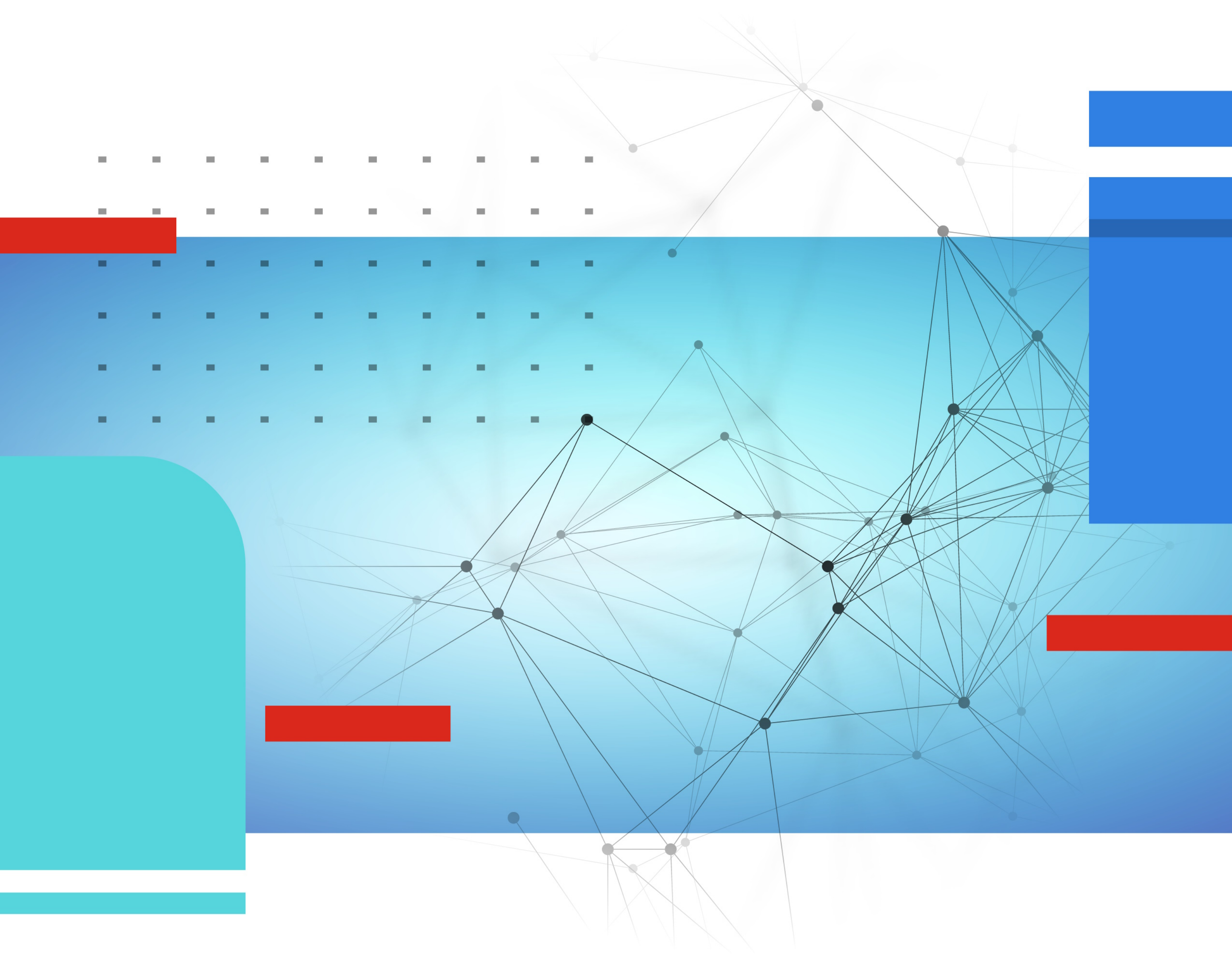
FORTINET

Upgrade Guide

FortiSOAR 7.6.0



FORTINET DOCUMENT LIBRARY
https://docs.fortinet.com

FORTINET VIDEO LIBRARY
https://video.fortinet.com

FORTINET BLOG
https://blog.fortinet.com

CUSTOMER SERVICE & SUPPORT
https://support.fortinet.com

FORTINET TRAINING & CERTIFICATION PROGRAM
https://www.fortinet.com/training-certification

FORTINET TRAINING INSTITUTE
https://training.fortinet.com

FORTIGUARD LABS
https://www.fortiguard.com

END USER LICENSE AGREEMENT
https://www.fortinet.com/doc/legal/EULA.pdf

FEEDBACK
Email: techdoc@fortinet.com

August, 2024
FortiSOAR 7.6.0 Upgrade Guide
00-400-000000-20210416

TABLE OF CONTENTS

Change Log

Date	Change Description
2025-03-26	Added information to unmount any external mount entries before upgrading FortiSOAR in the Preparing to Upgrade FortiSOAR chapter.
2025-02-03	Updated the Preparing to Upgrade FortiSOAR chapter to add a note about upgrading a FortiSOAR instance with an external PostgreSQL database.
2024-12-27	Updated the '*Upgrading a FortiSOAR Secure Message Exchange Cluster*' topic in the Upgrading a FortiSOAR Distributed Multi-Tenancy Configuration chapter.
2024-08-14	Updated the `csadm upgrade check-readiness` command in the Upgrading a FortiSOAR Enterprise instance using the Upgrade Framework chapter.
2024-08-01	Initial release of 7.6.0

Introduction

This guide covers upgrading a FortiSOAR™ enterprise instance, High Availability (HA) cluster, or a distributed multi-tenant configuration.

The FortiSOAR UI displays a notification when a new release (always the latest) is available. The notification also contains a link to that version's release notes so that you can get details about the latest available release. This keeps FortiSOAR users informed about the latest releases and then users can make informed decisions about upgrading to the latest available FortiSOAR version.

This document describes how to upgrade FortiSOAR to 7.6.0. This guide is intended to supplement the FortiSOAR Release Notes, and it includes the following sections:

- Preparing to Upgrade FortiSOAR
- Upgrading a FortiSOAR Enterprise Instance using Upgrade Framework
  An "Upgrade Framework" was introduced in release 7.5.0 to improve the flexibility, usability, and efficiency of the FortiSOAR upgrade process.
- Upgrading a FortiSOAR High Availability Cluster
- Upgrading a FortiSOAR Distributed Multi-Tenancy Configuration
- Upgrading FortiSOAR using the Offline Repository
- Upgrading your FortiSOAR Docker image and upgrading your FortiSOAR Docker on an Amazon Elastic Kubernetes Cluster

You can upgrade your FortiSOAR enterprise instance, High Availability (HA) cluster, or a distributed multi-tenant instance to release 7.6.0 from release 7.5.0. Also, once you have upgraded your instance, you must log out from the FortiSOAR UI and log back into FortiSOAR.

Also, note that the upgrade procedure temporarily takes the FortiSOAR application offline while the upgrade operations are taking place. We recommend that you send a prior notification to all users of a scheduled upgrade as users are unable to log in to the FortiSOAR Platform during the upgrade. Release 7.6.0 optimizes the upgrade process to minimize application downtime and provide a faster and more reliable upgrade experience. For more information, see the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework chapter.

Before you upgrade your FortiSOAR instance, it is highly recommended that you review the *Special Notices* chapter in the "Release Notes", so that you are aware of operational and breaking changes made in version 7.6.0.

For information on upgrading FortiSOAR using the offline repository and upgrading your FortiSOAR Docker image, see the "Deployment Guide."

To solve common issues that occur during the upgrade process, see the *Troubleshooting FortiSOAR* chapter in the “Deployment Guide.”

Preparing to Upgrade FortiSOAR

We recommend performing the following tasks to prepare for a successful FortiSOAR upgrade:

If you are upgrading a FortiSOAR instance with an external PostgreSQL database to FortiSOAR 7.6.0 or later, first upgrade PostgreSQL to version 16 or later. After upgrading PostgreSQL, proceed with the FortiSOAR upgrade. Upgrading to PostgreSQL 16 or later is required because FortiSOAR 7.6.0 and later use the 'pg_squeeze' and 'pg_repack' utilities—available only in PostgreSQL 16 and later—to optimize disk space reclamation. For more information, see the Externalization of your FortiSOAR PostgreSQL database chapter in the "Administration Guide."

To prepare for upgrading FortiSOAR (summary):

- Ensure that all data ingestion playbooks and schedules are stopped and wait for all existing active playbooks to complete before starting the upgrade process.
- Take a VM snapshot of your current system. Only after you have taken a VM snapshot of your system should you attempt to upgrade FortiSOAR. In case of any failures, these VM snapshots will allow you to revert to the latest working state. Follow the steps mentioned in the documentation of your platform for taking a snapshot and reverting to the current snapshot.
- Take a backup of your FortiSOAR Built-in connectors' (SSH, IMAP, Database, Utilities, etc.) configuration, since the configuration of your FortiSOAR Built-in connectors might be reset, if there are changes to the configuration parameters across versions.
- Ensure that repo.fortisoar.fortinet.com is reachable from your VM. If you are connecting using a proxy, then ensure that proxy details set are correct using the `csadm network list-proxy` command and also ensure that `repo.fortisoar.fortinet.com` is allowed in your proxy. For more information on `csadm` CLI, see the *FortiSOAR Admin CLI* chapter in the "Administration Guide."
- Ensure that there are no duplicate entries in the `/etc/fstab` file.
- Unmount any external mount entries listed in the `/etc/fstab` file before upgrading FortiSOAR. After the upgrade is complete, you can re-mount these entries.
- Ensure that all pending changes for modules are published before starting the upgrade process.
- Disable the IPv6 protocol from your VM where you are upgrading FortiSOAR if you are not using the IPv6 protocol.
- Ensure that you have reviewed the *Special Notices* chapter in the "Release Notes", so that you are aware of operational and breaking changes made in version 7.6.0.

Upgrading a FortiSOAR Enterprise instance using the Upgrade Framework

The "Upgrade Framework" is designed to improve the flexibility, usability, and efficiency of the FortiSOAR upgrade process. More importantly, it validates the feasibility of the upgrade before proceeding with the actual upgrade. The Upgrade Framework improves the upgrade experience by offering users the ability to customize the pre- or post-upgrade phases.

The Upgrade Framework aims to address this issue and provide the following benefits:

- **Flexibility**: A modular architecture that enables users to easily plug in their requirements as new tasks in a particular phase, allowing users to customize the upgrade process as per their needs, fostering a more personalized and adaptable user experience.
- **Control**: The ability to selectively execute phases and tasks, providing greater control over the upgrade process. Users can execute individual phases independently, allowing for focused testing, validation, and troubleshooting without requiring an entire upgrade cycle.
- **Efficient troubleshooting and proactive issue identification**: Granular phase execution facilitates targeted troubleshooting, including error logging for each task, leading to faster issue resolution. Furthermore, the ability to initiate pre-validation processes independently ensures that potential issues are identified and resolved proactively before proceeding with the entire upgrade. This contributes to a more streamlined and error-resistant upgrade process.
- **Enhanced state management and resilience**: Failures can be gracefully handled, ensuring a more robust upgrade process. Also, the upgraded framework introduces mechanisms for improved state management, which enable upgrade operations to be restarted from the point of failure, thus minimizing disruptions and optimizing the time taken for upgrades.

In release 7.6.0, the upgrade framework has been enhanced to improve resilience by separating activities (post-upgrade activities) that need to be performed after the code base upgrade, such as database migrations and other services. This separation allows for the code base of all packages to be upgraded first, followed by post-upgrade activities. If a failure occurs, the upgrade process is terminated, and errors are logged in a file named '`upgrade-fortisoar-<version>-<timestamp>.log`' in the `/var/log/cyops` folder. Resolving these errors and executing the upgrade command will resume the process from the point of failure. This approach improves the upgrade experience and provides benefits for handling upgrade failures, including displaying inline resolutions for encountered issues, resolving issues in task files, and resuming the upgrade process from the point of failure, etc.

Optimizing FortiSOAR Upgrades

Prior to release 7.6.0, FortiSOAR relied on the `yum` repository for updates. However, a slow connection to the repository could cause significant delays in the upgrade process, leading to increased application downtime and user inconvenience. In release 7.6.0 the upgrade process has been optimized to minimize application downtime by offering users the option to pre-download the necessary FortiSOAR upgrade packages and store them locally. During the upgrade, the process accesses these local packages, ensuring a faster and more reliable upgrade experience. Once the upgrade is complete, FortiSOAR will revert to using the `yum` repository for future updates.

It is recommended to upgrade your FortiSOAR instance promptly after downloading the upgrade packages to ensure that the latest packages are installed. Delaying the upgrade could result in outdated packages being installed.

Manage FortiSOAR upgrades

To manage your FortiSOAR upgrades post the 7.5.0 release, i.e., from FortiSOAR, Use the '`upgrade`' subcommand of the FortiSOAR Admin CLI ('`csadm`'). The various arguments that you can use with the `csadm upgrade` subcommand are explained in the Arguments available for the '`csadm upgrade`' subcommand topic.

Before you start upgrading your FortiSOAR system, make sure to disable the IPv6 protocol on your VM if you are not using the IPv6 protocol.

To upgrade your system to FortiSOAR from 7.5.0 to 7.6.0, perform the following steps:

1. Users who have `root` access must perform the upgrade process.
2. ssh to the VM that you want to upgrade.
3. Check your system to see if `tmux` is installed; if not, use the following command to install it:
   ```
   sudo yum install -y tmux
   ```
   Next, check that you are connected to a `tmux` session. A tmux session is needed for situations where network connectivity is less than favorable. You can check your tmux session using the following command:
   ```
   # tmux ls
   ```
   This command returns an output such as the following example:
   ```
   0: 1 windows (created Thu Nov 24 09:37:47 2022) [170x47]
   ```
   Log back into the SSH console and run the following command to reattach the `tmux` session:
   ```
   tmux attach-session -t 0
   ```
   If you do not find any `tmux` session, connect to one using the following command:
   ```
   # tmux
   ```
4. Run the following command to check if your FortiSOAR system is prepared for an upgrade to the 7.6.0 release:
   ```
   csadm upgrade check-readiness [--version VERSION]
   csadm upgrade check-readiness --version 7.6.0
   ```
   The check-readiness argument runs various checks including checking if there is sufficient space available for the upgrade. If there is insufficient space in any directory during the space check, appropriate messages will be added to the check-readiness report. The report will contain information on the recommended space, currently available space, and the additional space required to meet the recommendation. The check-readiness report is generated at '`/opt/fsr-elevate/elevate/outputs/`' and will include explanatory messages for any failures.
   After addressing any validation failures to ensure system readiness for the upgrade, rerun the `csadm upgrade check-readiness` command to confirm that the system is prepared for the upgrade.
5. (Optional) To download upgrade packages locally first and then upgrade your system at a later time, use the following command:
   `csadm upgrade execute [--target-version TARGET_VERSION] --download-packages] [--local-download-directory LOCAL_DOWNLOAD_DIRECTORY]`. This command will verify the availability of sufficient space in the default `/opt/cyops/packages` directory or in a directory specified by the user in the `--local-download-directory` argument. If enough space is present, the packages will be downloaded to the designated directory. If space is insufficient, the command will exit with an error message indicating the issue. Use the `--local-download-directory` argument to specify the absolute path in the local directory where the upgrade packages should be downloaded. By default, the FortiSOAR packages, OS Packages and third-party

packages are downloaded to the `/opt/cyops/packages/fsr-packages` folder, while system-connectors rpms are downloaded to the `/opt/cyops/packages/fsr-connectors` folder.
NOTE: If the upgrade packages are successfully downloaded to the local directory, the upgrade process will utilize those packages for the upgrade. Otherwise, the upgrade process will access the `yum` repository for the upgrade. Once the upgrade is complete, FortiSOAR will revert to using the `yum` repository for future updates. Additionally, note that if you re-run the `csadm upgrade` command with the `--download-packages` argument, it will download the packages again, overwriting the previously downloaded ones.

6. Run the following command to upgrade your system:
   ```
   csadm upgrade execute [--target-version TARGET_VERSION].
   ```
   For example to upgrade to 7.6.0 use the , `csadm upgrade execute --target-version 7.6.0` command.
   IMPORTANT: If your instance can only connect to "repo.fortisoar.fortinet.com" by using a proxy, then ensure that the proxy is set in the `/etc/wgetrc` file. For example,
   ```
   use_proxy=yes
   http_proxy=<proxy_server_ip:port>
   https_proxy=<proxy_server_ip:port>
   ```
   You can also set the proxy while running the FortiSOAR Configuration Wizard or by using the `csadm network` command.
   Important Notes: To upgrade a high availability cluster in FortiSOAR, you require to upgrade each node individually, one after the other. For more information, see the Upgrading a FortiSOAR High Availability Cluster section. For information on how to upgrade a FortiSOAR distributed multi-tenant configuration to 7.6.0, see the Upgrading a FortiSOAR Distributed Multi-Tenancy Configuration section.Note that when you upgrade your FortiSOAR enterprise instance, High Availability (HA) cluster, or a distributed multi-tenant configuration, the FortiSOAR appliance hostkey also gets changed.
7. Once your FortiSOAR instance is upgraded, you must log out from the FortiSOAR UI and log back into FortiSOAR.

Arguments available for the '`csadm upgrade`' subcommand

'`upgrade`' is a new subcommand added to the `csadm` utility in release 7.5.0. It is not available in releases prior to 7.5.0. This command can only be used to upgrade from release 7.5.0 to a later release, such as 7.6.0 or higher. For releases prior to 7.5.0, use the upgrade script to upgrade your FortiSOAR system.

Arguments for the '`csadm upgrade`' subcommand are as follows:

- `check-readiness [--version VERSION]`: This option checks if your FortiSOAR system is prepared for an upgrade to the release specified in the 'target-version' argument. This option executes the pre-upgrade phase and saves a report with the results in JSON format at `/opt/fsr-elevate/elevate/outputs/`. All pre-upgrade validations are performed during the 'pre-upgrade' phase. A sample of a check-readiness report is present in the Example of a check-readiness report topic.
  The format of the JSON file is:
  ```
  {
         <short description of task>:
     {
        "result":<boolean value>,
         "msg":<string value>
     }
  }
  ```
- `execute-task [--target-version TARGET_VERSION] [--phase PHASE] [--task-name TASK_NAME]`: This option allows you to run a specific task during a particular phase of the upgrade process. The possible options for the '`phase`' argument are `pre-upgrade`, `post-upgrade`. For details on 'tasks', see the Task Generator Utility topic.
  For instance, to execute a task file named '01_initialize' in the pre-upgrade phase, use the command:

`csadm upgrade execute-task --target-version 7.6.0 --phase pre-upgrade --task-name 01_initialize`
This option is useful for testing custom task files or modifications in existing task files.
Note the following important points

- Tasks from the 'pre-upgrade' phase can only be executed when the target version is higher than the current version.
- Tasks from the 'post-upgrade' phase can be executed when the target version is higher than or equal to the current version.
- Tasks from the 'upgrade' phase cannot be executed using this command.

NOTE: After running the check-readiness command to assess their readiness for upgrading to version 7.6.0 or later, users on release 7.5.0 will also see the execute-task argument. However, users on 7.5.0 cannot use the csadm upgrade command with this argument. To utilize this argument, run the following command using "python3":
`/opt/fsr-elevate/elevate/.env/bin/python3 /opt/fsr-elevate/elevate/main.py execute-task[--target-version TARGET_VERSION] [--phase PHASE] [--task-name TASK_NAME]`
For example, to run a task file named '01_remove_security_patch_versions' in the post-upgrade phase, use the following command:
`/opt/fsr-elevate/elevate/.env/bin/python3 /opt/fsr-elevate/elevate/main.py execute-task --target-version 7.6.0 --phase post-upgrade --task-name 01_remove_security_patch_versions`

- `execute-phase [--target-version TARGET_VERSION] [--phase PHASE]`: This option executes the specified phase in the upgrade process of the given version. The possible options for the '`phase`' argument are `pre-upgrade`, `post-upgrade`.
  For example, `csadm upgrade execute-phase --target-version 7.6.0 --phase pre-upgrade` executes the pre-upgrade phase of upgrading your FortiSOAR instance to release 7.6.0. This assists in anticipating problems and taking proactive measures to fix them before moving forward with the full upgrade.
  IMPORTANT: The 'post-upgrade' phase must be executed after upgrading the FortiSOAR instance to the target version. This phase will not be executed if the target version is not the same as the current version. For example, if your FortiSOAR instance is on version 7.6.0, then you can run post-upgrade for version 7.6.0 using the command `csadm upgrade execute-phase --target-version 7.6.0 --phase post-upgrade`. However, the command `csadm upgrade execute-phase --target-version 9.9.9 --phase post-upgrade` will fail with a message such as "`The post-upgrade phase can only be executed for the 9.9.9 version after upgrading FortiSOAR to the 9.9.9 version. The current version is 7.6.0, and the post-upgrade phase can be executed for the current version.`" Additionally, if the `execute-phase` or `execute` options encounter a failure while executing a specific task, the subsequent execution of the same phase starts with the tasks that failed. Tasks that completed successfully prior to the failure are not executed again.
- `execute [--target-version TARGET_VERSION] [--download-packages] [--local-download-directory LOCAL_DOWNLOAD_DIRECTORY]`: This option downloads packages that are required during the upgrade of your FortiSOAR instance to the specified release on your system.
  Use the `csadm upgrade execute --target-version 7.6.0 --download-packages` command to download upgrade packages required to upgrade your system to 7.6.0 to the default `/opt/cyops/packages` directory. Additionally, you can use the `--local-download-directory` argument to specify the absolute path in the local directory where the upgrade packages should be downloaded.
  Use the `csadm upgrade execute --target-version TARGET_VERSION` command to upgrade your system. For example, to upgrade FortiSOAR from release 7.5.0 to release 7.6.0 after downloading the upgrade packages locally, run the `csadm upgrade execute --target-version 7.6.0` command.
  NOTE: When you execute the command `csadm upgrade execute --target-version <TARGET_VERSION>`, a log file named '`upgrade-fortisoar-<target_version><timestamp>.log`' is created in the `/var/log/cyops` folder. For example, running the command `csadm upgrade execute --target-version 7.6.0` will generate the `upgrade-fortisoar-7.6.0-2024-05-22-1708597271.log` file in the `var/log/cyops` folder.
  This log file contains the complete CLI output, allowing you to review all the steps of the FortiSOAR upgrade

process and can also be viewed during a '`tmux`' session. You can also use '`tail -f`' to monitor the update from a different system.
If a failure occurs during the upgrade process, the upgrade process is terminated, and errors are logged in '`upgrade-fortisoar-<target_version><timestamp>.log` file. Resolving these errors and executing the upgrade command again resumes the process from the point of failure.

- `create-task [--phase PHASE] [--task-name TASK_NAME] [--cls-name CLS_NAME]`: This option adds a new task file to the specified upgrade phase based on the task name and class name you have specified. For details on naming conventions of tasks and class files and other information about the task file, see Task Generator Utility.
- `create-shell-script [--phase PHASE] [--shell-script-name SHELL_SCRIPT_NAME]`: This option adds a new shell script file to the specified upgrade phase based on the shell script name you have specified. For details on naming conventions of the script file and other information about the script file, see Shell Script Generator Utility.

Example of a check-readiness report

```
{
    "metadata": {
        "Current FortiSOAR version": "7.5.0",
        "File creation time": "24/06/2024, 06:40:03",
        "File modification time": "24/06/2024, 07:05:26"
    },
    "Verify Operating System Compatibility": {
        "result": true,
        "message": "The current operating system is Rocky Linux, which is supported for
upgrade."
    },
    "Verify Yum Repo Connection": {
        "result": true,
        "message": "Connection to 'https://repo.fortisoar.fortinet.com' repo is successful"
    },
    "Verify Instance Type Compatibility": {
        "result": true,
        "message": "Current instance is of type 'enterprise'."
    },
    "Check '/' Directory Free Space": {
        "result": true,
        "message": "Required free space is available in '/'."
    },
    "Check '/boot' Directory Free Space": {
        "result": true,
        "message": "Required free space is available in '/boot'."
    },
    "Check '/var/log' Directory Free Space": {
        "result": true,
        "message": "Required free space is available in '/var/log'."
    },
    "Check '/opt' Directory Free Space": {
        "result": true,
        "message": "Required free space is available in '/opt'."
    },
    "Check '/var/tmp' Directory Free Space": {
        "result": true,
        "message": "Required free space is available in '/var/tmp'."
    },
    "Verify Cyops RPM Installation": {
        "result": true,
        "message": "'cyops' rpm is installed on this instance."
    },
    "Verify Publish Status Of All Modules": {
        "result": true,
        "message": "All modules in current system are in published state"
    },
    "Verify presence of cluster": {
        "result": true,
        "message": "No other cluster nodes found."
    },
    "Install Cyops Repo Update": {
        "result": true,
        "message": "Successfully installed /opt/fsr-elevate/elevate/cyops-repo-update-
7.6.0.el9.x86_64.rpm."
    }
}
```

The '`metadata`' key in the check-readiness report contains the following data:

- The "`Current FortiSOAR version`" key contains the version of FortiSOAR on which the report is generated
- The "`File creation time`" key contains the date and time when the report was generated.
- The "`File modification time`" key contains the date and time when the report was modified upon rerunning the `csadm upgrade check-readiness` command. It will be empty when the report is first generated.

Task Generator Utility

To add a new task file to the specified upgrade phase, use the following command:

```
csadm upgrade create-task --phase {pre-upgrade,post-upgrade} --task-name TASK_NAME --
cls-name CLS_NAME
```

The arguments must follow these conventions:

Argument Name	Conventions
`--phase`	Phase in which you want to create the task file. Options are `pre-upgrade`, `post-upgrade`
`--task-name`	Name of the task file. The task file name must follow these conventions: • Start with a number that represents the sequence of the task's execution • The execution order sequence number must be followed by an underscore • Use Snake case An example of a valid task name is `10_test_task`
`--cls-name`	Name of the class in which the user wants to create the task file. The class file name must use Pascal case. For example, `TestTask`.
NOTE: Task files get generated at: `/opt/fsr-elevate/elevate/workflow/<phase>/<task_file_name>`	

For example, to add a task named 'test_task' to the pre-upgrade phase, which should be run as the 5th task in the pre-upgrade phase, use the following command:

```
csadm upgrade create-task --phase pre-upgrade --task-name 05_test_task --cls-name
```

`TestTask`
This command generates the task file '`05_test_task`' in the specified phase '`pre-upgrade`'. The generated task file will be located at `/opt/fsr-elevate/elevate/workflow/pre_upgrade/05_test_task.py`

About the task file

The task file generated contains a class with the specified name, which inherits from the parent class 'Tasks'. To be considered as a registered task file for the phase, the task file must be inherited from the parent 'Tasks' class. For more information on the parent 'Tasks' class, see Functions available from the parent 'Tasks' class.

The generated class includes the following functions:

- `def __init__( ) -> None`: This constructor function initializes several variables that can be used in development:
  - `self.logger`: A 'Logger' class object used for logging messages.
  - `self.cmd_line_utilities`: A 'CmdUtils' class object used for accessing the '`execute_cmd`' function from the 'CmdUtils' class. This function allows commands to be executed on the CLI.
    The '`execute_cmd`' function in the '`CmdUtils`' class takes the two parameters:
    - '`cmd`', which is of 'string' type and takes a command as an argument.
    - '`trap_output`', which is of 'boolean' type, and its value controls whether or not the output of a command is shown on the CLI. If it is set to 'true', the command's output won't be displayed on the CLI; if it is set to 'false', the output will be displayed.
  - `self.store_result_path`: A variable that stores the path of the '`step_result-<version>.json`' file.
  - `self.report_path`: A variable that specifies the path of the '`report-<version>.json`' file.
    NOTE: `<version>` get replaced by the specified target version. For example, if the '`check-readiness`' command is being executed for FortiSOAR 7.6.0 version, then `step_result-7.6.0.json` and `report-7.6.0.json` are generated in the '`/opt/fsr-elevate/elevate/outputs`' directory.
- `def tags( ) -> str`: This function is used to tag tasks.
- `def get_description( ) -> str`: This function is used to provide descriptions for a task file, which are displayed on the CLI when the task is being executed.
- `def is_supported( ) -> bool`: This function is used to add conditions for the execution of a task file. For example, if you want to execute a particular task file only when a specific condition is met, you can define the condition using this function. Then the task file will only be run when the condition is met.
- `def execute( ) -> bool`: This function is used to execute the main code of the task.
- `def validate( ) -> bool`: This function is used to execute any validations that have been added for the task.
- `def _print_status_msg(msg:str, status:str) -> None`: This function is used to display the status of the task file in color on the CLI while the task is being executed. The first parameter, `msg:str`, is a short, one-line description of the task file. The second parameter, `status:str`, indicates the status of the task. Task status can be set using the imported '`TASK_STATUS`' constant.

Functions available from the parent 'Tasks' class

- `def store_result_to_json( json_file_path, phase_name, task_name, key, data ) -> None`: This function stores data in `report-<version>.json` or `step_result-<version>.json` files.
  To store data in the `step_result-<version>.json` file, pass five arguments: the first is the `step_result-<version>.json` file path, the second is the phase name, the third the task file name, the fourth is the key, and the fifth is the value of the key. The data is stored as per the specified key and value in the provided task name, which is created in the provided phase in a `.json` file. To update any particular key in `step_result-<version>.json` provide that particular key as the fourth argument and its updated value as the fifth argument. To update multiple keys, provide ‘None’ for the key argument, and their updated JSON data as the fifth argument.
  To store data in the `report-<version>.json` file, pass five arguments: the first argument is the `report-`

`<version>.json` file path, the second and third arguments are 'None', the fourth is the key, and the fifth is the value of the key.
NOTE: In the `step_result-<version>.json` and `report-<version>.json` files, a `'metadata'` key is added, which contains information about the version in which the files were created and the timestamp when the files were created or modified. The format of the 'metadata' is as follows:

```
"metadata": {
   "Current FortiSOAR version": "7.6.0",
   "File creation time": "20/05/2024, 07:03:39",
   "File modification time": ""
}
```

where,
'`Current FortiSOAR version`' contains information about the FortiSOAR version in which these files were created.
'`File creation time`' contains the timestamp of when these files were created.
'`File modification time`' contains the timestamp of when these files were modified using command execution.

- `def get_step_results( phase_name, task_name ) -> dict`: This function retrieves data from the `step_result-<version>.json` file. To retrieve data from the `step_result-<version>.json` file, pass two arguments: the first is the phase name and the second is the task name. This function returns the data stored in the '`key`' of the '`task_name`' that is present in the '`phase_name`' in '`step_result-<version>.json`' file.
- `def print_txt( txt )`: This function prints text in the 'justified' format.

Each task file is independent and is intended to perform a specific action, so they cannot be imported into one another. However, changes made in the `step_result-<version>.json` file might impact on other tasks due to their interdependence, when for example the updates made in the `step_result-<version>. json` file from one task file are expected in another task file that should be executed after the first one.

Sample Task File

Task File Name : `05_test_task.py`
Class Name : `TestTask`

```
"""AUTOGENERATED python code."""

from framework.base.tasks import Tasks
from helpers.cmd_utils import CmdUtils
from helpers.logger import Logger
from constants import FRAMEWORK_PATH, LOG_FILE, REPORT_FILE, STEP_RESULT_FILE, TASK_STATUS,
TASK_LOG_STATUS, TEXT_COLOR, TEXT_DECORATION


class TestTask(Tasks):
    """Implementation class for tasks"""

    def __init__(self) -> None:
        super().__init__()
        self.logger = Logger.get_logger(__name__)
        self.cmd_line_utilities = CmdUtils()
        self.store_result_path = STEP_RESULT_FILE.format(self.target_upgrade_version)
        self.report_path = REPORT_FILE.format(self.target_upgrade_version)
    @property
    def tags(self) -> str:
        """Tags for task"""
        return 'pre_upgrade'


    def get_description(self) -> str:
        """Description of a method"""
        return ""


    def is_supported(self) -> bool:
        """To check task is for target upgrade version/platform"""
        current_version = int(self.current_version.replace('.', ''))
        target_upgrade_version = int(self.target_upgrade_version.replace('.', ''))
        return target_upgrade_version > current_version


    def execute(self) -> None:
        """To execute main code"""
        # TODO
        pass


    def validate(self) -> bool:
        """Write validations in this function"""
        return self.work_on_something()


    def _print_status_msg(self, msg:str, status:str) -> None:
        """Use this function to print status message of task execution"""
        reset = TEXT_COLOR["RESET"]
        if status == TASK_STATUS["DONE"]:
            color = TEXT_COLOR["GREEN"]
        else:
            color = TEXT_COLOR["RED"]
        truncated_message = msg[:65] + "..." if len(msg) > 65 else msg
        width = 8
        status = f"{status:^{width}}"
        colored_status = f"{color}{status}{reset}"
        final_msg = "{:<70}{}[{}]".format(truncated_message," ",colored_status)
        print(final_msg)

    def work_on_something(self):
        step_result = self.get_step_results('pre-upgrade', 'initialize')
        s_release_version = step_result['s_release_version']
        if s_release_version == '7.6.0':
            # ToDo
            pass

        # There are two ways to update data in step_result-<version>.json
        # Added/Updated key in dictionary and storing dictionary
        step_result['new_key_1'] = 'new_value_1'
        self.store_result_to_json(self.store_result_path, 'pre-upgrade', 'initialize', None,
step_result)
```

```
        # Adding/Updating key directly in step_result-<version>.json file
        self.store_result_to_json(self.store_result_path, 'pre-upgrade', 'initialize', 'new_
key_2', 'new_value_2')
        return True
```

Shell Script Generator Utility

To add a new shell script file to the specified upgrade phase, use the following command:
`csadm upgrade create-shell-script --phase {pre-upgrade,post-upgrade} --shell-script-name SCRIPT_NAME`
The arguments must follow these conventions:

Argument Name	Conventions
`--phase`	Phase in which you want to create the shell script file. Options are `pre-upgrade`, `post-upgrade`
`--task-name`	Name of the shell script file. The shell script file name must be in Snake case. An example of a valid shell script name is `test_script`
NOTE: Script files get generated at: `/opt/fsr-elevate/elevate/workflow/<phase>/scripts/<script_file_name>`	

For example, to add a shell script to the post-upgrade phase, use the following command:
`csadm upgrade create-shell-script --phase post-upgrade --shell-script-name test_script`
This command generates the script file '`test_script`' in the specified phase '`post-upgrade`'. The generated script file will be located at `/opt/fsr-elevate/elevate/workflow/post_upgrade/scripts/test_script.sh`

About the Shell Script file

Some operations are more conveniently handled using a shell script. In such cases, you can create shell scripts using the Shell Script Generator Utility. During phase execution, only task files are called and executed; therefore, to call a shell script during phase execution, you must create a custom task file in that phase that will call the shell script. If you are already using custom shell scripts to perform some operations before or after the FortiSOAR upgrade, you can create custom task files and call the shell scripts using the task files. For testing purposes, you can also independently execute these shell scripts.

The generated shell script file contains two functions, which are as follows:

- `store_to_json( )`: This function stores data in `step_result-<version>.json` files.
  To use this function, pass five arguments: the first is the target upgrade version of FortiSOAR, the second is the phase name, the third is the task file name, the fourth is the key, and the fifth is the value of the key.
  For example, if you want to add 'test_value' against 'test_key' in the `step_result-<version>.json` file in the 'initialize' task in the 'pre-upgrade' phase for the '7.6.0' version, then pass the arguments to the `store_to_json( )` function as follows:
  `store_to_json '7.6.0' 'pre-upgrade' 'initialize' 'test_key' 'test_value'`
  This command adds the 'test_key' and its corresponding value, 'test_value' into the `step_result-<version>.json` file. If the ' test_key' is already present in the `step_result-<version>.json` file, then it updates its value to the provided value, 'test_value'.
- `get_data_from_json( )`: This function retrieves data from the `step_result-<version>.json` file. To use this function, pass four arguments: the first is the target upgrade version of FortiSOAR, the second is the phase

name, the third is the task file name, and the fourth is the key.
For example, if you want to retrieve 'test_value' stored against the 'test_key' stored in the 'initialize' task in the 'pre-upgrade' phase for the '7.6.0' version, then pass the arguments to the `get_data_from_json( )` function as follows:
`get_data_from_json ‘7.6.0’ ‘pre-upgrade’ ‘initialize’ ‘test_key’`

Sample Shell Script

Shell Script File Name : `test_script.sh`

```
#!/usr/bin/env bash
#AUTOGENERATED shell script code.

store_to_json(){
    # This method can be used to store data in step_result.json
    # $1 is target fsr version which is passed to this file during execution. Pass $1 as it
is as first argument while calling this function
    # $2 is to take phase_name
    # $3 is to take task_name
    # $4 is to take key for data to store in json
    # $5 is to take data to store in json
    json_file_path="/opt/fsr-elevate/elevate/outputs/step_result-$1.json"
    python3 -c "import sys; sys.path.append('/opt/fsr-elevate/elevate'); from
framework.utility.task_utils import TaskUtilities; task_utils=TaskUtilities(); task_
utils.store_result_to_json(\"$json_file_path\",\"$2\",\"$3\",\"$4\",\"$5\")"
}

get_data_from_json(){
    # This method can be used to get data from step_result.json
    # $1 is target fsr version which is passed to this file during execution. Pass $1 as it
is as first argument while calling this function
    # $2 is to take phase_name
    # $3 is to take task_name
    # $4 is to take key in which data is stored
    target_version=$1
    json_file_path="/opt/fsr-elevate/elevate/outputs/step_result-$target_version.json"
    if [ $# = 1 ];then
    result=$(jq -r "" "$json_file_path")
    elif [ $# = 2 ];then
    key1=$2
    result=$(jq -r --arg key1 "$key1" '.[$key1]' "$json_file_path")
    elif [ $# = 3 ];then
    key1=$2
    key2=$3
    result=$(jq -r --arg key1 "$key1" --arg key2 "$key2" '.[$key1][$key2]' "$json_file_
path")
    elif [ $# = 4 ];then
    key1=$2
    key2=$3
    key3=$4
    result=$(jq -r --arg key1 "$key1" --arg key2 "$key2" --arg key3 "$key3" '.[$key1][$key2]
[$key3]' "$json_file_path")
    else
    result="Filter upto 3 level is supported. Provided more than 3 filter parameters in
'$0'"
```

```
    fi
    if [ "${result}" = "null" ];then echo "Please provide valid filter key in '$0'."
    else echo "$result"
    fi
}

# This will get value stored against 's_release_version' key in step_result-<version>.json
file
data=$(get_data_from_json '7.6.0' 'pre-upgrade' 'initialize' 's_release_version')
# echo $data => '7.6.0'

# This will store/update key and value in step_result-<version>.json file
store_to_json '7.6.0' 'pre-upgrade' 'initialize' 's_release_version' '7.6.0'
```

Upgrading a FortiSOAR High Availability Cluster

This section describes the procedure to upgrade a FortiSOAR High Availability (HA) cluster. This section considers that the HA setup has a Reverse Proxy or Load Balancer such as "HAProxy" configured.

Refer to the Preparing to Upgrade FortiSOAR section and ensure that all the prerequisites mentioned in that section are met. The upgrade installer will handle all FortiSOAR services management.

Upgrading an Active-Active HA Cluster

For the purpose of the following procedure, *Node1* is considered as the Active Primary node, *Node2* is considered as the Active Secondary node. Both the nodes are fronted by a Reverse Proxy or Load Balancer such as "HAProxy".

Approximately 30 minutes of downtime is required for the upgrade.

To upgrade your active-active HA cluster to FortiSOAR 7.6.0, perform the following steps:

1. Configure the Reverse Proxy to pass requests only to *Node1*.
   This ensures that FortiSOAR requests are passed only to *Node1*, and *Node2* can be upgraded.
2. Use the `#csadm ha` command as a root user and run the `suspend-cluster` command on *Node2*.
   This makes *Node2* a standalone system.
3. Upgrade *Node2* using the process mentioned in the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework chapter.
   Once the upgrade of *Node2* is completed successfully, you can now upgrade *Node1*.
   Important: Upgrade of *Node1* will incur downtime.
4. Once both the nodes are upgraded then run the `resume-cluster` command from *Node2*.
5. Configure the Reverse Proxy again to handle requests from both *Node1* and *Node2*.

Upgrading an Active-Passive HA Cluster

For the purpose of the following procedure, *Node1* is considered as the Active Primary node, *Node2* is considered as the Passive Secondary node. Both the nodes are fronted by a Reverse Proxy or Load Balancer such as "HAProxy".

Approximately 30 minutes of downtime is required for the upgrade.

To upgrade your active-passive HA cluster to FortiSOAR 7.6.0, perform the following steps:

1. Reverse Proxy is configured to have *Node2* as backup system. Therefore, you require to comment out that part from Reverse Proxy configuration.
2. Use the `#csadm ha` command as a `root` user and run the `suspend-cluster` command on *Node2*..
   This makes *Node2* a standalone system.
3. Upgrade *Node2* using the process mentioned in the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework chapter.
   Once the upgrade of *Node2* is completed successfully, you can now upgrade *Node1*.
   Important: Upgrade of *Node1* will incur downtime.
4. Once both the nodes are upgraded then run the `resume-cluster` command from *Node2*.
5. Configure the Reverse Proxy again to set *Node2* as the backup server.

Upgrading a FortiSOAR Distributed Multi-Tenancy Configuration

This section describes the procedure to upgrade a FortiSOAR distributed multi-tenant configuration for managed security services providers (MSSPs) or Distributed SOC configuration.

You must first upgrade the master node of your FortiSOAR distributed multi-tenant configuration and only then upgrade the tenant nodes of your FortiSOAR multi-tenancy setup.

In case of a distributed deployment, both the master and the tenant nodes must be upgraded. A version mismatch will not work if either of them upgrades to 7.6.0.

Upgrading a FortiSOAR master node

Before you upgrade your FortiSOAR master node, ensure the following:

- All playbooks have completed their execution on the master.
- The tenant node(s) are deactivated from the master node before upgrading the master node, and tenant nodes have disabled communication to the master node from the "`Master Configuration`" page.

If the master node of your multi-tenant configuration is part of an HA setup, i.e., MSSP +HA, then follow the steps mentioned in the Upgrading a FortiSOAR High Availability Cluster chapter.

If the master node of your multi-tenant configuration is not part of an HA setup, then follow the steps mentioned in the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework chapter.

Upgrading a FortiSOAR Tenant node

Before you upgrade your FortiSOAR tenant node, ensure the following:

- Data replication from the tenant node to the master node is stopped. You can stop data replication by logging on to the tenant node and clicking **Settings** to open the `System` page, then in the `Multi Tenancy` section, click the **Master Configuration** menu item and then in the `Communication With Master Node` section, toggle the **Enabled** button to **NO**.
  Once you have completed the upgrade process, i.e., upgrading both your master and tenant nodes from a release prior to 7.4.2 to release 7.4.2 or later, you must restart the `cyops-integrations-agent` service on tenant nodes using the following command:
  ```
  systemctl restart cyops-integrations-agent
  ```
  You must restart the `cyops-integrations-agent` service at tenant nodes before you download the agent's or tenant's logs, from the master node's console..
- All playbooks have completed their execution on the tenant.
- All schedule playbooks that fetch data from data sources to the tenant are stopped.
- Any application that pushes data from data sources to the tenant is stopped.

If the tenant node of your multi-tenant configuration is part of an HA setup, i.e., MSSP +HA, then follow the steps mentioned in the Upgrading a FortiSOAR High Availability Cluster section.

If the tenant node of your multi-tenant configuration is not part of an HA setup, then follow the steps mentioned in the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework section.

After the tenant node has been successfully upgraded, you must toggle the **Allow Module Management** setting to **NO** and then back to **YES**. This is needed only if you were already using the 'Allow Module Management' feature and is required to synchronize the tenant module metadata with the master instance. You can ignore this step, if your 'Allow Module Management' setting was already disabled before the upgrade.

Upgrading a FortiSOAR Secure Message Exchange

A secure message exchange establishes a secure channel that is used to relay information to the agents or tenant nodes. To create a dedicated secure channel, you are required to add the reference of the installed and configured secure message exchange, when you add agent or tenant nodes to your environment. For information on agents see the *Segmented Network support in FortiSOAR* chapter in the "Administration Guide," and for more information on secure message exchange and tenants, see the "Multi-Tenancy support in FortiSOAR Guide".

1. Ensure that you stop data replication between the master and the tenant nodes. You can stop data replication by logging on to the tenant node and clicking **Settings** to open the `System` page, then in the `Multi Tenancy` section, click the **Master Configuration** menu item and then in the `Communication With Master Node` section, toggle the **Enabled** button to **NO**.
2. SSH to the secure message exchange VM that you want to upgrade as a '`root`' user.
3. Check your system to see if `tmux` is installed; if not, use the following command to install it:
   ```
   sudo yum install -y tmux
   ```
   Next, check that you are connected to a `tmux` session. A tmux session is needed for situations where network connectivity is less than favorable. You can check your tmux session using the following command:
   ```
   # tmux ls
   ```
   This command returns an output such as the following example:
   ```
   0: 1 windows (created Thu Nov 24 09:37:47 2022) [170x47]
   ```
   Log back into the SSH console and run the following command to reattach the `tmux` session:
   ```
   tmux attach-session -t 0
   ```
   If you do not find any `tmux` session, connect to one using the following command:
   ```
   # tmux
   ```
4. Run the following command to download the upgrade installer:
   ```
   # wget https://repo.fortisoar.fortinet.com/7.6.0/upgrade-fortisoar-7.6.0.bin
   ```
5. Run the upgrade installer using the following command:
   ```
   # sh upgrade-fortisoar-7.6.0.bin
   ```
   OR
   ```
   # chmod +x upgrade-fortisoar-7.6.0.bin
   # ./upgrade-fortisoar-7.6.0.bin
   ```
6. Once you have successfully upgraded the secure message exchange, start the data replication between the master and the tenant nodes again by toggling the **Data Replication** button to **ON**, and then verify the replication.

Upgrading a FortiSOAR Secure Message Exchange Cluster

RabbitMQ supports clustering, which, when combined with Queue Mirroring, enables an Active-Active configuration. For detailed setup instructions and guidance on monitoring queues, see the Clustering Guide and the Highly Available (Mirrored) Queues article. For optimal performance, the clustered instances should be managed by a TCP Load Balancer such as HAProxy, and clients should connect to the cluster via the proxy's address. For more information, see the *Multi-tenancy support in FortiSOAR* guide.

This procedure covers upgrading a two-node mirrored MQ cluster, both configured with the Reverse Proxy.

1. Configure the Reverse Proxy to route requests exclusively to *Node1*, which is the primary node of the MQ cluster. This ensures *Node1* handles all requests, while *Node2* (the secondary node) is available for maintenance.
2. Before upgrading, break the cluster on the secondary node (*Node2*) by executing the following commands:
   a. `rabbitmqctl stop_app`
   b. `rabbitmqctl reset`
   c. `rabbitmqctl start_app`
   NOTE: These commands will reset the RabbitMQ node. You will need to reconfigure the server using the `csadm mq db flush` command and add the '`admin`' user when prompted.
3. Log into the *Node2* terminal session as the `root` user, and upgrade *Node2* following the steps in the Upgrading a FortiSOAR Secure Message Exchange section.
   NOTE: Downtime begins when the upgrade process starts on *Node2*.
4. Remove the *Node1* entry from the Reverse Proxy.
5. Log into *Node1* terminal session as the `root` user, and upgrade *Node2* following the steps in the Upgrading a FortiSOAR Secure Message Exchange section.
   NOTE: Downtime ends when the upgrade process is completed on *Node1*.
6. Add the *Node1* entry back to the Reverse Proxy.
7. Once the SME cluster is upgraded, use the `join-cluster` command to create the SME cluster. For details, see the Setting up High Availability of the Secure Message Exchange topic in the *Multi-tenancy support in FortiSOAR* guide.
8. Reconfigure the Reverse Proxy to load balance requests between *Node1* and *Node2*.

Troubleshooting upgrade issues for MSSP setups

Replication from tenant to master stops once you upgrade an MSSP with an HA setup

If you have upgraded an MSSP+HA setup, then post-upgrade the replication from tenant nodes to the master node stopped.

Resolution

To resolve this issue, once you have upgraded your MSSP setup and created the HA cluster, you must restart all services on the primary master node and the primary tenant node using the following command:

```
csadm services --restart
```

Upgrading FortiSOAR using the Offline Repository

1. Ensure that the offline repository host is accessible from the FortiSOAR appliance and to ensure that the upgrade is not affected if the session times out, run the `tmux` command:
   ```
   [root@localhost ~]# tmux
   ```
2. If you are using your private repository to upgrade FortiSOAR, then specify that URL in the "`custom_yum_url`" key that is present in the `/opt/cyops/configs/fsr-elevate/config.yml` file before upgrading FortiSOAR.
3. Upgrade to FortiSOAR 7.6.0 using the process mentioned in the Upgrading a FortiSOAR Enterprise Instance using the Upgrade Framework chapter.
4. If you are using a self-signed certificate, then you must add your custom CA certificate in the OS and python trust store as a trusted certificate. For detailed steps, see the Adding a custom CA (self-signed) certificate in Rocky Linux or RHEL as a trusted certificate topic in the Additional configuration settings for FortiSOAR chapter of the "Deployment Guide."

Upgrading your FortiSOAR Docker image and upgrading your FortiSOAR Docker on an Amazon Elastic Kubernetes Cluster

Do not use the 'Upgrade Framework' to upgrade the Docker images from release 7.5.0 to 7.6.0. Follow the steps outlined in this chapter to upgrade the Docker instances.

Upgrading your FortiSOAR Docker image

1. Download the FortiSOAR docker image from https://support.fortinet.com; details are in the Downloading the FortiSOAR Docker image section of the "Deployment Guide".
2. Load the downloaded Docker image using the following command:
   `docker load -i <image-path>`
3. Download the FortiSOAR Docker installer from
   `https://repo.fortisoar.fortinet.com/7.6.0/install-fortisoar-docker-<release_version>.bin`
   For example, `https://repo.fortisoar.fortinet.com/7.6.0/install-fortisoar-docker-7.6.0.bin`
4. Update the `fortisoar.env` file with the ID of the Docker Image that is loaded in step 2. For more information, see Understanding the `fortisoar.env` file topic in the *Deploying FortiSOAR on a Docker Platform* chapter in the "Deployment Guide."
   Important: Ensure that the value of the `PROJECT_NAME` field in the `fortisoar.env` file must be the same as the value in the earlier version of the Docker image.
5. Before you begin the upgrade, it is recommended to take a backup of your FortiSOAR Docker as listed in the following commands.
   Note: The following commands uses `fortisoar_fortisoar_1` as the Docker name. You must replace this sample name with your own Docker name, which you can find using the `docker ps` command.
   a. `docker exec -ti fortisoar_fortisoar_1 bash -c 'export LANG=en_US.UTF-8;csadm db --backup /home/csadmin'`
      Note: This command stores the backup file at `/home/csadmin/DR_BACKUP_<release_version>-*_*_*.tgz` (for example, `/home/csadmin/DR_BACKUP_7.6.0-*_*_*.tgz`) inside your FortiSOAR Docker.
   b. Copy the backup file from your FortiSOAR Docker on the Docker host using the following command:
      `docker cp fortisoar_fortisoar_1:/home/csadmin/DR_BACKUP_<release_version>-*_*_*.tgz /data`
      For example, `docker cp fortisoar_fortisoar_1:/home/csadmin/DR_BACKUP_7.6.0-*_*_*.tgz /data`
      Note: The FortiSOAR Docker backup file is stored in the `/data` directory on your Docker host.
6. Stop your FortiSOAR Docker using the following command:
   `docker stop fortisoar_fortisoar_1`
7. Remove your FortiSOAR Docker using the following command:
   `docker rm fortisoar_fortisoar_1`

8. Run the FortiSOAR Docker using the updated `fortisoar.env` file that contains the ID of the new Docker Image using the following command:
   `./install-fortisoar-docker-<release_version>.bin --env-file fortisoar.env`
   For example, `./install-fortisoar-docker-7.6.0.bin --env-file fortisoar.env`

Reverting the upgrade on your FortiSOAR Docker image

In case the FortiSOAR Docker image upgrade fails, and you want to revert to the previous release, then you need to restore the backup of the previous version that was take in the `/data` directory on your Docker host. Following are the steps for restoring the backup.
Note: The following commands uses `fortisoar_fortisoar_1` as the Docker name. You must replace this sample name with your own Docker name, which you can find using the `docker ps` command.

1. Stop the running FortiSOAR Docker using the following command:
   `docker stop fortisoar_fortisoar_1`
2. Remove the FortiSOAR Docker using the following command:
   `docker rm fortisoar_fortisoar_1`
3. Remove the FortiSOAR Docker volumes using the following command:
   `docker volume rm $(docker volume ls --filter name=fortisoar_fortisoar_* -q)`
4. Update the `fortisoar.env` file with the ID of the previous Docker Image.
5. Run the previous version FortiSOAR Docker using the updated `fortisoar.env` file that contains the ID of the previous Docker Image using the following command:
   `./install-fortisoar-docker-<release_version>.bin --env-file fortisoar.env`
   For example, `./install-fortisoar-docker-7.6.0.bin --env-file fortisoar.env`
6. Wait until you see EULA page on the UI at `https://<docker-host-hostname>:<PORT_UI>/`
7. Copy the FortiSOAR Docker backup file from the `/data` directory on your Docker host using the following command:
   ```
   docker cp /data/DR_BACKUP_<release_version>-*_*_*.tgz fortisoar_fortisoar_
   1:/home/csadmin/
   docker cp /data/DR_BACKUP_7.6.0-*_*_*.tgz fortisoar_fortisoar_1:/home/csadmin/
   ```
8. Restore the Docker image using the following command:
   ```
   docker exec -it fortisoar_fortisoar_1 bash -c "export LANG=en_US.UTF-8;csadm db --
   restore /home/csadmin/DR_BACKUP_<release_version>-*_*_*.tgz"
   ```
   For example, `docker exec -it fortisoar_fortisoar_1 bash -c "export LANG=en_US.UTF-8;csadm db --restore /home/csadmin/DR_BACKUP_7.6.0-*_*_*.tgz"`

Upgrading your FortiSOAR Docker on an Amazon Elastic Kubernetes Cluster

1. Download the FortiSOAR docker image from https://support.fortinet.com; details are in the Downloading the FortiSOAR Docker image section of the "Deployment Guide".
2. Upload the downloaded FortiSOAR Docker image to your AWS Elastic container registry or any other Docker repository that is accessible from within your Kubernetes cluster. For example:
   `# docker push <account-id>.dkr.ecr.<region>.amazonaws.com/fortisoar/fortisoar:7.6.0`
3. Before you begin the upgrade process, it is recommended to take a backup of your FortiSOAR pod as listed in the following commands.
   The following commands uses `fsr-0` as the pod name. You must replace this sample name with your own pod

name that you can find using the `#kubectl get pods -o=name -n fsr` command.

```
#kubectl exec -ti -n fsr -c fsr fsr-0 -- bash -c "csadm db --backup
/\home/\csadmin/\ "
```

Note: This command stores the backup file in the `/home/csadmin/DR_BACKUP_<release_version>-*_*_*.tgz` folder inside your FortiSOAR pod. For example, `/home/csadmin/DR_BACKUP_7.3.2-*_*_*.tgz` inside your FortiSOAR pod.

Copy the backup file from your FortiSOAR pod on the EKS cluster node to your machine using the following command:

```
#kubectl cp fsr-0:/home/csadmin/DR_BACKUP_<release_version>-*_*_*.tgz -c fsr -n fsr
DR_BACKUP_<release_version>.tgz
```

4. Stop the running FortiSOAR pod using the following command:
   ```
   #kubectl delete statefulset <statefulset_name> -n <fortisoar_namespace>
   ```
5. Update the path of the new FortiSOAR image in the `fortisoar-statefulset.yaml` file.
6. Run the following command to deploy a new statefulset with the latest docker image:
   ```
   #kubectl apply -f fortisoar-statefulset.yaml
   ```

Post-Upgrade Tasks and Notes

While upgrading from 7.5.0 to 7.6.0 there are no specific post-upgrade tasks that require to be performed.

www.fortinet.com

Copyright© 2025 Fortinet, Inc. All rights reserved. Fortinet®, FortiGate®, FortiCare® and FortiGuard®, and certain other marks are registered trademarks of Fortinet, Inc., and other Fortinet names herein may also be registered and/or common law trademarks of Fortinet. All other product or company names may be trademarks of their respective owners. Performance and other metrics contained herein were attained in internal lab tests under ideal conditions, and actual performance and other results may vary. Network variables, different network environments and other conditions may affect performance results. Nothing herein represents any binding commitment by Fortinet, and Fortinet disclaims all warranties, whether express or implied, except to the extent Fortinet enters a binding written contract, signed by Fortinet's Chief Legal Officer, with a purchaser that expressly warrants that the identified product will perform according to certain expressly-identified performance metrics and, in such event, only the specific performance metrics expressly identified in such binding written contract shall be binding on Fortinet. For absolute clarity, any such warranty will be limited to performance in the same ideal conditions as in Fortinet's internal lab tests. Fortinet disclaims in full any covenants, representations, and guarantees pursuant hereto, whether express or implied. Fortinet reserves the right to change, modify, transfer, or otherwise revise this publication without notice, and the most current version of the publication shall be applicable.