Infrastructure deployment and testing

Release 1.0

Skander Lahbaiel

Jul 24, 2024

CONTENTS:

CHAPTER

ONE

TESTS

1.1 tests package

1.1.1 Subpackages

tests.check_failures package

Submodules

tests.check_failures.detect module

tests.check_failures.detect_cluster_cis module

This script parses the kube-bench results summary file to determine if there are any failed checks after system hardening. It reads the summary from the specified file and prints the number of failed checks.

The process involves: 1. Reading the kube-bench results directory from the environment variable. 2. Opening and reading the summary file. 3. Checking for failed checks in the summary. 4. Printing the number of failed checks.

Usage:

Ensure the *KUBE_BENCH_RESULTS* environment variable is set to the directory containing the kube-bench results file.

Functions:

- parse_txt_summary(file_path): Parses the kube-bench results summary file and prints the number of failed checks.

Examples:

To parse the summary file and print the number of failed checks, execute:

```
import os

# Retrieve the kube-bench results directory from environment variable
kube_bench_results_directory = os.getenv('KUBE_BENCH_RESULTS')

def parse_txt_summary(file_path):
    failed_checks = 0
    with open(file_path, 'r') as file:
        for line in file:
            if line.startswith("== Summary total =="):
                for _ in range(4):  # Read the next 4 lines for summary details
```

(continues on next page)

(continued from previous page)

```
                    summary_line = next(file).strip()
                    if summary_line.startswith("0 checks FAIL"):
                        failed_checks = int(summary_line.split()[0])
                        print(f"Failed checks: {failed_checks}")
                        break
                    else:
                        print("Failed checks: 1 or more")
                        failed_checks = 1
                        break
    return failed_checks == 0

# Ensure the file path is correctly set based on your directory structure
file_path = os.path.join(kube_bench_results_directory, "kube-bench-results_after_
↪hardening_20240521_033910.report")
parse_txt_summary(file_path)
```

Details:

- The *kube_bench_results_directory* is retrieved from the *KUBE_BENCH_RESULTS* environment variable.
- The *parse_txt_summary(file_path)* function reads the specified file and checks the summary section for failed checks.
- If the summary indicates "0 checks FAIL", it prints the number of failed checks as 0.
- If any checks fail, it prints "Failed checks: 1 or more" and sets the failed_checks variable to 1.

Notes:

- Ensure the *KUBE_BENCH_RESULTS* environment variable is set correctly to point to the directory containing the kube-bench results file.
- The file path should be updated based on the actual name and location of the results file.

Warnings:

- Ensure the file path is correct and accessible before running the script to avoid errors.
- The script assumes the summary section starts with "== Summary total ==" and that the next 4 lines contain the relevant summary details.

`tests.check_failures.detect_cluster_cis.`**`parse_txt_summary`**(*file_path*)

Parses the kube-bench results summary file and prints the number of failed checks.

Args:
file_path (str): The path to the kube-bench results summary file.

Returns:
bool: True if no checks failed, False otherwise.

tests.check_failures.detect_vm_cis_failure module

This script parses an HTML file to extract the compliance score after system hardening. It reads the HTML results from a specified file and prints the compliance score.

The process involves: 1. Reading the VM CIS results directory from the environment variable. 2. Opening and reading the HTML file to parse the compliance score.

Usage:
Ensure the *LATEST_REPORT* environment variable is set to the directory containing the HTML results file.

Functions:

- parse_html(file_path): Parses an HTML file to extract the compliance score.

Examples:
To parse the HTML file and print the compliance score, execute:

```
import os

# Retrieve the VM CIS results directory from environment variable
vm_cis_results_directory = os.getenv('LATEST_REPORT')

# Parse HTML file
parse_html(vm_cis_results_directory)
```

Details:

- The *vm_cis_results_directory* is retrieved from the *LATEST_REPORT* environment variable.
- The *parse_html(file_path)* function reads the specified HTML file and extracts the compliance score.
- If the compliance score is 100.0, it prints the score and returns True.
- If the compliance score is less than 100.0, it prints the score and returns False.

Notes:

- Ensure the *LATEST_REPORT* environment variable is set correctly to point to the directory containing the HTML results file.
- The file path should be updated based on the actual name and location of the results file.

Warnings:

- Ensure the file path is correct and accessible before running the script to avoid errors.
- The script assumes the HTML file contains a 'td' element with the class 'text-center' for the compliance score.

`tests.check_failures.detect_vm_cis_failure.`**`parse_html`**`(`*`file_path`*`)`

Parses an HTML file to extract the compliance score.

Args:
file_path (str): The path to the HTML file.

Returns:
bool: True if the compliance score is 100.0, False otherwise.

Module contents

1.1.2 Submodules

1.1.3 tests.an_overview module

Overview

This project involves the deployment and testing of a MicroK8s-based Kubernetes infrastructure, along with automated testing using Sonobuoy and kube-bench for compliance and security assessments. The project is managed and automated through a GitLab repository with CI/CD pipelines.

Project Structure

- **Infrastructure Deployment**: Scripts for setting up MicroK8s, configuring Kubernetes clients, and deploying necessary tools like Sonobuoy and kube-bench.
- **Testing**: Scripts for running Sonobuoy and kube-bench tests to ensure the infrastructure meets security and compliance standards.
- **GitLab Repository**: Contains all the necessary scripts, configurations, and the CI/CD pipeline definition.

GitLab Repository

Repository URL: *git@192.168.1.16:developers/charmed-microk8s.git*

Key Components - **Infrastructure Scripts**: Bash scripts for setting up MicroK8s and running compliance tests. - **Configuration Files**: Kubernetes client configuration and benchmark configuration files. - **.gitlab-ci.yml**: Defines the CI/CD pipeline for automating the deployment and testing process.

GitLab CI/CD Workflow The *.gitlab-ci.yml* file defines a CI/CD pipeline with the following stages:

1. **Setup:**
    - Download and prepare Sonobuoy.
    - Configure Kubernetes client.
    - Cleanup existing namespaces if any.
2. **Pre-Hardening Tests:**
    - Run kube-bench to check initial compliance.
    - Run Sonobuoy to perform initial system tests.
3. **Deployment:**
    - Deploy the hardened MicroK8s infrastructure.
4. **Post-Hardening Tests:**
    - Run kube-bench and Sonobuoy again to verify compliance and functionality after hardening.

How to Run the Project

1. **Clone the Repository**:

```
git clone git@192.168.1.16:developers/charmed-microk8s.git
cd charmed-microk8s
```

2. **Setup the Environment:**
    - Ensure you have the necessary permissions and tools installed (e.g., *wget*, *tar*, *sudo*, *curl*).
    - Configure your Kubernetes client as per the provided scripts.
3. **Deploy the Infrastructure:**
    - Use the provided scripts to deploy MicroK8s and configure the Kubernetes client.
    - Follow the instructions in the *README.md* and the deployment scripts.
4. **Run the Tests:**
    - Execute the pre-hardening and post-hardening test scripts as defined in the CI/CD pipeline.
    - Review the test results stored in the specified directories.

How to Deploy the Infrastructure

1. **Initial Setup:**
    - Run the setup scripts to prepare the environment and configure the Kubernetes client.
    - Deploy MicroK8s and ensure it is running correctly.
2. **Pre-Hardening Compliance Tests:**
    - Execute the kube-bench and Sonobuoy pre-hardening test scripts to assess the initial compliance state.
3. **Apply Hardening:**
    - Follow the hardening guidelines and scripts to secure the MicroK8s infrastructure.
4. **Post-Hardening Compliance Tests:**
    - Re-run the kube-bench and Sonobuoy tests to verify that the infrastructure remains compliant and functional after hardening.

Role of .gitlab-ci.yml and Workflow

The *.gitlab-ci.yml* file defines the CI/CD pipeline that automates the entire process: - **Stages**: The pipeline is divided into stages for setup, pre-hardening tests, deployment, and post-hardening tests. - **Jobs**: Each stage contains jobs that execute the relevant scripts for downloading tools, configuring the environment, running tests, and deploying the infrastructure. - **Automation**: The pipeline ensures that each step is executed in the correct order, with results collected and stored for review.

By following the CI/CD pipeline, the project ensures a consistent and repeatable process for deploying and testing the Kubernetes infrastructure, improving reliability and compliance.

1.1.4 tests.audit_ubuntu module

This script generates a tailoring file and audits the system using the *usg* tool.

The process involves two main steps: 1. Generating the tailoring file. 2. Auditing the system with the generated tailoring file.

Usage:
Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *sudo usg generate-tailoring cis_level1_server tailor.xml*
- *sudo usg audit –tailoring-file tailor.xml*

Examples:
To generate the tailoring file and audit the system, run:

```
echo "Generating the tailoring file and auditing the system..."
sudo usg generate-tailoring cis_level1_server tailor.xml
sudo usg audit --tailoring-file tailor.xml
```

Details:

- The first command (*sudo usg generate-tailoring cis_level1_server tailor.xml*) creates a tailoring file named *tailor.xml* based on the *cis_level1_server* profile. This file customizes the security checks to be applied during the audit.
- The second command (*sudo usg audit –tailoring-file tailor.xml*) performs an audit of the system using the *tailor.xml* file. This command checks the system's compliance with the customized security profile defined in the tailoring file.

Notes:

- Ensure that the *usg* tool is installed and properly configured on your system.
- The commands must be run with *sudo* to have the necessary permissions to perform system-level auditing and modifications.
- The tailoring file (*tailor.xml*) is crucial for customizing the audit to specific security requirements. Ensure that it is generated correctly before running the audit.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.

1.1.5 tests.fix_ubuntu module

This script runs the CIS benchmark, attaches to the pro account, installs necessary tools, and fixes issues found by the CIS benchmark. It also checks if a system reboot is required and creates a flag file if necessary.

The process involves several steps: 1. Attaching to the pro account. 2. Installing necessary tools. 3. Enabling and running the CIS benchmark. 4. Fixing issues found by the CIS benchmark. 5. Checking if a reboot is required and creating a flag file.

Usage:
Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *sudo pro attach <PRO_ACCOUNT_KEY>*
- *sudo apt update -y*
- *sudo apt install ubuntu-advantage-tools -y*
- *sudo pro enable usg*
- *sudo apt install usg -y*
- *sudo usg fix cis_level1_server*
- Check if a reboot is required and create a flag file if necessary.

Examples:

To run the CIS benchmark, attach to the pro account, install necessary tools, fix issues, and check for reboot requirement, run:

```
echo "Running CIS benchmark, attaching to the pro account and installing the␣
↪necessary tools..."
PRO_ACCOUNT_KEY="C13XWRLiKrfxme6882eHH5JRVihC6n"  # Make it a variable
sudo pro attach $PRO_ACCOUNT_KEY
echo "Installing the necessary tools for the CIS benchmark..."
sudo apt update -y
sudo apt install ubuntu-advantage-tools -y
sudo pro enable usg
sudo apt install usg -y
echo "Fixing the issues found by the CIS benchmark..."
sudo usg fix cis_level1_server

# Check if a reboot is required and create the flag file
if [ -f /var/run/reboot-required ]; then
  echo "Reboot required. Creating reboot-required.flag"
  touch /home/ubuntu/reboot-required.flag
fi
```

Details:

- The first command (*sudo pro attach $PRO_ACCOUNT_KEY*) attaches the system to the pro account using the provided account key.
- The second and third commands (*sudo apt update -y* and *sudo apt install ubuntu-advantage-tools -y*) update the package list and install the *ubuntu-advantage-tools* package.
- The fourth command (*sudo pro enable usg*) enables the *usg* tool.
- The fifth command (*sudo apt install usg -y*) installs the *usg* tool.
- The sixth command (*sudo usg fix cis_level1_server*) fixes the issues found by the CIS benchmark using the *cis_level1_server* profile.
- The script then checks if a reboot is required by looking for the */var/run/reboot-required* file. If found, it creates a *reboot-required.flag* file in the */home/ubuntu/* directory.

Notes:

- Ensure that the *pro* and *usg* tools are installed and properly configured on your system.
- The commands must be run with *sudo* to have the necessary permissions to perform system-level modifications.

- The pro account key (*$PRO_ACCOUNT_KEY*) should be kept secure and not exposed in public scripts.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.
- Attaching to a pro account may have associated costs. Ensure you have the necessary subscriptions and understand the terms of service.
- The reboot flag file (*reboot-required.flag*) is created to signal that a reboot is needed. Ensure proper handling of this flag in your environment.

1.1.6 tests.run_microk8s_tests module

This script runs the CIS benchmark, attaches to the pro account, installs necessary tools, and fixes issues found by the CIS benchmark. It also checks if a system reboot is required and creates a flag file if necessary.

The process involves several steps: 1. Attaching to the pro account. 2. Installing necessary tools. 3. Enabling and running the CIS benchmark. 4. Fixing issues found by the CIS benchmark. 5. Checking if a reboot is required and creating a flag file.

Usage:
Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *sudo pro attach <PRO_ACCOUNT_KEY>*
- *sudo apt update -y*
- *sudo apt install ubuntu-advantage-tools -y*
- *sudo pro enable usg*
- *sudo apt install usg -y*
- *sudo usg fix cis_level1_server*
- Check if a reboot is required and create a flag file if necessary.

Examples:
To run the CIS benchmark, attach to the pro account, install necessary tools, fix issues, and check for reboot requirement, run:

```
echo "Running CIS benchmark, attaching to the pro account and installing the␣
↪necessary tools..."
PRO_ACCOUNT_KEY="C13XWRLiKrfxme6882eHH5JRVihC6n"  # Make it a variable
sudo pro attach $PRO_ACCOUNT_KEY
echo "Installing the necessary tools for the CIS benchmark..."
sudo apt update -y
sudo apt install ubuntu-advantage-tools -y
sudo pro enable usg
sudo apt install usg -y
echo "Fixing the issues found by the CIS benchmark..."
sudo usg fix cis_level1_server

# Check if a reboot is required and create the flag file
```

(continues on next page)

(continued from previous page)

```
if [ -f /var/run/reboot-required ]; then
  echo "Reboot required. Creating reboot-required.flag"
  touch /home/ubuntu/reboot-required.flag
fi
```

Details:

- The first command (*sudo pro attach $PRO_ACCOUNT_KEY*) attaches the system to the pro account using the provided account key.
- The second and third commands (*sudo apt update -y* and *sudo apt install ubuntu-advantage-tools -y*) update the package list and install the *ubuntu-advantage-tools* package.
- The fourth command (*sudo pro enable usg*) enables the *usg* tool.
- The fifth command (*sudo apt install usg -y*) installs the *usg* tool.
- The sixth command (*sudo usg fix cis_level1_server*) fixes the issues found by the CIS benchmark using the *cis_level1_server* profile.
- The script then checks if a reboot is required by looking for the */var/run/reboot-required* file. If found, it creates a *reboot-required.flag* file in the */home/ubuntu/* directory.

Notes:

- Ensure that the *pro* and *usg* tools are installed and properly configured on your system.
- The commands must be run with *sudo* to have the necessary permissions to perform system-level modifications.
- The pro account key (*$PRO_ACCOUNT_KEY*) should be kept secure and not exposed in public scripts.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.
- Attaching to a pro account may have associated costs. Ensure you have the necessary subscriptions and understand the terms of service.
- The reboot flag file (*reboot-required.flag*) is created to signal that a reboot is needed. Ensure proper handling of this flag in your environment.

1.1.7 tests.run_pre_audit module

This script automates the process of setting up and running kube-bench, a tool for checking the compliance of Kubernetes clusters with CIS Kubernetes benchmarks. It includes downloading and installing kube-bench, setting up configuration files, running a pre-audit test, and saving the results.

The process involves several steps: 1. Setting up the installation directory for kube-bench. 2. Downloading the kube-bench tarball. 3. Extracting the kube-bench binary. 4. Moving the binary to */usr/local/bin* and making it executable. 5. Downloading necessary configuration files. 6. Running kube-bench with the specified configuration, benchmark version, and checks. 7. Saving the results of the pre-audit tests to a file.

Usage:

Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *sudo mkdir -p ${INSTALL_DIR}*
- *sudo curl -L ${KUBE_BENCH_URL} -o ${INSTALL_DIR}/kube-bench.tar.gz*
- *sudo tar -xvf ${INSTALL_DIR}/kube-bench.tar.gz -C ${INSTALL_DIR}*
- *sudo mv ${INSTALL_DIR}/kube-bench /usr/local/bin/*
- *sudo chmod +x /usr/local/bin/kube-bench*
- Download each configuration file from *CONFIG_FILES* array.
- *sudo kube-bench run –benchmark ${BENCHMARK_VERSION} –config-dir ${INSTALL_DIR}/cfg –config ${INSTALL_DIR}/cfg/config.yaml –check ${CHECKS}*
- Save the report to a file.

Examples:

To set up and run kube-bench, execute the following commands:

```
#!/bin/bash

# Define variables
KUBE_BENCH_VERSION="0.6.13"
KUBE_BENCH_TAR="kube-bench_${KUBE_BENCH_VERSION}_linux_amd64.tar.gz"
KUBE_BENCH_URL="https://github.com/aquasecurity/kube-bench/releases/download/v$
↪{KUBE_BENCH_VERSION}/${KUBE_BENCH_TAR}"
INSTALL_DIR="/opt/kube-bench"
RESULT_FILE="pre-audit-results_kube-bench.report"
BENCHMARK_VERSION="cis-1.24"
CONFIG_URL_BASE="https://raw.githubusercontent.com/aquasecurity/kube-bench/main/cfg/
↪cis-1.24"
CONFIG_FILES=("config.yaml" "controlplane.yaml" "etcd.yaml" "master.yaml" "node.yaml
↪" "policies.yaml")
CHECKS="1.1.1,1.1.2,1.1.3,1.1.4,1.1.5,1.1.6,1.1.7,1.1.8,1.1.9,1.1.10,1.1.11,1.1.12,
↪1.1.13,1.1.14,1.1.15,1.1.16,1.1.17,1.1.18,1.1.19,1.1.20,1.1.21,1.2.1,1.2.2,1.2.3,
↪1.2.4,1.2.5,1.2.6,1.2.7,1.2.8,1.2.9,1.2.10,1.2.11,1.2.12,1.2.13,1.2.14,1.2.15,1.2.
↪16,1.2.17,1.2.18,1.2.19,1.2.20,1.2.21,1.2.22,1.2.23,1.2.24,1.2.25,1.2.26,1.2.27,1.
↪2.28,1.2.29,1.2.30,1.2.31,1.3.1,1.3.2,1.3.3,1.3.4,1.3.5,1.3.6,1.3.7,1.4.1,1.4.2,2.
↪1,2.2,2.3,2.4,2.5,2.6,2.7,3.1.1,3.2.1,3.2.2,4.1.1,4.1.2,4.1.3,4.1.4,4.1.5,4.1.6,4.
↪1.7,4.1.8,4.1.9,4.1.10,4.2.1,4.2.2,4.2.3,4.2.4,4.2.5,4.2.6,4.2.7,4.2.8,4.2.9,4.2.
↪10,4.2.11,4.2.12,4.2.13,5.1.1,5.1.2,5.1.3,5.1.4,5.1.5,5.1.6,5.1.7,5.1.8,5.2.1,5.2.
↪2,5.2.3,5.2.4,5.2.5,5.2.6,5.2.7,5.2.8,5.2.9,5.2.10,5.2.11,5.2.12,5.2.13,5.3.1,5.3.
↪2,5.4.1,5.4.2,5.5.1,5.7.1,5.7.2,5.7.3,5.7.4"

# Create directory for kube-bench
echo "Creating kube-bench install directory..."
sudo mkdir -p ${INSTALL_DIR}
sudo mkdir -p ${INSTALL_DIR}/cfg/cis-1.24

# Download kube-bench
echo "Downloading kube-bench..."
sudo curl -L ${KUBE_BENCH_URL} -o ${INSTALL_DIR}/kube-bench.tar.gz

# Extract kube-bench
echo "Extracting kube-bench..."
sudo tar -xvf ${INSTALL_DIR}/kube-bench.tar.gz -C ${INSTALL_DIR}
```

(continues on next page)

(continued from previous page)

```
# Move kube-bench binary to /usr/local/bin
echo "Moving kube-bench to /usr/local/bin..."
sudo mv ${INSTALL_DIR}/kube-bench /usr/local/bin/
sudo chmod +x /usr/local/bin/kube-bench

# Download configuration files
echo "Downloading kube-bench configuration files..."
for file in "${CONFIG_FILES[@]}"; do
    sudo curl -L ${CONFIG_URL_BASE}/${file} -o ${INSTALL_DIR}/cfg/cis-1.24/${file}
done

# Run kube-bench with specified configuration, benchmark version, and checks
echo "Running kube-bench pre-audit tests..."
sudo kube-bench run --benchmark ${BENCHMARK_VERSION} --config-dir ${INSTALL_DIR}/
↪cfg --config ${INSTALL_DIR}/cfg/config.yaml --check ${CHECKS}

# Save the report to a file
echo "Saving kube-bench report to ${RESULT_FILE}..."
sudo kube-bench run --benchmark ${BENCHMARK_VERSION} --config-dir ${INSTALL_DIR}/
↪cfg --config ${INSTALL_DIR}/cfg/config.yaml --check ${CHECKS} > ${RESULT_FILE}

echo "kube-bench pre-audit tests completed. Results are saved in ${RESULT_FILE}"
```

Details:

- The first step is to define variables for the script, such as *KUBE_BENCH_VERSION*, *KUBE_BENCH_TAR*, *KUBE_BENCH_URL*, *INSTALL_DIR*, *RESULT_FILE*, *BENCHMARK_VERSION*, *CONFIG_URL_BASE*, *CONFIG_FILES*, and *CHECKS*. These variables are used throughout the script to configure the kube-bench installation and execution.
- The *sudo mkdir -p ${INSTALL_DIR}* and *sudo mkdir -p ${INSTALL_DIR}/cfg/cis-1.24* commands create the necessary directories for installing kube-bench and its configuration files.
- The *sudo curl -L ${KUBE_BENCH_URL} -o ${INSTALL_DIR}/kube-bench.tar.gz* command downloads the kube-bench tarball from the specified URL.
- The *sudo tar -xvf ${INSTALL_DIR}/kube-bench.tar.gz -C ${INSTALL_DIR}* command extracts the downloaded tarball into the installation directory.
- The *sudo mv ${INSTALL_DIR}/kube-bench /usr/local/bin/* command moves the kube-bench binary to */usr/local/bin* to make it executable from any location. The *sudo chmod +x /usr/local/bin/kube-bench* command sets the necessary executable permissions on the binary.
- The script then downloads each configuration file listed in the *CONFIG_FILES* array using a loop and the *sudo curl -L ${CONFIG_URL_BASE}/${file} -o ${INSTALL_DIR}/cfg/cis-1.24/${file}* command.
- The *sudo kube-bench run –benchmark ${BENCHMARK_VERSION} –config-dir ${INSTALL_DIR}/cfg –config ${INSTALL_DIR}/cfg/config.yaml –check ${CHECKS}* command runs kube-bench with the specified benchmark version, configuration directory, configuration file, and checks.
- Finally, the *sudo kube-bench run –benchmark ${BENCHMARK_VERSION} –config-dir ${INSTALL_DIR}/cfg –config ${INSTALL_DIR}/cfg/config.yaml –check ${CHECKS} > ${RESULT_FILE}* command saves the results of the kube-bench pre-audit tests to the specified result file.

Notes:

- Ensure that the *curl* command is installed on your system to download files.

- The commands must be run with *sudo* to have the necessary permissions to perform system-level modifications and installations.
- The configuration files should match the version of the benchmark being used (*cis-1.24* in this case).
- The *CHECKS* variable lists specific CIS benchmark checks to be performed. Modify this list according to your requirements.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.
- The results of the kube-bench tests should be reviewed carefully to understand any compliance issues or vulnerabilities found.
- The *RESULT_FILE* should be secured appropriately as it contains sensitive audit information about your Kubernetes cluster.

1.1.8 tests.run_pre_sonobuoy_tests module

This script automates the process of downloading and preparing Sonobuoy, configuring the Kubernetes client, running Sonobuoy tests, and storing the results. The tests are run before the hardening of the system, hence the results are prefixed with "pre".

The process involves several steps: 1. Downloading and extracting Sonobuoy. 2. Configuring the Kubernetes client. 3. Setting the KUBECONFIG environment variable. 4. Cleaning up any existing Sonobuoy namespace. 5. Running Sonobuoy tests. 6. Storing the test results.

Usage:

Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *wget https://github.com/vmware-tanzu/sonobuoy/releases/download/v0.57.1/sonobuoy_0.57.1_linux_amd64.tar.gz*
- *sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/*
- *mkdir -p /home/ubuntu/.kube*
- *sudo cp /var/snap/microk8s/current/credentials/client.config /home/ubuntu/.kube/config*
- *sudo chown ubuntu:ubuntu /home/ubuntu/.kube/config*
- *chmod 644 /home/ubuntu/.kube/config*
- *export KUBECONFIG=/home/ubuntu/.kube/config*
- *sudo microk8s.kubectl delete namespace sonobuoy || true*
- *sonobuoy run –wait*
- *sonobuoy retrieve*
- *tar -xvf $results -C ~/sonobuoy_results_pre*

Examples:

To set up and run Sonobuoy, execute the following commands:

```
#!/bin/bash
# Ensure the script exits on any error
set -e

echo "Downloading and preparing Sonobuoy..."
wget https://github.com/vmware-tanzu/sonobuoy/releases/download/v0.57.1/sonobuoy_0.
↪57.1_linux_amd64.tar.gz
sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/

echo "Configuring Kubernetes client..."
mkdir -p /home/ubuntu/.kube
sudo cp /var/snap/microk8s/current/credentials/client.config /home/ubuntu/.kube/
↪config
sudo chown ubuntu:ubuntu /home/ubuntu/.kube/config
chmod 644 /home/ubuntu/.kube/config

export KUBECONFIG=/home/ubuntu/.kube/config
echo "KUBECONFIG set to: $KUBECONFIG"

echo "Cleaning up existing Sonobuoy namespace if exists..."
sudo microk8s.kubectl delete namespace sonobuoy || true

echo "Running Sonobuoy tests..."
sonobuoy run --wait
results=$(sonobuoy retrieve)
mkdir -p ~/sonobuoy_results_pre
tar -xvf $results -C ~/sonobuoy_results_pre

echo "Tests complete. Results stored in ~/sonobuoy_results_pre"
```

Details:

- The *set -e* command ensures the script exits immediately if any command exits with a non-zero status, which is useful for error handling.
- The *wget* command downloads the specified version of Sonobuoy from the official GitHub releases.
- The *sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/* command extracts the downloaded tarball into */usr/local/bin/* to make Sonobuoy executable from any location.
- The script then configures the Kubernetes client by creating the *.kube* directory in the user's home directory, copying the Kubernetes client configuration file, setting the appropriate ownership and permissions, and exporting the *KUBECONFIG* environment variable.
- The *sudo microk8s.kubectl delete namespace sonobuoy || true* command ensures any existing Sonobuoy namespace is deleted to avoid conflicts during the test run.
- The *sonobuoy run –wait* command runs the Sonobuoy tests and waits for their completion.
- The *sonobuoy retrieve* command retrieves the results of the Sonobuoy tests.
- The *tar -xvf $results -C ~/sonobuoy_results_pre* command extracts the retrieved results into the *~/sonobuoy_results_pre* directory for later review.

Notes:

- Ensure that the *wget* and *tar* commands are installed on your system to download and extract files.

- The commands must be run with *sudo* to have the necessary permissions to perform system-level modifications and installations.
- The results of the Sonobuoy tests are stored in a directory prefixed with "pre" to indicate they are taken before system hardening.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.
- The results of the Sonobuoy tests should be reviewed carefully to understand any compliance issues or vulnerabilities found.
- The *results* variable should be handled carefully as it contains sensitive audit information about your Kubernetes cluster.

1.1.9 tests.sonobuoy_tests module

This script automates the process of downloading and preparing Sonobuoy, configuring the Kubernetes client, running Sonobuoy tests, and storing the results. These tests are run after hardening the system.

The process involves several steps: 1. Downloading and extracting Sonobuoy. 2. Configuring the Kubernetes client. 3. Setting the KUBECONFIG environment variable. 4. Cleaning up any existing Sonobuoy namespace. 5. Running Sonobuoy tests. 6. Storing the test results.

Usage:
Run the script in a terminal or command line interface with the necessary permissions.

Commands:

- *wget https://github.com/vmware-tanzu/sonobuoy/releases/download/v0.57.1/sonobuoy_0.57.1_linux_amd64.tar.gz*
- *sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/*
- *mkdir -p /home/ubuntu/.kube*
- *sudo cp /var/snap/microk8s/current/credentials/client.config /home/ubuntu/.kube/config*
- *sudo chown ubuntu:ubuntu /home/ubuntu/.kube/config*
- *chmod 644 /home/ubuntu/.kube/config*
- *export KUBECONFIG=/home/ubuntu/.kube/config*
- *sudo microk8s.kubectl delete namespace sonobuoy || true*
- *sonobuoy run –wait*
- *sonobuoy retrieve*
- *tar -xvf $results -C ~/sonobuoy_results*

Examples:
To set up and run Sonobuoy after system hardening, execute the following commands:

```
#!/bin/bash
# Ensure the script exits on any error
set -e

echo "Downloading and preparing Sonobuoy..."
```

(continues on next page)

(continued from previous page)

```
wget https://github.com/vmware-tanzu/sonobuoy/releases/download/v0.57.1/sonobuoy_0.
↪57.1_linux_amd64.tar.gz  # Downloads Sonobuoy from GitHub.
sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/  # Extracts␣
↪Sonobuoy binary to the local bin directory for execution.

echo "Configuring Kubernetes client..."
mkdir -p /home/ubuntu/.kube  # Creates a directory for Kubernetes config files if␣
↪it doesn't exist.
sudo cp /var/snap/microk8s/current/credentials/client.config /home/ubuntu/.kube/
↪config  # Copies the MicroK8s client configuration to the standard Kubernetes␣
↪config location.
sudo chown ubuntu:ubuntu /home/ubuntu/.kube/config  # Changes ownership of the␣
↪config file to the ubuntu user.
chmod 644 /home/ubuntu/.kube/config  # Sets read and write permissions for the user,
↪ and read for others on the config file.

export KUBECONFIG=/home/ubuntu/.kube/config  # Sets the KUBECONFIG environment␣
↪variable to point to the config file.
echo "KUBECONFIG set to: $KUBECONFIG"  # Outputs the path to the KUBECONFIG.

echo "Cleaning up existing Sonobuoy namespace if exists..."
sudo microk8s.kubectl delete namespace sonobuoy || true

echo "Running Sonobuoy tests..."
sonobuoy run --wait  # Runs Sonobuoy tests and waits for them to complete.
results=$(sonobuoy retrieve)  # Retrieves the results from the Sonobuoy run.
mkdir -p ~/sonobuoy_results  # Creates a directory to store Sonobuoy results if it␣
↪doesn't exist.
tar -xvf $results -C ~/sonobuoy_results  # Extracts the results into the results␣
↪directory.

export SONOBUOY_RESULTS=~/sonobuoy_results

echo "Tests complete. Results stored in ~/sonobuoy_results"  # Notifies that tests␣
↪are complete and results are stored.
```

Details:

- The *set -e* command ensures the script exits immediately if any command exits with a non-zero status, which is useful for error handling.
- The *wget* command downloads the specified version of Sonobuoy from the official GitHub releases.
- The *sudo tar -xvf sonobuoy_0.57.1_linux_amd64.tar.gz -C /usr/local/bin/* command extracts the downloaded tarball into */usr/local/bin/* to make Sonobuoy executable from any location.
- The script then configures the Kubernetes client by creating the *.kube* directory in the user's home directory, copying the Kubernetes client configuration file, setting the appropriate ownership and permissions, and exporting the *KUBECONFIG* environment variable.
- The *sudo microk8s.kubectl delete namespace sonobuoy || true* command ensures any existing Sonobuoy namespace is deleted to avoid conflicts during the test run.
- The *sonobuoy run –wait* command runs the Sonobuoy tests and waits for their completion.
- The *sonobuoy retrieve* command retrieves the results of the Sonobuoy tests.

- The *tar -xvf $results -C ~/sonobuoy_results* command extracts the retrieved results into the *~/sonobuoy_results* directory for later review.
- The *export SONOBUOY_RESULTS=~/sonobuoy_results* command sets the environment variable to the results directory.

Notes:

- Ensure that the *wget* and *tar* commands are installed on your system to download and extract files.
- The commands must be run with *sudo* to have the necessary permissions to perform system-level modifications and installations.
- The results of the Sonobuoy tests are stored in a directory after hardening, indicating the system's compliance status post-hardening.

Warnings:

- Running these commands can modify system settings and configurations. Make sure you understand the implications of the changes being made.
- Always backup important data before performing system audits or modifications.
- The results of the Sonobuoy tests should be reviewed carefully to understand any compliance issues or vulnerabilities found.
- The *results* variable should be handled carefully as it contains sensitive audit information about your Kubernetes cluster.

1.1.10 tests.workflow_pipeline_gitlab module

CI/CD Pipeline Configuration Documentation

This documentation provides a detailed overview of the CI/CD pipeline configuration for deploying, securing, auditing, and testing a Kubernetes cluster using microk8s, ceph, metallb, nginx-ingress, cert-manager, and Sonobuoy.

Pipeline Stages

The pipeline is divided into the following stages:

1. **deploy**: Deploys microk8s and other necessary components.
2. **pre-audit-and-secure-os**: Runs pre-audit checks and secures the operating system.
3. **wait_for_reboot**: Reboots all machines to apply changes.
4. **audit-os**: Audits the operating system for compliance.
5. **harden-and-test-cluster**: Hardens the cluster and runs tests to validate the setup.

Stages and Variables

The variables used in this pipeline ensure consistent and reusable configuration across different stages. Each stage performs specific tasks as outlined below.

Deploy Stage

Variables:

MICROK8S_MODEL_NAME: Name of the microk8s model. MICROK8S_CLOUD_NAME: Name of the microk8s cloud. MICROK8S_CONFIG_FILE: Name of the microk8s configuration file. METALLB_MODEL_NAME: Name of the metallb model.

Steps:

- Add and deploy the microk8s model using the MAAS cloud configuration.
- Deploy the components listed in the bundle.yaml file.
- Wait for each application to become active or idle with a timeout of 300 minutes.
- Enable CephFS in ceph-csi.
- Add microk8s as a Juju cloud and configure k8s-config.
- Add and deploy the Metallb model.
- Deploy Nginx-Ingress.
- Deploy cert-manager and cluster-issuer.

Pre-Audit and Secure OS Stage

Variables:

MICROK8S_MODEL_NAME: Name of the microk8s model. MACHINE: Target machine for certain operations.

Steps:

- Switch to the microk8s model.
- Verify sudo access and Juju SSH.
- Get machine names and iterate through them.
- Verify sudo access on each machine.
- Copy and execute the secure script on each machine.

Wait for Reboot Stage

Steps:

- Get machine names.
- Initiate reboot on all machines.
- Wait for systems to come back (adjust wait time as needed).

Audit OS Stage

Variables:
MICROK8S_MODEL_NAME: Name of the microk8s model. MACHINE: Target machine for certain operations.

Steps:

- Switch to the microk8s model.
- Get machine names and iterate through them.
- Verify sudo access on each machine.
- Create directory for reports on each machine.
- Copy and execute the audit script on each machine.
- Copy audit reports to the local directory and check for failures.

Harden and Test Cluster Stage

Variables:
MICROK8S_MODEL_NAME: Name of the microk8s model. MACHINE: Target machine for certain operations.

Steps:

- Switch to the microk8s model.
- Verify sudo access and Juju SSH.
- Get machine names and iterate through them.
- Verify sudo access on each machine.
- Copy and execute the test script on each machine.
- Create directory for test results on GitLab Runner.
- Copy test results to the local directory and check for failures.
- Run Sonobuoy tests on the specified machine.
- Copy Sonobuoy test results to the local directory and check for failures.

Artifacts

Paths:

- *sonobuoy_results*: Directory containing Sonobuoy test results.
- *kube-bench-results*: Directory containing kube-bench test results.

Expiration:

- Artifacts will expire in 1 week (adjust as needed).

Detailed Pipeline Configuration

1.1.11 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX