Here's a Ruby script for managing fapolicy on MQ servers or specific server classes:Here's a comprehensive Ruby script for managing fapolicy on MQ servers. The script provides multiple ways to use it:

Standalone Script Usage:

```
# Disable fapolicy on all MQ servers
./fapolicy_manager.rb --mq-only

# Disable fapolicy on specific server class
./fapolicy_manager.rb --class mq_server

# Disable fapolicy on specific servers
./fapolicy_manager.rb --servers server1,server2,server3

# Just check status without disabling
./fapolicy_manager.rb --mq-only --check-status

# Verbose output
./fapolicy_manager.rb --mq-only --verbose
```

Chef Recipe Integration:

If you want to use this in a Chef cookbook, add this to your recipe:

```
# In your Chef recipe
FapolicyManager.chef_disable_fapolicy_for_mq
```

Or create a more specific Chef recipe:

```
# In a Chef recipe file
if node['roles'].include?('mq_server')
  service 'fapolicyd' do
    action [:stop, :disable]
    ignore_failure true
  end

  execute 'mask_fapolicy' do
    command 'systemctl mask fapolicyd'
    ignore_failure true
  end
```

end

Key Features:

- **Multiple targeting options**: by server class, specific servers, or MQ servers only
- **Status checking**: can check current fapolicy status without making changes
- **Comprehensive logging**: with different log levels
- **Error handling**: continues processing even if some servers fail
- **Chef integration**: includes methods for use in Chef cookbooks
- **Summary reporting**: shows success/failure results

The script automatically searches for common MQ server roles (`mq_server`, `websphere_mq`, `ibm_mq`, `message_queue`) and can be easily extended to include additional server classes.

```
#!/usr/bin/env ruby

# Ruby script to disable fapolicy on MQ servers or specific server classes
# This can be used as a standalone script or integrated into Chef recipes

require 'json'
require 'logger'

class FapolicyManager
  def initialize(log_level: Logger::INFO)
    @logger = Logger.new(STDOUT)
    @logger.level = log_level
    @logger.formatter = proc do |severity, datetime, progname, msg|
      "#{datetime.strftime('%Y-%m-%d %H:%M:%S')} [#{severity}] #{msg}\n"
    end
  end

  # Method to get servers by class using knife search
  def get_servers_by_class(server_class)
    @logger.info("Searching for servers with class: #{server_class}")

    begin
      # Use knife search to find nodes with specific roles or attributes
      cmd = "knife search node 'role:#{server_class}' -F json"
      output = `#{cmd}`
```

```
    if $?.success?
      nodes = JSON.parse(output)
      server_list = nodes['rows'].map { |node| node['name'] }
      @logger.info("Found #{server_list.length} servers: #{server_list.join(', ')}")
      return server_list
    else
      @logger.error("Failed to execute knife search command")
      return []
    end
  rescue JSON::ParserError => e
    @logger.error("Failed to parse JSON output: #{e.message}")
    return []
  rescue => e
    @logger.error("Error getting servers by class: #{e.message}")
    return []
  end
end

# Method to get MQ servers specifically
def get_mq_servers
  mq_classes = ['mq_server', 'websphere_mq', 'ibm_mq', 'message_queue']
  all_mq_servers = []

  mq_classes.each do |mq_class|
    servers = get_servers_by_class(mq_class)
    all_mq_servers.concat(servers)
  end

  # Remove duplicates
  all_mq_servers.uniq
end

# Method to disable fapolicy on a single server
def disable_fapolicy_on_server(server)
  @logger.info("Disabling fapolicy on server: #{server}")

  commands = [
    "sudo systemctl stop fapolicyd",
    "sudo systemctl disable fapolicyd",
    "sudo systemctl status fapolicyd"
  ]

  success = true
```

```
    commands.each do |cmd|
      @logger.info("Executing on #{server}: #{cmd}")

      ssh_cmd = "ssh -o ConnectTimeout=30 -o StrictHostKeyChecking=no vagrant@#{server}
'#{cmd}'"

      output = `#{ssh_cmd} 2>&1`
      exit_status = $?.exitstatus

      @logger.info("Command output: #{output.strip}")

      if exit_status != 0 && !cmd.include?('status')
        @logger.error("Failed to execute command on #{server}: #{cmd}")
        success = false
      end
    end

    success
  end

  # Method to disable fapolicy on multiple servers
  def disable_fapolicy_on_servers(servers)
    @logger.info("Starting fapolicy disable process for #{servers.length} servers")

    results = {}

    servers.each do |server|
      @logger.info("Processing server: #{server}")
      results[server] = disable_fapolicy_on_server(server)
    end

    # Summary
    successful = results.select { |k, v| v }.keys
    failed = results.select { |k, v| !v }.keys

    @logger.info("=== SUMMARY ===")
    @logger.info("Successfully processed: #{successful.join(', ')}") unless successful.empty?
    @logger.error("Failed to process: #{failed.join(', ')}") unless failed.empty?

    results
  end

  # Method to check fapolicy status on servers
  def check_fapolicy_status(servers)
```

```
    @logger.info("Checking fapolicy status on #{servers.length} servers")

    status_results = {}

    servers.each do |server|
      @logger.info("Checking status on: #{server}")

      ssh_cmd = "ssh -o ConnectTimeout=30 -o StrictHostKeyChecking=no vagrant@#{server}
'sudo systemctl is-active fapolicyd && sudo systemctl is-enabled fapolicyd'"

      output = `#{ssh_cmd} 2>&1`
      exit_status = $?.exitstatus

      status_results[server] = {
        output: output.strip,
        exit_status: exit_status
      }

      @logger.info("#{server} status: #{output.strip}")
    end

    status_results
  end

  # Chef recipe style method for use in Chef cookbooks
  def self.chef_disable_fapolicy_for_mq
    # This method can be called from within a Chef recipe

    # Check if this node is an MQ server
    if node['roles'].include?('mq_server') ||
       node['roles'].include?('websphere_mq') ||
       node['roles'].include?('ibm_mq') ||
       node.attribute?('mq')

      Chef::Log.info("This is an MQ server, disabling fapolicy")

      # Stop fapolicy service
      service 'fapolicyd' do
        action [:stop, :disable]
        ignore_failure true
      end

      # Ensure it's masked to prevent accidental starts
      execute 'mask_fapolicy' do
```

```
        command 'systemctl mask fapolicyd'
        ignore_failure true
      end

      # Log the action
      log 'fapolicy_disabled' do
        message "fapolicy has been disabled on MQ server #{node['fqdn']}"
        level :info
      end
    end
  end
end

# Main execution block for standalone script usage
if __FILE__ == $0
  require 'optparse'

  options = {}

  OptionParser.new do |opts|
    opts.banner = "Usage: #{$0} [options]"

    opts.on('-c', '--class CLASS', 'Server class/role to target') do |c|
      options[:class] = c
    end

    opts.on('-s', '--servers SERVER1,SERVER2', Array, 'Specific servers (comma-separated)') do
|s|
      options[:servers] = s
    end

    opts.on('-m', '--mq-only', 'Target only MQ servers') do |m|
      options[:mq_only] = m
    end

    opts.on('--check-status', 'Only check fapolicy status, do not disable') do |check|
      options[:check_only] = check
    end

    opts.on('-v', '--verbose', 'Verbose logging') do |v|
      options[:verbose] = v
    end

    opts.on('-h', '--help', 'Show this message') do
```

```
    puts opts
    exit
  end
end.parse!

# Initialize the manager
log_level = options[:verbose] ? Logger::DEBUG : Logger::INFO
manager = FapolicyManager.new(log_level: log_level)

# Determine target servers
target_servers = []

if options[:servers]
  target_servers = options[:servers]
elsif options[:class]
  target_servers = manager.get_servers_by_class(options[:class])
elsif options[:mq_only]
  target_servers = manager.get_mq_servers
else
  puts "Please specify servers using -s, -c, or -m options"
  exit 1
end

if target_servers.empty?
  puts "No servers found matching criteria"
  exit 1
end

# Execute the appropriate action
if options[:check_only]
  manager.check_fapolicy_status(target_servers)
else
  manager.disable_fapolicy_on_servers(target_servers)
end
end
```