Developer Days
Network Services Orchestrator

DeMystifying NSO Commit Queues

Understanding NSO Commit Queues

Dan Sullivan
Principal Architect
06/16/2020

Abstract

NSO Commit Queues are used in many deployments and customers frequently ask questions regarding the settings

The purpose of this presentation is to examine commit queues and provide explanation and demonstration of a few key features and concepts

Agenda

- Overview of NSO commit queues
- Commit Queue Key Features & concepts
- Commit Queue Scenarios
  - Sync vs. Async API calls
  - ATOMIC Transactions
  - Overlapping Configuration
  - Automatic Error Recovery
  - Transaction bundling
- Questions

NSO Commit Queue Overview

What are NSO commit queues and why are they used?

- NSO feature in which operators can ”decouple” the CDB transaction from the network transaction
- Typically used for those operations who need to increase NSO transactional throughput
- Large numbers of devices need to be modified
- Quite a bit of flexibility with NSO commit queues needed to support a variety of use cases

NSO Commit Queues

- Default transaction mode requires NSO to keep a transaction open until all devices have consumed the specified configuration changes
- NSO commit queues “decouple” the CDB transaction from programming the changes in the set of devices defined by the transaction
- CDB transaction is often characterized in milliseconds while the device transaction time can be on the order of minutes
- Allow for higher transactional throughput.
  - NSO CDB transaction is completed first
  - NSO will then queue the changes on per device basis
  - Many devices can be programmed in parallel allowing for higher transactional throughput

NSO Commit Queues Options

- Default transaction mode requires NSO to keep a transaction open until all devices have consumed the specified configuration changes
- NSO commit queues “decouple” the CDB transaction from programming the changes in the set of devices defined by the transaction
- CDB transaction is often characterized in milliseconds while the device transaction time can be on the order of minutes
- Allow for higher transactional throughput.
  - NSO CDB transaction is completed first
  - NSO will then queue the changes on per device basis
  - Many devices can be programmed in parallel allowing for higher transactional throughput

NSO Commit Queue Key Features & Concepts

NSO Commit Queue settings and options need to be considered on a per use case basis

API features: sync or async

The sync or async settings are largely an API constraint and don't really affect commit queue operations

ATOMIC vs. non-ATOMIC

Treating a transaction as ATOMIC needs to be considered as it affects the intended use case

Error Recovery

Does the use case allow for the utilizing automatic rollback on error or is manual error recovery required?

A few more key settings

Commit Queue Tags

Commit queue tag functionality allows the operator to “label” a transaction, quite useful for end-to-end visibility with commit queues

Error action

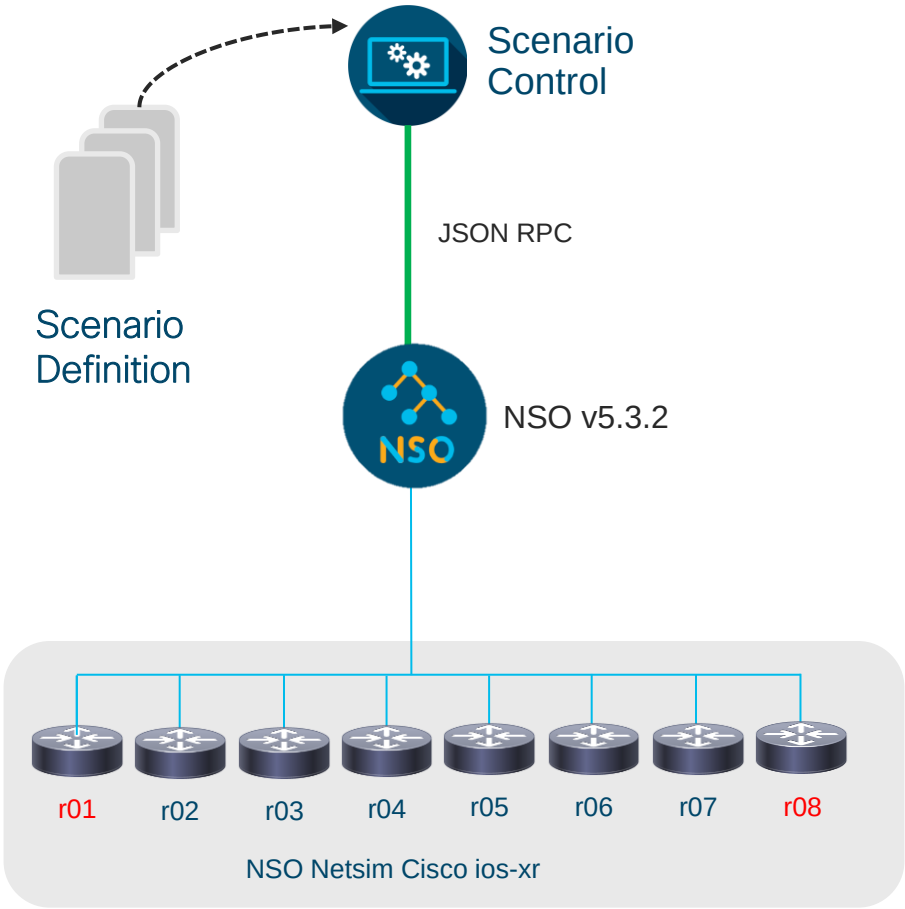
How should errors be treated stop-on-error, continue-on-error, rollback-on-error

NSO Commit Queue Scenarios

Commit Queue Scenarios

- Goal is to demonstrate all of the key features and API usage of commit queues
- Demonstrating these key features involves multiple steps done in sequence
- Needed an alternative to just simply typing on the CLI
- Created Python based scenario engine to define and execute multi-step scenarios

Setup



Setup NSO Packages

cisco-iosxr-cli-7.25 : Cisco IOS-XR NED
base-config : Base service
Hostname : Hostname service
Intf : Interface Service

*netsim devices r01, r08 are built with a modified version of the above NED

*Service packages are basic services built to help demonstrate commit queue functionality

Commit Queue Scenarios

JSON Based Scenario Definition files used to define multi step scenarios

Python based application reads and executes scenario steps

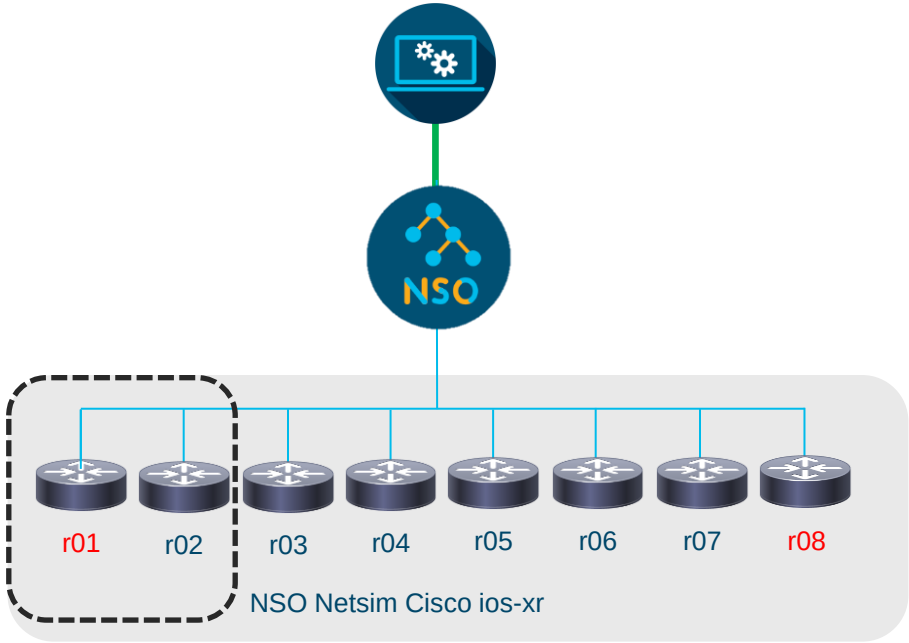
JSON RPC API used to communicate between scenario control application and NSO

NSO commit queue settings are all defaults. Each scenario utilizes commit flags

Scenario (1)

Commit Async/Sync API calls

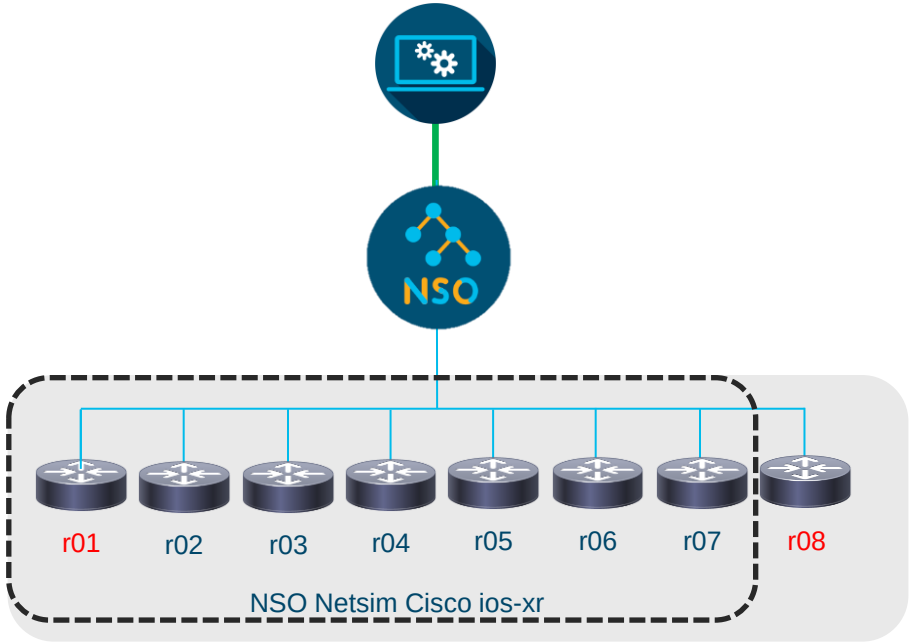
Examine usage of commit queue async and sync flags



Scenario (2)

Commit Queue Atomic

Examine usage of commit queue atomic setting

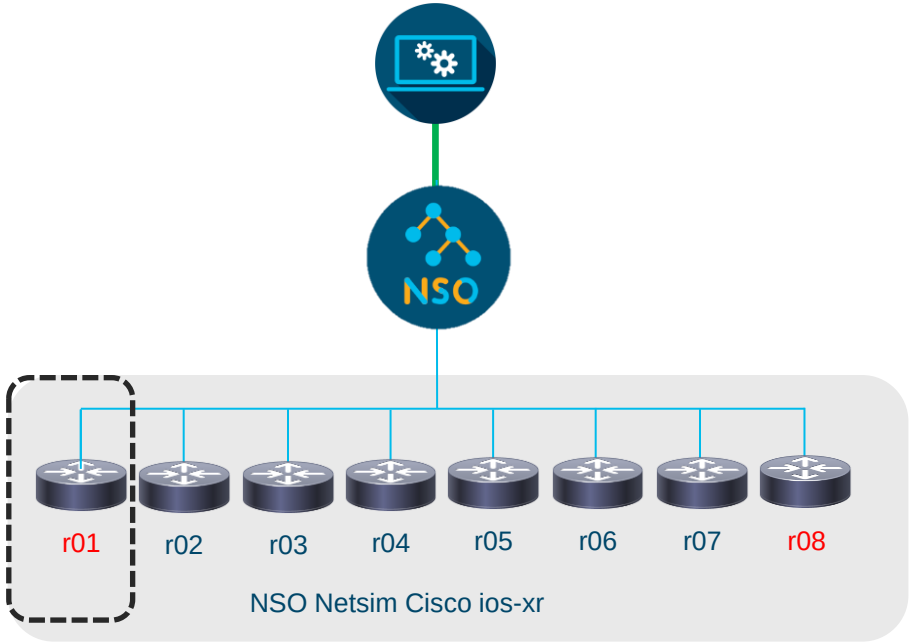


Scenario (3)

Commit Queue Overlapping Config

Why do commit queue overlap errors occur?

Run multiple scenarios with and without overlap errors



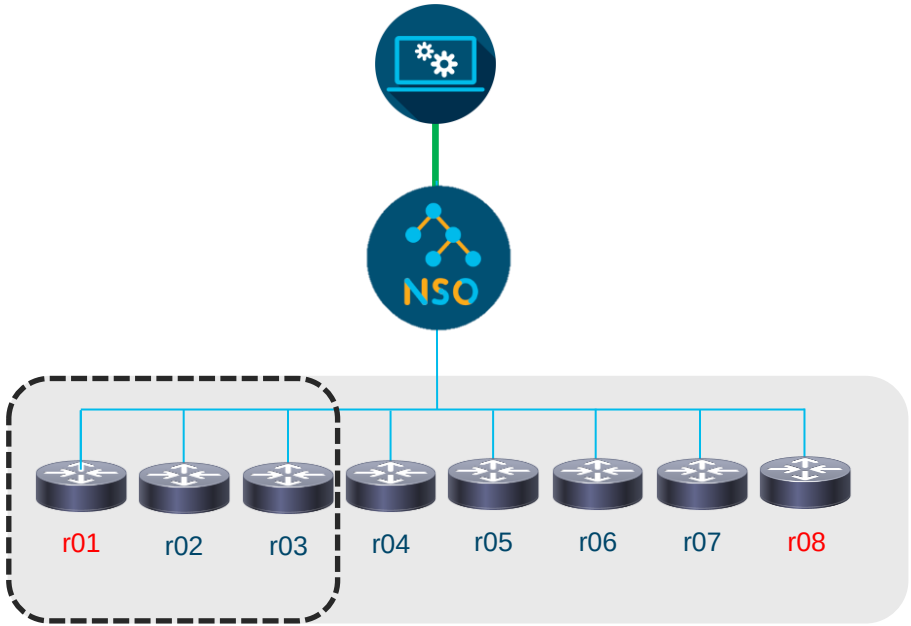
Scenario (4)

Rollback on error

Create services instances across multiple routers

Specify error recovery on commit

Induce a failure and validate the error recovery

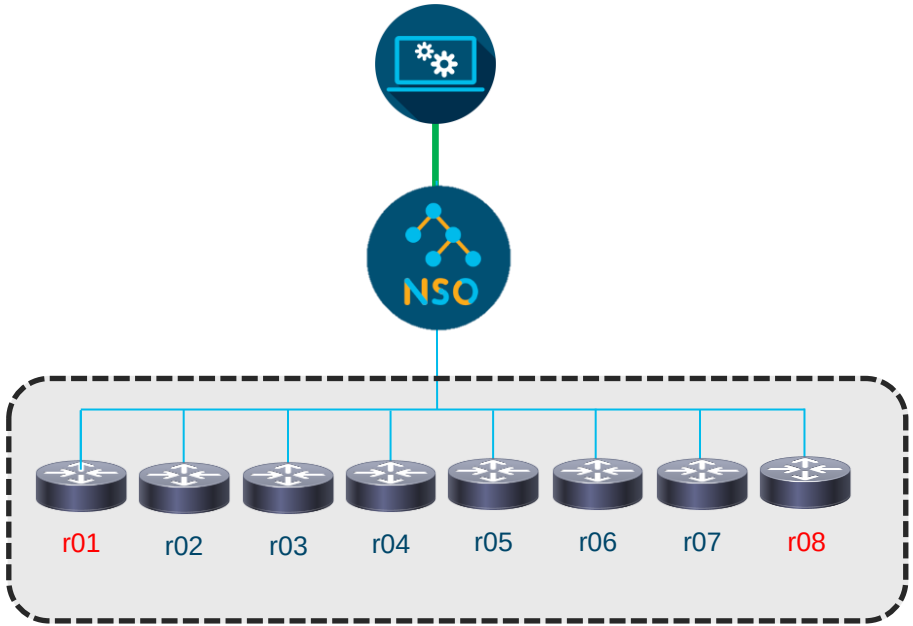


Scenario (5)

Transaction bundling

When is it useful to ”bundle” multiple services and/or device changes into a single transaction?

What is the best way to recover from an error should one occur?



Questions?


CISCO