..."

docker run -d --restart unless-stopped --name thehive --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "CORTEX\_URL=http://cortex:9000" \

-e "HIVE\_ADMIN\_LOGIN=$USERNAME" \

-e "HIVE\_ADMIN\_PASSWORD=$PASSWORD" \

-p 9001:9001 \

thehiveproject/thehive:4.1.11-1

check\_success "TheHive container start"

# 4. Cortex

docker pull thehiveproject/cortex:3.1.1-1

echo "Starting Cortex container..."

docker run -d --restart unless-stopped --name cortex --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "CORTEX\_ADMIN\_LOGIN=$USERNAME" \

-e "CORTEX\_ADMIN\_PASSWORD=$PASSWORD" \

-p 9000:9000 \

thehiveproject/cortex:3.1.1-1

check\_success "Cortex container start"

# 5. MISP

docker pull harvarditsecurity/misp

echo "Starting MISP container..."

docker run -d --restart unless-stopped --name misp --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "MYSQL\_USER=$USERNAME" \

-e "MYSQL\_PASSWORD=$PASSWORD" \

-e "MISP\_BASEURL=http://$IP\_ADDRESS:8080" \

-p 8080:80 \

harvarditsecurity/misp

check\_success "MISP container start"

# 6. Infection Monkey

docker pull guardicore/monkey

echo "Starting Infection Monkey container..."

docker run -d --restart unless-stopped --name infection-monkey --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-p 5000:5000 \

guardicore/monkey

check\_success "Infection Monkey container start"

# 7. OpenVAS

docker pull mikesplain/openvas

echo "Starting OpenVAS container..."

docker run -d --restart unless-stopped --name openvas --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "PUBLIC\_HOSTNAME=$IP\_ADDRESS" \

-p 443:443 \

mikesplain/openvas

check\_success "OpenVAS container start"

# 8. Cuckoo Sandbox

docker pull blacktop/cuckoo

echo "Starting Cuckoo Sandbox container..."

docker run -d --restart unless-stopped --name cuckoo --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-p 8090:8090 \

blacktop/cuckoo

check\_success "Cuckoo Sandbox container start"

# 9. Elastic Agent

docker pull docker.elastic.co/beats/elastic-agent:7.17.10

echo "Starting Elastic Agent container..."

docker run -d --restart unless-stopped --name elastic-agent --network soc-network \

-v /var/lib/docker\_volumes/kibana:/usr/share/kibana/data \

-v /var/lib/docker\_volumes/elasticsearch:/usr/share/elasticsearch/data \

-e "FLEET\_ENROLL=1" \

-e "FLEET\_URL=http://kibana:5601" \

-p 8200:8200 \

docker.elastic.co/beats/elastic-agent:7.17.10

check\_success "Elastic Agent container start"

# Check the status of all containers

echo "Verifying the installation and container status..."

for service in elasticsearch kibana logstash thehive cortex misp infection-monkey openvas cuckoo elastic-agent; do

if [[ $(docker inspect -f '{{.State.Running}}' $service) != "true" ]]; then

echo "Warning: $service container did not start as expected."

fi

done

docker ps

# Save credentials and access detail in a file

echo "Saving service access details..."

{

echo "You can access each service using the following URLs:"

echo "Elasticsearch: http://$IP\_ADDRESS:9200"

echo "Kibana: http://$IP\_ADDRESS:5601"

echo "TheHive: http://$IP\_ADDRESS:9001"

echo "Cortex: http://$IP\_ADDRESS:9000"

echo "MISP: http://$IP\_ADDRESS:8080"

echo "Infection Monkey: http://$IP\_ADDRESS:5000"

echo "OpenVAS: https://$IP\_ADDRESS"

echo "Cuckoo Sandbox: http://$IP\_ADDRESS:8090"

echo "Log in with username: $USERNAME and the $PASSWORD you provided."

} > SOC-Box-Credentials.txt

check\_success "Credential file save"

# Display completion message with access URLs

echo -e "\nSOC Stack installation completed successfully!"

echo -e "You can access each service using the following URLs:"

echo -e "Elasticsearch: http://$IP\_ADDRESS:9200"

echo -e "Kibana: http://$IP\_ADDRESS:5601"

echo -e "TheHive: http://$IP\_ADDRESS:9001"

echo -e "Cortex: http://$IP\_ADDRESS:9000"

echo -e "MISP: http://$IP\_ADDRESS:8080"

echo -e "Infection Monkey: http://$IP\_ADDRESS:5000"

echo -e "OpenVAS: https://$IP\_ADDRESS"

echo -e "Cuckoo Sandbox: http://$IP\_ADDRESS:8090"

echo -e "Log in with username: $USERNAME and the password you provided."

echo -e "\nAccess URLs and credentials are saved in SOC-Box-Credentials.txt on home path.\n"

Save the file with name SOC-In-A-Box\_Demo\_Docker\_Deployment\_Script.sh

Install dos2unix in ubuntu

Convert by using "dos2unix SOC-In-A-Box\_Demo\_Docker\_Deployment\_Script.sh"

Now, run the script with root privilege.

**Explanation of Additions**

1. **Root Privilege Check**: Ensures the script is only run as root.
2. **User Prompts**: The script prompts for the IP address, username, and password, making it configurable for any network.
3. **User Feedback**: Each step now provides feedback so users can track progress.
4. **Environment Variables for Configurations**: Each service uses the provided IP, username, and password to allow for a seamless single-credential login.
5. **Success Message with URLs**: A final message gives access URLs for all services with the specified IP address and user credentials.

**Important Notes**

* **Elastic Agent and Fleet Configuration**: The Elastic Agent requires a Fleet server. Ensure that Kibana’s Fleet server is properly configured to allow Elastic Agent management.
* **OpenVAS SSL Warnings**: OpenVAS uses HTTPS on port 443. It may give SSL warnings because it uses self-signed certificates.
* **Security**: Consider configuring firewall rules to allow access to specific services only from trusted IPs, especially for production environments.

This script can now serve as a robust all-in-one installation guide for your SOC stack.