

Keywords: FOS REST API Zoning Zone Configuration Merge

Description:

This video discusses how restore a zone database, merge zone databases, purge zones and aliases when decommissioning devices, perform zoning operations from an Excel workbook, and execute old CLI zoning scripts via the FOS RESTConf API.

## Disclaimers & Notices

- All sample scripts described herein are licensed under the Apache License, Version 2.0. See module headers for details.
- Sample scripts and libraries are provided as examples. Consoli-Solutions is not responsible for errors.
- Sample scripts and libraries are available for free when used by single customers for their own internal use. Use of these scripts and libraries in the manufacturing process, configuration of equipment for other customers, or providing services requires additional licensing.
- Consoli Solutions is an LLC

Consoli Solutions. Copyright (c) 2024

The legal disclaimer is here for you to read.

## Contact

Jack Consoli

[jack@consoli-solutions.com](mailto:jack@consoli-solutions.com)

<https://github.com/jconsoli>

Consoli Solutions. Copyright (c) 2024

Questions, comments, and feedback are always appreciated and I am available for consulting.

## YouTube Videos In This Series

Simplified Brocade FOS API: Getting Started

<https://www.youtube.com/watch?v=BWz7L0QOtYQ>

Simplified Brocade FOS API: Capture Data and Reports

[https://youtu.be/n9-Eni\\_AFCg](https://youtu.be/n9-Eni_AFCg)

Simplified Brocade FOS API: Configuring Switches

<https://www.youtube.com/watch?v=WGxXZrvhG2E>

Simplified Brocade FOS API: Zoning

<https://youtu.be/x1OvuRZRdA8>

Scripts & Documentation: <https://github.com/jconsoli>

Consoli Solutions. Copyright (c) 2024

Follow the github link at the bottom of the page for a copy of this presentation and the scripts described herein. The last section of the Getting Started video discusses common features and inputs for the sample scripts.

### Additional Notes

#### **Simplified Brocade FOS API: Getting Started**

Discusses how to download Python, what libraries are needed, how to install libraries, some tips for Python newbies, a few FOS settings you need to be aware of, an overview of the libraries I created to simplify scripting tasks, and an introduction to the sample scripts.

#### **Simplified Brocade FOS API: Configuring Switches**

Discusses a few nuances of configuring logical switches, Examples include: how to use sample scripts to configure logical switches from a workbook, and how to use an existing switch as a template for creating and configuring logical switches.

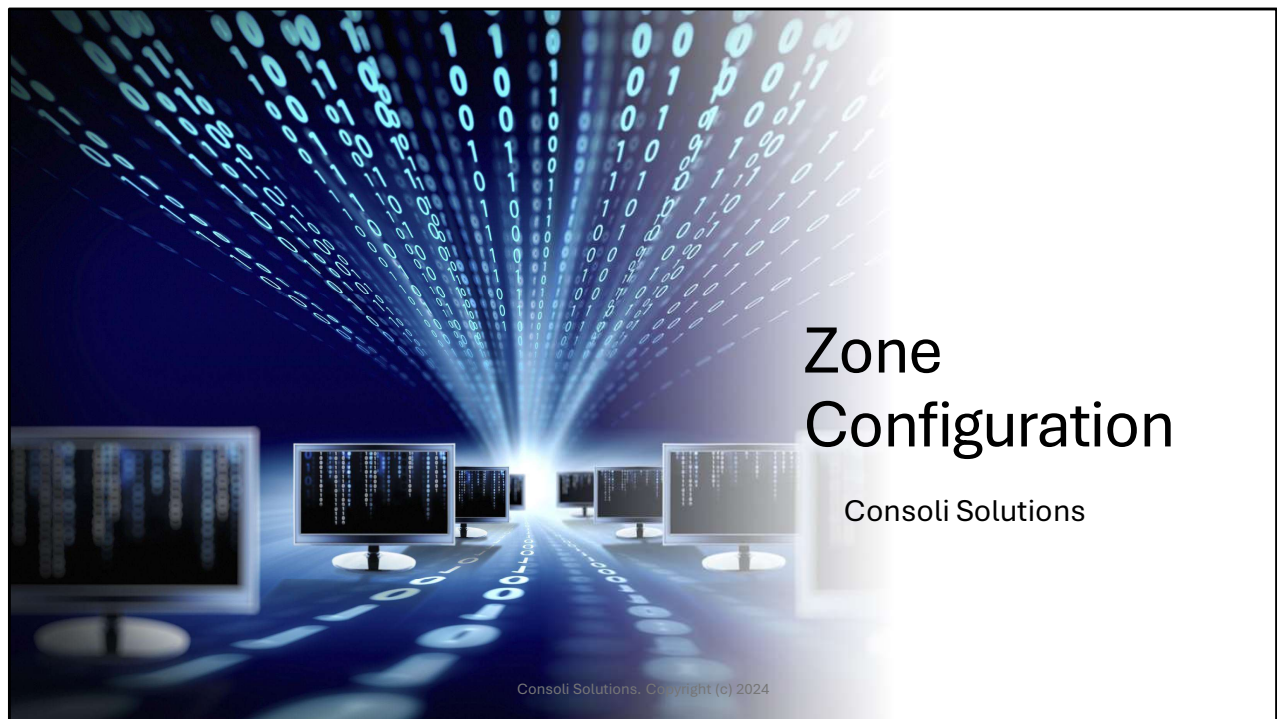
#### **Simplified Brocade FOS API: Capture Data and Reports (This video)**

Includes examples on how to collect data from switches, generate a general report, a

comparison report, a MAPS report, and an extensive nodefind utility.

### **Simplified Brocade FOS API: Zoning**

Zone from a workbook, restore zoning from a previous data capture, merge zoning databases from multiple fabrics, and zone from CLI



Let's start with zoning from a workbook

## zone\_config.py: Overview

- All zoning changes from a single worksheet
- Test and validate zoning changes in advance
- Minimize zone freeze windows
- Special features:
  - Purge zones and aliases
  - Wild card and RegEx matching/searching
  - Ask about FICON (mainframe) zoning

Consoli Solutions. Copyright (c) 2024

This module makes all zoning changes from a single worksheet. It's common to use this script for planning purposes and then use SANnav for actual implementation, but zone changes can be made from this script. The purge feature is intended for decommissioning storage arrays or server clusters.












### Additional Notes:

- All zoning changes, including standard and peer zones, are made from a single worksheet
- Test and validate zoning changes in advance
  - This script operates on live fabrics or from previously captured data.
  - It's similar to FOS in that all transactions are stored in a buffer, validated, and then, assuming the validation check passed, sent to the switch. The only difference is that changes don't need to be in order. For example, you can delete an alias and then the zone it's used in. As long as the zone database is valid after all changes are made, the test will pass.
- Minimize zone freeze windows
  - Even large zone changes can be implemented in seconds
  - Since zone changes can be validated in advance, there is usually no need to

- validate zone changes during the freeze
- If validating zone changes during the freeze is required, that happens in seconds
- Special features
  - For me, the most common use of this script is for planning device migrations or decommissions which is what the purge feature is for.
  - Most organizations have an alias naming convention that makes it easy to identify servers and storage arrays using standard RegEx or wild card filtering.
  - You can also create domain, index zones for mainframe environments.



## zone\_config.py: Input Parameters

Parameter	Description
 -ip, -id, -pw	Optional. Required when -i is not specified. Login credentials
 -fid	Optional. Required when -i is not specified. Fabric ID of logical switch.
 -i	Optional. Output of capture.py, multi_capture.py, or combine.py.
 -wwn	Optional. Fabric WWN. Required when -i is specified. Otherwise not used.
 -z	Workbook with zone definitions. ".xlsx" is automatically appended. See zone_config_sample.xlsx.
 -sheet	Sheet name in workbook, -z, to use for zoning definitions.
 -a	Optional. Name of zone configuration to activate (enable).
 -save	Optional. Save changes to the switch. By default, this module is in test mode only.
 -strict	Optional. When set, warnings are treated as errors.
 -scan	Optional. No parameters. Scan file specified with -i for fabric information.
 -cli	Optional. Name of file for CLI commands.

Consoli Solutions. Copyright (c) 2024

**-ip, -id, -pw:** Only required if you intend to make changes to a fabric. Otherwise, a captured data file must be specified with “-i”

**-fid:** This is the Fabric ID of the logical switch on the chassis pointed to by the login credentials where the zoning changes are to be made.

**-wwn:** This is the fabric WWN of the fabric in the data collection (source) whose zone database you want test against. Run with –scan to determine the fabric WWNs.

**-z:** Name of workbook with zone definitions. An example is next.

**-sheet:** The name of the worksheet in the workbook specified with –z to use for zone definitions.

**-a, -save:** Equivalent to cfgenable. Keep in mind that if you modified the active zone configuration, you will need to use this parameter to re-enable the zone configuration. Changes are automatically saved when enabling a zone configuration so there is no need to specify –save (equivalent to cfgesave). An error will occur and no zone changes saved to the switch if the zone configuration is not in the zone

database of the fabric specified with the `-wwn` option.

**-strict:** Warnings are treated as errors. Warnings are for inconsequential errors such as deleting a zone that doesn't exist. Although optional, this should be set when working on production fabrics.

**-scan:** Output is by chassis. Each logical switch, along with the fabric WWN it belongs to and the names of the zone configurations. `-scan` works on the logical switch if login credentials are provided and the `-l` file if provided.

**-cli:** Generates FOS zoning commands. This may be useful when the people implementing the zoning do not have access to SANnav or the API.

#### Additional Notes:

**-cli:** I use it for assisting customers who are not using the API. In that case, I've done the analysis and just giving the customer CLI commands to implement the zone changes. This can also be useful for system integrators and value added resellers, but please keep in mind we need to discuss licensing if this script is to be used for service delivery.

## zone\_config.py:

	Zone_Object	Action	Name	Match	Member	Principal Member	Comments
→	alias	purge	server_alias_(7 8)	regex_m			
→	zone	full_purge	zone_2				
→	alias	ignore	storage_alias_*	wild			

See zone\_config\_sample.xlsx for an example. Also works on peer zones.

Consoli Solutions. Copyright (c) 2024

The easiest way to explain how the worksheet works is with examples. This is the worksheet specified with the -sheet option discussed on the previous slide. I suggest making a copy of zone\_config\_sample and clear the cell contents without deleting them so that the pull down menus are preserved.

Assume I want to decommission a server cluster whose HBAs are identified by aliases beginning with server\_alias\_7 and server\_alias\_8.

The first thing I'm going to do is select "alias" for the Zone Object. As you can see, the other zone object options are zone\_cfg, peer\_zone, and zone.

The next thing I'm going to do is select the "purge" action. Other actions are create, add\_mem, delete and remove\_mem. Those are pretty straight forward, so I'm going to limit this discussion to purge, full\_purge, and ignore.

I'll use a ReGex syntax in the name for these servers.

Cells in the "Match" column are used to tell the script how to interpret the name. In this case I'm using a RegEx match. Other options are exact (which is the default if the

cell is empty), wild (for wild card matching), and RegEx searching.

So in this first row, any alias beginning with `server_alias_7` and `server_alias_8` will be removed from any zone they are used in and then the aliases will be deleted. The script then counts all remaining members that are not in the ignore list. If the count is zero, the zone will be purged. More on the ignore list shortly. Purging a zone means it will be removed from any zone configuration it is used in and then deleted.

Empty rows are ignored.

The next row is an example of how to purge an entire zone. Since `"full_purge"` is selected, all members of the zone that are aliases will be purged, then the zone is purged as we just discussed. If I selected `"purge"` instead of `"full_purge"`, the zone would be purged but not the members. That means the zone or zones will be removed from any zone configuration they are used in and deleted, but no action would be taken on the alias members.

In the first example, I mentioned an ignore list. With simple single initiator zoning, only storage will be left in the zone, so I'll still want to purge the zone. Although servers and storage can usually be determine, there is no reliable way to differentiate server logins from storage logins all the time. The ignore action alleviates this problem by creating a list of members to ignore when counting the membership list. I typically run the script in test mode without specifying any members to ignore, so that I can use the script output to tell me what members of zones are preventing the zone from being deleted. Script output is discussed in a few more slides.

Purge works on peer zones as well. The only difference is in the way a peer zone is deemed no longer needed. Instead of just checking for members that are not in the ignore list, it checks for any regular members and target members. If either membership list is empty, the zone is purged.

## zone\_config.py: Example

### Typical command line to scan fabrics:

```
py zone_config.py -ip 10.144.72.15 -id admin -pw password  
-i _capture_2024_05_15_08_00_46/combined  
-log _logs -scan
```

### Typical command line to test against previously collected data:

```
py zone_config.py  
-i _capture_2024_05_15_08_00_46/combined  
-wwn 10:00:c4:f5:7c:16:8b:3e -z zone_sample  
-sheet zone -log _logs
```

Consoli Solutions. Copyright (c) 2024

These are here for copy, paste, and substitute values for your own environment. Login credentials are not required with the `–scan` option but recommended if the chassis is available.

### Additional Notes:

**-i:** Previously captured data. Typically the project data created when running `multi_capture`. This option allows you to validate zoning changes from previously collected data rather than a live switch.

**-scan:** The input file is read and, if login credentials were provided, the switch. This is used to interrogate the contents of the input file.

**-wwn:** Only used with `–i`. This is the fabric WWN for the fabric in the project you want to validate zoning changes against. This is why the `–scan` option is useful.

**-z:** The name of the workbook discussed on the previous slide

**-sheet:** The name of the worksheet to read in the workbook specified with `–z`.

## zone\_config.py: Example

**Typical command line to configure zoning on a live switch:**

```
py zone_config.py -ip 10.144.72.15 -id admin -pw password  
-fid 1 -z zone_sample -sheet zone  
-a zonecfg_0 -strict -log _logs
```

Consoli Solutions. Copyright (c) 2024

The only difference between testing and actually configuring zoning on a switch is that you will always use login credentials and a FID, -fid. You may also enable a zone configuration with the -a option. A test is always performed before attempting changes. The zone database is always validated before attempting to make any changes on the switch. No changes are made if there any errors. Note that since activating a zone also saves the changes, there is no need to specify -save.

## zone\_config.py: Ports Effected by Purge

Ports Effected By Purge (switch name followed by ports)

```
REPSW1 FID: 0 DID: 1 (10:00:88:94:71:7e:74:20)
0/31: STG_AZ123_8B
0/23: STG_AZ123_8K
0/10: SVR_REP789DS09AS_3E98_01
0/20: SVR_REP789DS09BS_3EB8_01
0/14: SVR_REP789AS09BQ_D32A_01
```

**Switch name**

**Port number followed by  
alias of login**

```
REPSW2 FID: 0 DID: 3 (10:00:88:94:71:7c:0c:20)
0/12: STG_AZ456_8B
0/36: STG_AZ456_8K
0/11: SVR_REPA789DS09DS_D2FE_01
0/22: SVR_REPA789DS09ES_D31E_01
0/13: SVR_REPA789AS09AQ_3EAE_01
0/10: SVR_REPA789DS09X_6B2A_01
```

Consoli Solutions. Copyright (c) 2024

There are three summaries in the output. The first is the “Ports Effected By Purge” section. This report is intended for cable teams that need to know specific port numbers when migrating applications or decommissioning devices. If no purge actions were used in the workbook, this section of the output will be empty.

## zone\_config.py: Error Output

Purge Faults (zone name followed by remaining members)

Sever_1_storage_1_zone	<b>Zone name</b>
storage_2_alias	<b>Member not purged (remaining member)</b>

Purge Faults (members only)

storage_2_alias	<b>For easy copy and paste</b>
-----------------	--------------------------------

Error Detail

Could not complete purges. Zone members or aliases in the zone members ...

Warning Detail

None

Ports Effected By Purge (switch name followed by ports)

None

Consoli Solutions. Copyright (c) 2024

Next is a summary of purge issues encountered.

**Purge Faults:** Contains all of the zones that could not be purged. In this example, since the only remaining member of Sever\_1\_storage\_1\_zone is a single storage alias, you will likely just add this alias to the list of aliases to ignore.

**Purge Faults (members only):** This is redundant. The only difference is that it's just a list of members for easy copy and paste into the workbook.

**Error Detail:** includes additional detail about the zone and zone members that are listed in the "Purge Faults"

**Waning Detail:** If --strict was not used, these are all the warning.

If no purge actions were used in the workbook, this section of the output will be empty.



## zone\_config.py: Validation Output

What happened?      What Object?      Where?

Invalid alias name, \_bad\_alias name, at row 2

Principal members only supported in peer zones at row 9  
Principal members not supported in zone configurations at row 18  
Invalid member name: 00:06:04:82:d5:31:d4:01 in alias bad\_alias\_member. Rows: 3  
Zone single\_member\_zone has 1 members. Rows: 10  
Zone no\_member\_zone has no members. Rows: 11  
Zone no\_member\_zone has 0 members. Rows: 11  
Zone peer\_zone\_missing\_principal has no peer members. Rows: 13, 14  
...  
17 Errors  
Outstanding zone changes not saved

Display all related rows.

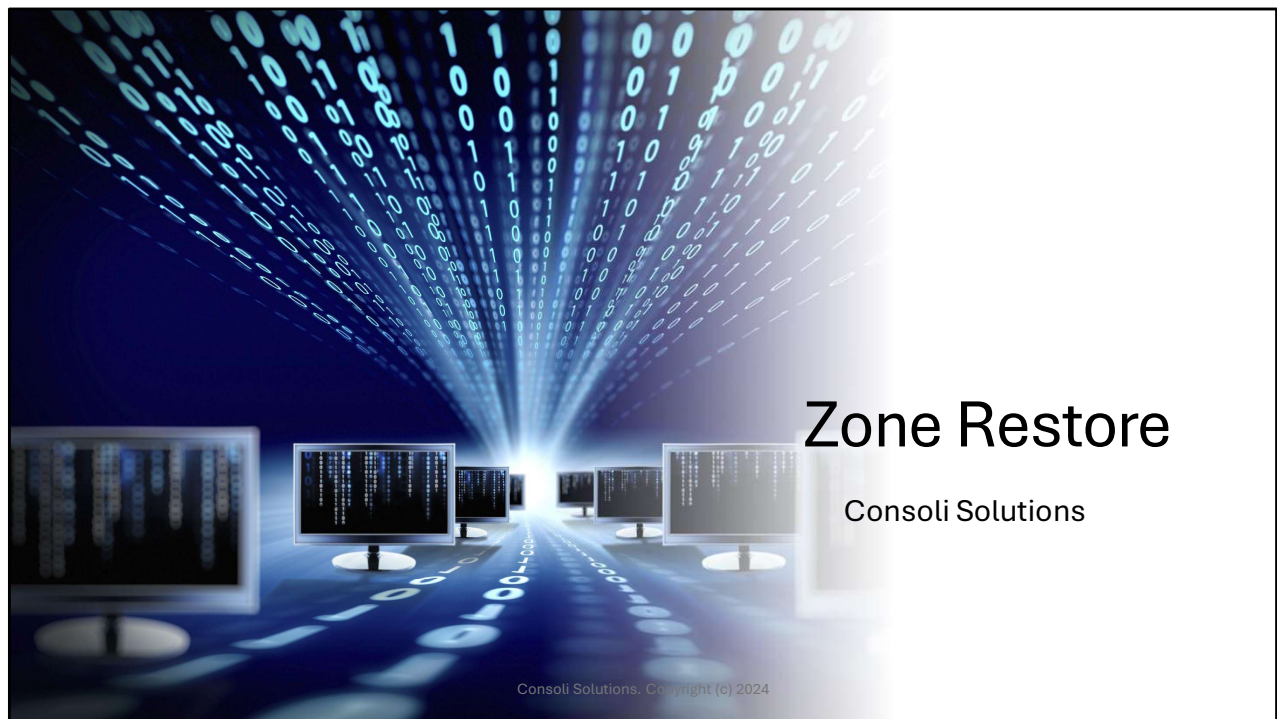
One of my pet peeves with validation software is that it often stops checking after the first fault and provides little insight as to what or where the fault is. This utility continues to validate each row in the workbook after an error is encountered. The output includes the row number(s) as well as a brief explanation of what the problem is. This eliminates the need to correct one problem at a time.

### Additional Notes:

\_bad\_aliasname – Bad because zoning object names cannot begin with a special character. In this case, the leading underscore.

bad\_alias\_member - 00:06:04:82:d5:31:d4:01 is an invalid WWN because it begins with 00

Understanding other errors would require a deeper review of the workbook. The point here isn't to discuss zoning errors, it's simply to provide an example of error checking.



Zone restore is a completely replaces the current zone database.

## zone\_restore.py

- Restore from a previously captured data
  - The switch type doesn't matter.
- The same as restore.py -p z,ze
  - Simplified Brocade FOS API: Configuring Switches
- Only difference is the -cli option
  - Generates CLI commands needed to restore the zone database.
- Only makes zoning changes.
  - Eliminates making a mistake that makes other changes.
  - Fewer command line options.

Consoli Solutions. Copyright (c) 2024

Reach out to me directly if you need to restore a zone database from a Cisco SAN.

### Additional Notes

Restores the zone database from a previous data capture. It can be any previous data capture. If you capture data regularly, you can use that data as a point in time zone database backup.

The zone\_restore.py script does the same thing as restore.py, see "Simplified Brocade FOS API: Configuring Switches", when using just z and ze with the -p parameters. This one is a little easier to use and, since it can't do anything other than make zone changes, it's less likely to make mistakes that would cause other configuration changes.

The only significant difference is that this script has an option to generate the equivalent CLI commands.

## zone\_restore.py: Input Parameters

Parameter	Description
-ip, -id, -pw	Login credentials
-fid	Optional. Required when -i is not specified. Fabric ID of logical switch.
-i	Required. Output of capture.py, multi_capture.py, or combine.py.
-wwn	Optional with -scan. Otherwise, required. Fabric WWN for the source zone database in the file specified with -i.
-a	Optional. Specifies the zone configuration to activate. If not specified, no change is made to the effective configuration. If a zone configuration is in effect and this option is not specified, the effective zone may result in the defined zone configuration being inconsistent with the effective zone configuration.
-scan	Optional. No parameters. Scan fabric information. No other actions are taken.
-cli	Optional. Name of file for CLI commands.

Consoli Solutions. Copyright (c) 2024

Most of these inputs are the same as with zone\_config.py. I'm not going to go through them. They are here for reference.

### Additional Notes

The usual login credentials. -log, -sup-, -d, and -nl as discussed in "Simplified Brocade FOS API: Getting Started" are also supported.

**-wwn:** This is the fabric WWN of the fabric in the data collection (source) whose zone database you want to restore to the switch specified by the login credentials (target). The fabric WWN is necessary because there may be multiple fabrics in a data collection. The only way to ensure unique identification of a fabric is with the fabric WWN. See -scan for help obtaining the fabric WWN.

**-a:** Equivalent to cfgenable. Keep in mind that if you modified the active zone configuration, you will need to use this parameter to re-enable the zone configuration. Changes are automatically saved when enabling a zone configuration so there is no need to specify -save (equivalent to cfgesave). An error will occur and no zone changes saved to the switch if the zone configuration is not in the zone

database of the fabric specified with the `-wwn` option.

**-scan:** Output is by chassis. Each logical switch, along with the fabric WWN it belongs to and the names of the zone configurations. `-scan` works on the logical switch if login credentials are provided and the `-l` file if provided.

**-cli:** Generates FOS zoning commands. I use it for assisting customers who are not using the API. In that case, I've done the analysis and just giving the customer CLI commands to implement the zone changes. This can also be useful for system integrators and value added resellers, but please keep in mind we need to discuss licensing if this script is to be used for service delivery.

## zone\_restore.py: Execution Example

### Typical command line to scan fabrics:

```
py zone_restore.py
-i _capture_2024_05_15_08_00_46/combined
-log _logs -scan
```

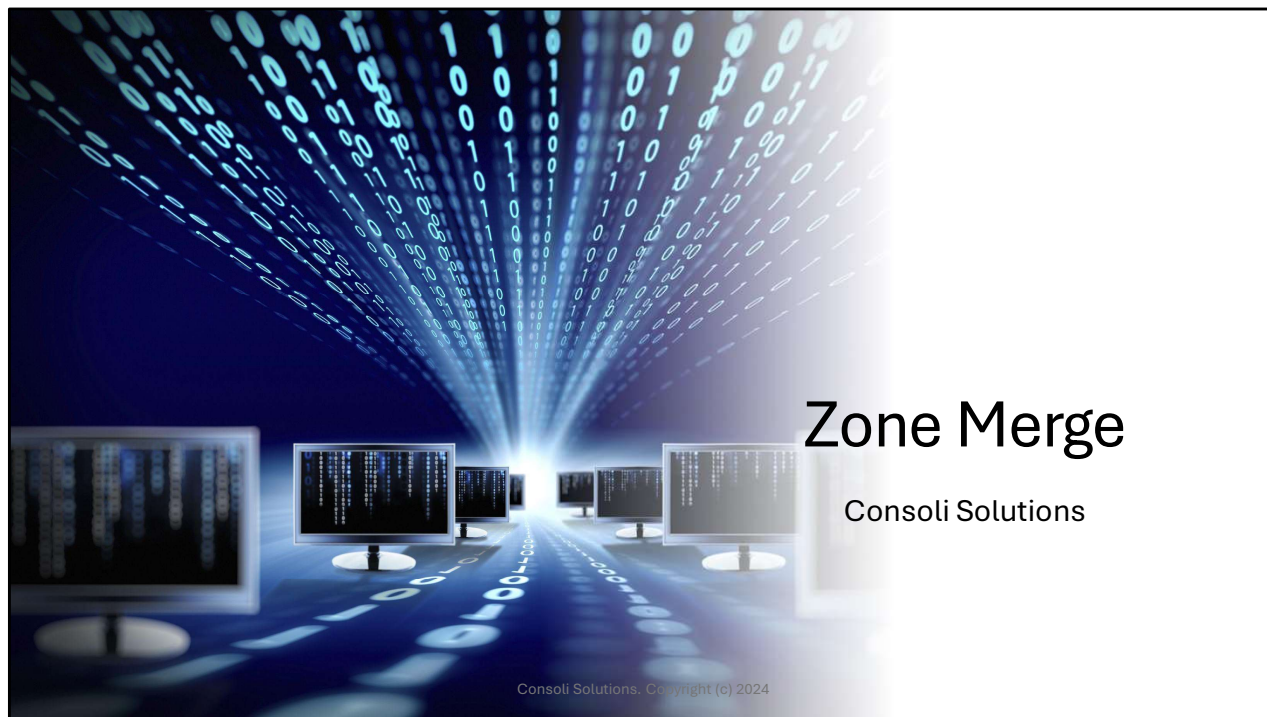
### Typical command line to restore a zone configuration:

```
py zone_restore.py -ip 10.144.72.15 -id admin
-pw password -i _capture_2024_05_15_08_00_46/combined
-fid 1 -wwn 10:00:c4:f5:7c:16:8b:3e -a zonecfg_0
-log _logs
```

Consoli Solutions. Copyright (c) 2024

-scan works the same as with the zone\_config.py script. This time, I didn't include the login credentials. If you specify login credentials, the live chassis will also be included in the scan report.

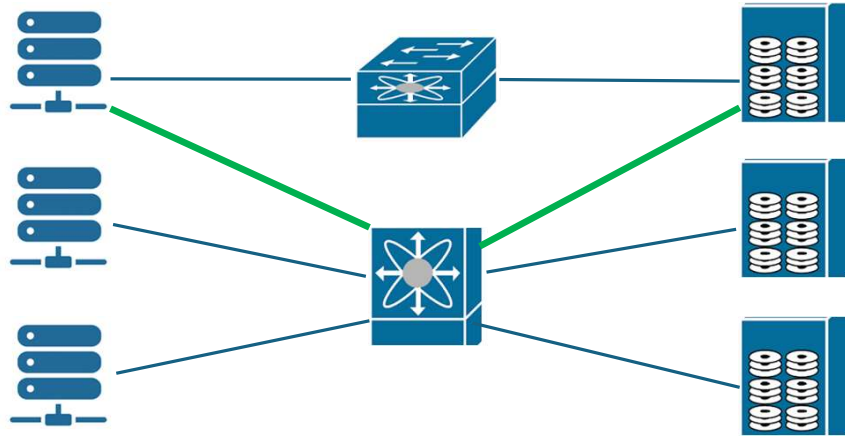
This intended for copy, paste, and modify to suit your needs.



The zone merge utility is used primarily to:

1. Determine if there is anything that would prevent merging zone databases.
2. Simplify zone merges.
3. Minimize zone freeze windows, typically to just seconds.

## zone\_merge.py Use Case: Migrate to New Fabric Without Merging The Fabrics



Consoli Solutions. Copyright (c) 2024

Sometimes, there is a need to merge zone databases without merging the fabrics. This is typical of SANs with older equipment to move of or incompatible switch types. Reach out to me directly if you need to merge the zone databases from a Cisco fabric to a Brocade fabric.

### Additional Notes

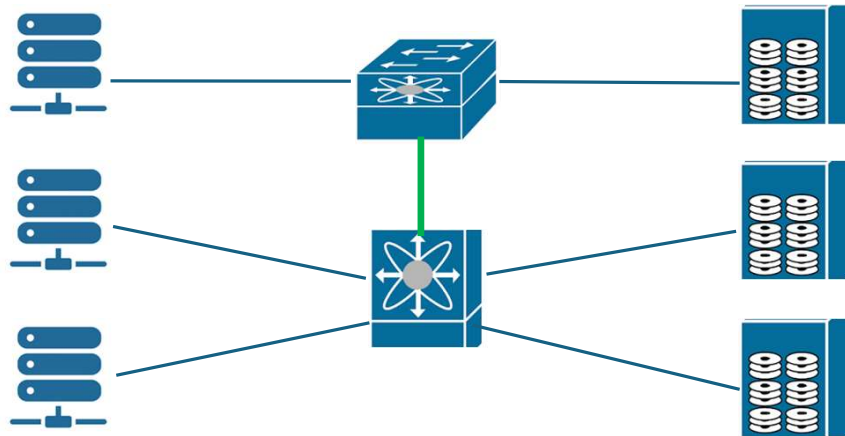
The most common use cases are:

- Migrating servers and storage off a SAN without having to merge the fabrics. The common scenarios for merging zone databases without merging the fabrics are:
  - Not all servers and storage will be migrated and isolation from servers and storage not being migrated is desired. In this case, you'll need to clean up the aliases and zones that will not be used in the new fabric.
  - You did a two generation switch upgrade (Gen5 to Gen7 for example), the older switch is on code that is not compatible with the code on the new switch, and no one wants to make any changes to the older switch.
  - You are migrating off a Cisco SAN fabric. Tools to do this are not discussed in this video. Reach out to me if you need to do that.



- A SAN consolidation when isolation between the old fabric and new fabric is desired.
  - Typically multiple fixed port switches to a director class switch.

## zone\_merge.py Use Case: Merge Fabric



Consoli Solutions. Copyright (c) 2024

This is the more common scenario. It's typical of SAN switch upgrades that had to happen in pieces, merging of prod and dev environments, and SAN consolidations. Don't forget that there are other fabric parameters to consider: unique DIDs, unique switch names, time out values, etc. when merging fabrics.

### Additional Notes

The `zone_merge.py` script only merges the zone database. While this is typically all that is needed, keep in mind that all other fabric parameters must match, including the Fabric ID (FID). The domain ID and switch names must be unique. Although usually not modified, keep in mind that all other fabric parameters such as addressing mode and time out values must match.







When merging fabrics, the zone database must be updated in all fabrics.

The most common use cases for this are:

- Sharing storage resources
  - Upgrading storage with higher speed and capacity

- Applications were migrated off the SAN and storage is being consolidated.
- Slow migration off older SAN switches but didn't merge the two initially
  - Sometimes customers built a new SAN, thinking the old one would go away and later realize the SFPs in the new SAN won't support the older storage.
  - Or they just realized they can't get off the older storage as soon as they thought they could.
- Merge a development environment into a production environment.

## zone\_merge.py: Input Parameters

*Option	Description
 -i	Required. Workbook with zone merge instructions. See zone_merge_sample.xlsx for details. ".xlsx" is automatically appended.
 -cfg	Optional. Typically used. The specified zone configuration is a merge of all zone configuration defined in the workbook specified with the -i parameter. If the zone configuration does not exist, it is created in all fabrics where the "Update" column in the workbook is "Yes". When this option is not used, only the aliases and zones are copied.
 -a	Optional. No parameters. Activates the zone configuration specified with the -cfg option.
 -t	Optional. No parameters. Perform the merge test only. No fabric changes are made.
 -scan	Optional. No parameters. Scan fabric information for data capture files and IP addresses specified in the workbook specified with the -i option. No other actions are taken.
 -cli	Optional. No parameters. Prints the zone merge CLI commands to the log and console whether -t is specified or not.

Consoli Solutions. Copyright (c) 2024

This slide is primarily for reference. The only thing I want to highlight on this slide is that by default, zone\_merge.py will take action unless -t, test mode, is explicitly specified on the command line. This is backwards from any other scripts I wrote. I only noticed it when putting this presentation together. Changing this is a future considerations. See the handout notes for additional comments.

A pre-flight check is performed. No fabric changes are made if the pre-flight check fails.

**-i:** This is the workbook discussed on the next slide.

**-cfg:** Typically, the active zone configurations are merged. These are specified in the cells in the "cfg" column as discussed on the next slide. This parameter names the zone configuration that is the result of the merge. If it doesn't exist, it will be created.

**-a:** Effectively a FOS cfgenable command. If not specified, the zone database is saved. Effectively a FOS cfsave command.

**-t:** As previously mentioned, by default this should have been in test mode. With this

module, you must specify `-t` to put it in test mode only.

**-cli:** As with `zone_config.py`, this is useful when the implementation team won't be using the API for configuration work.

## zone\_merge.py: File Specified With -i

Use a project file rather than login live

user_id	pw	ip_addr	security	fid	fab_wwn	cfg	update	project_file
					10:00:d8:1f:cc:1d:e0:60	Zoneconfig	No	_capture_2021_07_17_08_24_19/sw_115
admin	xx	x.15	self	128		Brocade_zCFG	No	
admin	xx	x.15	self	30		Zonecfg_0	Yes	

Logical switches can be in the same chassis. This is useful for collapsing multiple logical switches into one.

Consoli Solutions. Copyright (c) 2024

See zone\_merge\_sample.xlsx which is included in the github/jconsoli package. There is an “Instructions” sheet that details what each of these columns are. There is no limit as to the number of zone databases that can be merged.

Columns:

**“user\_id”, “pw”, “ip\_addr”, and “security”:** are the login credentials for each switch. Alternatively you can use a previous data capture.

**fab\_wwn:** As with zone\_config.py, a fabric WWN is required identify which fabric to use when using a previous data capture.

**cfg:** Optional. Specifies the zone configuration to be merged with the zone configuration specified with the -cfg option in the command line when launching zone\_merge.py. Ignored if -cfg is not specified. If -cfg is specified and this cell is empty, this fabric is not included in the merged zone configuration. All aliases and zones are copied, it’s just the zone configuration that’s not merged.

**update:** If yes, the result of the merged zone database are pushed to the switch.

Updates can be sent to multiple fabrics if the intent is to actually merge the SAN fabrics.

**project\_file:** This is the same project file discussed with zone\_config.py.

## zone\_merge.py: Execution Example

**Typical command line to merge zone databases, create a new zone configuration, new\_zone\_cfg, and enable the new zone database:**

```
py zone_merge.py  
  -i zone_merge_sample -cfg new_zone_cfg -a -log _logs
```

Consoli Solutions. Copyright (c) 2024

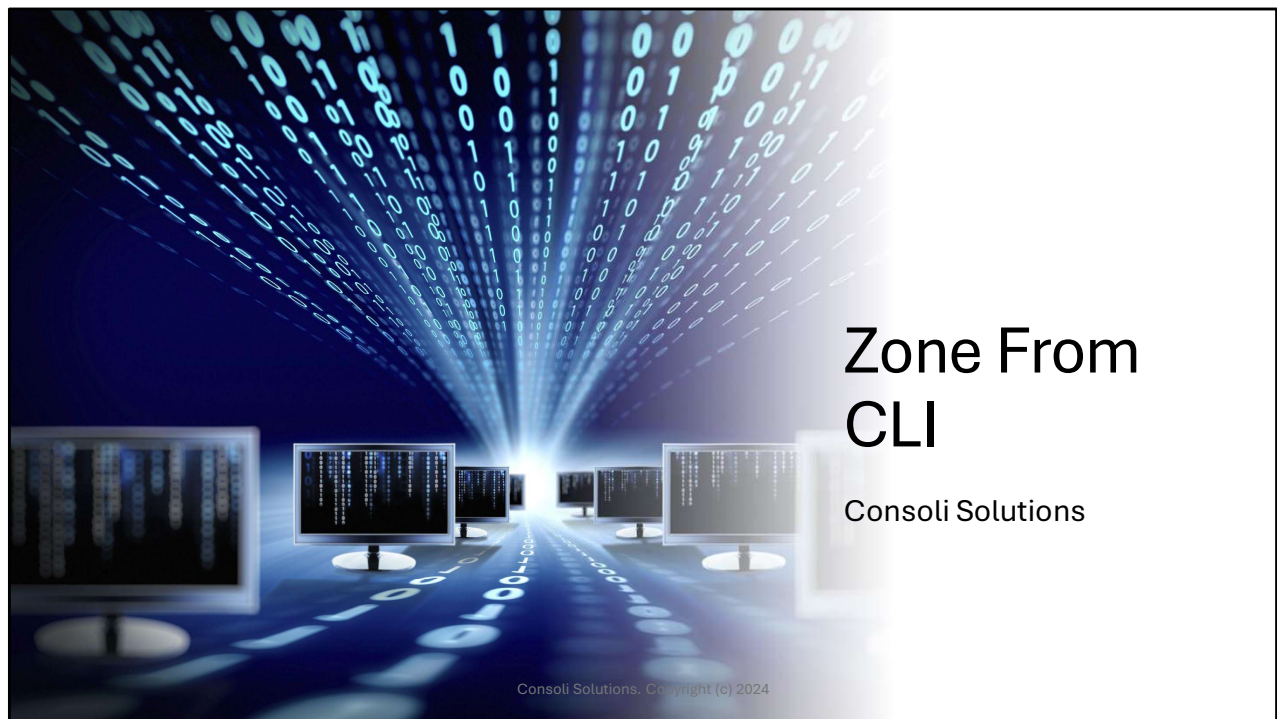
**-i:** This is the workbook we just discussed.

**-cfg:** If “new\_zone\_cfg” already exists, it will be added to. Otherwise, it will be created.

**-a:** Activates (enables) new\_zone\_cfg.

If an error occurs, no changes are made on the switches.





This is worth just a quick mention.

## cli\_zone.py

- Zone from a plain text file containing CLI zoning commands
- Handles all zone commands except:
  - zoneobjectexpunge, zoneobjectreplace, cfgdisable
- For use with existing scripts that generate CLI
- Advantages
  - Validates all commands in the input file
  - Has a test mode to validate zoning before implementation
  - Much faster than executing commands in an SSH session
- Batch process common actions

Consoli Solutions. Copyright (c) 2024

SAN engineers often have scripts that generate CLI or perhaps they got a CLI file from the SAN Health team. It's much faster than zoning from CLI scripts and using this script has the added benefit of validating zoning changes in advance. As with the other zoning scripts, it validates the entire file. If there are errors, all zoning changes are aborted.

Batch process zone operations. For example, during end of quarter processing you may want to launch a script for each fabric to use a zone configuration that gives more storage access to certain servers.

It may also be useful to developers.



Thank you  
for watching

Consoli Solutions

Consoli Solutions. Copyright (c) 2024